

Elements of Data Science, Machine Learning, and Artificial Intelligence Using R

Frank Emmert-Streib • Salissou Moutari •
Matthias Dehmer

Elements of Data Science, Machine Learning, and Artificial Intelligence Using R

Frank Emmert-Streib
Tampere University
Tampere, Finland

Salissou Moutari
Queen's University Belfast
Belfast, UK

Matthias Dehmer
Swiss Distance University of Applied
Science
Brig, Switzerland

ISBN 978-3-031-13338-1 ISBN 978-3-031-13339-8 (eBook)
<https://doi.org/10.1007/978-3-031-13339-8>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Digitalization of all areas of science, industry, and society has led to an unprecedented flood of data. However, after the initial enthusiasm about an anticipated wealth of information, it has been realized that in most cases this information is deeply buried inside the data, and it needs to be uncovered. This requires an analysis of the data, which is usually non-trivial and often challenging. All these developments led to the establishment of the field *data science*.

Data science is a field combining methods and approaches from machine learning, artificial intelligence, and statistics. This makes it inherently interdisciplinary, leveraging scientific approaches to derive valuable insights from data. A key for the success of data science is that it centers an analysis around *data*. This allows to move away from making theoretical assumptions, which dictate the analysis of a problem toward *data-driven* approaches. As a consequence, this requires often nonparametric approaches, which rely on computational implementations. However, in general, data science encompasses a strong computational component, enabling to put the theoretical concepts into practice. For this reason, in this book, we provide many examples for such implementations using the statistical programming language R.

Our motivation for writing this book arose out of our experience over many years. From teaching, supervising, and conducting scientific and industrial research, we realized that many students and scientists are struggling to understand the underlying concepts of methods from data science, because these were derived from a variety of fields, including machine learning, artificial intelligence, and statistics. For this reason, we present in this book the basics, core, and advanced methods with an emphasis on an understanding of the corresponding concepts. That means, we are not aiming for a comprehensive coverage of all existing methods, but we provide selected topics from data science to foster a thorough understanding of the subject. Based on these, deeper insights about all aspects of data science can be gained. This will provide a springboard for mastering advanced methods. Furthermore, we combine this with computational realizations of analysis methods using the widely used programming language R.

The present book is intended for graduate students and advanced undergraduate students in the interdisciplinary field of data science with a major in computer

science, information technology, or engineering. The book is organized into three main parts. Part I: General Topics; Part II: Core Methods; and Part III: Advanced Topics. Each chapter contains the theoretical basics and many practical examples that can be practiced side by side. This way, one can put the learned theory into a practical application and gain a profound conceptual understanding over time.

During the preparation of the present book, many colleagues have provided us with input, help, and support. In particular, we would like to thank Abbe Mowshowitz, Arno Homburg, Amer Farea, Galina Glazko, Markus Geuss, Andreas Holzinger, Oliver Ittig, Joni-Kristian Kämäräinen, Juho Kanninen, Urs-Martin Künzi, Aliyu Musa, Jin Tao, Yongtang Shi, and Olli Yli-Harja and apologize to all those who have not been named mistakenly. For proof-reading and help with various chapters, we would like to express our special thanks to Shailesh Tripathi, Kalifa Manjang, Tanvi Sharma, Nadeesha Perera, Zhen Yang, and many students from the course Computational Diagnostics of Data (DATA.ML.390). We would also like to thank our editors Mary James, Zoe Kennedy, Vinodhini Srinivasan, Sanjana Sundaram, and Brian Halm from Springer who have been always available and helpful.

Finally, we hope this book helps to spread out the enthusiasm and joy we have for this field, and inspires students and scientists for their studies or research questions.

Tampere, Finland
Belfast, UK
Brig, Switzerland
August 2022

Frank Emmert-Streib
Salissou Moutari
Matthias Dehmer

Contents

1	Introduction to Learning from Data	1
1.1	What Is Data Science?	2
1.2	Converting Data into Knowledge	4
1.2.1	Big Aims: Big Questions	4
1.2.2	Generating Insights by Visualization	7
1.3	Structure of the Book	8
1.3.1	Part I	9
1.3.2	Part II	9
1.3.3	Part III	10
1.4	Our Motivation for Writing This Book	10
1.5	How to Use This Book	12
1.6	Summary	14
 Part I General Topics		
2	General Prediction Models	17
2.1	Introduction	17
2.2	Categorization of Methods	17
2.2.1	Properties of the Data	18
2.2.2	Properties of the Optimization Algorithm	20
2.2.3	Properties of the Model	20
2.2.4	Summary	21
2.3	Overview of Prediction Models	21
2.4	Causal Model vs Predictive Model	22
2.5	Explainable AI	23
2.6	Fundamental Statistical Characteristics of Prediction Models	24
2.6.1	Example	26
2.7	Summary	27
2.8	Exercises	28

3	General Error Measures	29
3.1	Introduction	29
3.2	Motivation	30
3.3	Fundamental Error Measures	30
3.4	Error Measures	32
3.4.1	True-Positive Rate and True-Negative Rate	33
3.4.2	Positive Predictive Value and Negative Predictive Value	33
3.4.3	Accuracy	34
3.4.4	F-Score	34
3.4.5	False Discovery Rate and False Omission Rate	36
3.4.6	False-Negative Rate and False-Positive Rate	36
3.4.7	Matthews Correlation Coefficient	37
3.4.8	Cohen's Kappa	39
3.4.9	Normalized Mutual Information	40
3.4.10	Area Under the Receiver Operator Characteristic Curve	41
3.5	Evaluation of Outcome	44
3.5.1	Evaluation of an Individual Method	45
3.5.2	Comparing Multiple Binary Decision-Making Methods	47
3.6	Summary	50
3.7	Exercises	51
4	Resampling Methods	53
4.1	Introduction	53
4.2	Resampling Methods for Error Estimation	54
4.2.1	Holdout Set	54
4.2.2	Leave-One-Out CV	55
4.2.3	K-Fold Cross-Validation	56
4.3	Extended Resampling Methods for Error Estimation	58
4.3.1	Repeated Holdout Set	58
4.3.2	Repeated K-Fold CV	58
4.3.3	Stratified K-Fold CV	58
4.4	Bootstrap	59
4.4.1	Resampling With vs Resampling Without Replacement	60
4.5	Subsampling	61
4.6	Different Types of Prediction Data Sets	62
4.7	Sampling from a Distribution	63
4.8	Standard Error	66
4.9	Summary	69
4.10	Exercises	70

5	Data	71
5.1	Introduction	71
5.2	Data Types	72
5.2.1	Genomic Data	72
5.2.2	Network Data	74
5.2.3	Text Data	79
5.2.4	Time-to-Event Data	83
5.2.5	Business Data	85
5.3	Summary	87

Part II Core Methods

6	Statistical Inference	91
6.1	Exploratory Data Analysis and Descriptive Statistics	92
6.1.1	Data Structure	92
6.1.2	Data Preprocessing	93
6.1.3	Summary Statistics and Presentation of Information	93
6.1.4	Measures of Location	94
6.1.5	Measures of Scale	98
6.1.6	Measures of Shape	99
6.1.7	Data Transformation	101
6.1.8	Example: Summary of Data and EDA	102
6.2	Sample Estimators	104
6.2.1	Point Estimation	104
6.2.2	Unbiased Estimators	105
6.2.3	Biased Estimators	106
6.2.4	Sufficiency	107
6.3	Bayesian Inference	110
6.3.1	Conjugate Priors	112
6.3.2	Continuous Parameter Estimation	112
6.3.3	Discrete Parameter Estimation	116
6.3.4	Bayesian Credible Intervals	118
6.3.5	Prediction	121
6.3.6	Model Selection	122
6.4	Maximum Likelihood Estimation	123
6.4.1	Asymptotic Confidence Intervals for MLE	124
6.4.2	Bootstrap Confidence Intervals for MLE	127
6.4.3	Meaning of Confidence Intervals	128
6.5	Expectation-Maximization Algorithm	129
6.5.1	Example: EM Algorithm	131
6.6	Summary	134
6.7	Exercises	135
7	Clustering	137
7.1	Introduction	137
7.2	What Is Clustering?	138

7.3	Comparison of Data Points	138
7.3.1	Distance Measures	141
7.3.2	Similarity Measures	142
7.4	Basic Principle of Clustering Algorithms.....	143
7.5	Non-hierarchical Clustering Methods.....	145
7.5.1	<i>K</i> -Means Clustering	145
7.5.2	<i>K</i> -Medoids Clustering.....	147
7.5.3	Partitioning Around Medoids (PAM)	148
7.6	Hierarchical Clustering	149
7.6.1	Dendrograms	149
7.6.2	Two Types of Dissimilarity Measures	150
7.6.3	Linkage Functions for Agglomerative Clustering	151
7.6.4	Example	151
7.7	Defining Feature Vectors for General Objects	153
7.8	Cluster Validation	155
7.8.1	External Criteria	156
7.8.2	Assessing the Numerical Values of Indices.....	158
7.8.3	Internal Criteria	158
7.9	Summary	161
7.10	Exercises	161
8	Dimension Reduction	163
8.1	Introduction	163
8.2	Feature Extraction	164
8.2.1	An Overview of PCA	164
8.2.2	Geometrical Interpretation of PCA	165
8.2.3	PCA Procedure.....	165
8.2.4	Underlying Mathematical Problems in PCA	167
8.2.5	PCA Using Singular Value Decomposition	168
8.2.6	Assessing PCA Results	169
8.2.7	Illustration of PCA Using R	170
8.2.8	Kernel PCA	175
8.2.9	Discussion.....	177
8.2.10	Non-negative Matrix Factorization	179
8.3	Feature Selection.....	184
8.3.1	Filter Methods Using Mutual Information	186
8.4	Summary	188
8.5	Exercises	189
9	Classification	191
9.1	Introduction	191
9.2	What Is Classification?	191
9.3	Common Aspects of Classification Methods	192
9.3.1	Basic Idea of a Classifier	192
9.3.2	Training and Test Data.....	193
9.3.3	Error Measures.....	194

9.4	Naive Bayes Classifier	197
9.4.1	Educational Example	197
9.4.2	Example	200
9.5	Linear Discriminant Analysis	202
9.5.1	Extensions	206
9.6	Logistic Regression	207
9.7	k-Nearest Neighbor Classifier	211
9.8	Support Vector Machine	216
9.8.1	Linearly Separable Data	216
9.8.2	Nonlinearly Separable Data	218
9.8.3	Nonlinear Support Vector Machines	219
9.8.4	Examples	220
9.9	Decision Tree	222
9.9.1	What Is a Decision Tree	223
9.9.2	Step 1: Growing a Decision Tree.....	227
9.9.3	Step 2: Assessing the Size of a Decision Tree.....	230
9.9.4	Step 3: Pruning a Decision Tree.....	234
9.9.5	Predictions	235
9.10	Summary	235
9.11	Exercises	237
10	Hypothesis Testing.....	239
10.1	Introduction	239
10.2	What Is Hypothesis Testing?	240
10.3	Key Components of Hypothesis Testing	241
10.3.1	Step 1: Select Test Statistic	242
10.3.2	Step 2: Null Hypothesis H_0 and Alternative Hypothesis H_1	242
10.3.3	Step 3: Sampling Distribution	243
10.3.4	Step 4: Significance Level α	247
10.3.5	Step 5: Evaluate the Test Statistic from Data	248
10.3.6	Step 6: Determine the p -Value	248
10.3.7	Step 7: Make a Decision About the Null Hypothesis ..	249
10.4	Type 2 Error and Power	250
10.4.1	Connections Between Power and Errors.....	252
10.5	Confidence Intervals	253
10.5.1	Confidence Intervals for a Population Mean with Known Variance	254
10.5.2	Confidence Intervals for a Population Mean with Unknown Variance	255
10.5.3	Bootstrap Confidence Intervals	255
10.6	Important Hypothesis Tests.....	255
10.6.1	Student's t-Test.....	256
10.6.2	Correlation Tests	259
10.6.3	Hypergeometric Test.....	262
10.6.4	Finding the Correct Hypothesis Test.....	265

10.7	Permutation Tests	267
10.8	Understanding vs Applying Hypothesis Tests	269
10.9	Historical Notes and Misinterpretations	270
10.10	Summary	272
10.11	Exercises	272
11	Linear Regression Models	273
11.1	Introduction	273
11.1.1	What Is Linear Regression?	273
11.1.2	Motivating Example	274
11.2	Simple Linear Regression	276
11.2.1	Ordinary Least Squares Estimation of Coefficients	277
11.2.2	Variability of the Coefficients	280
11.2.3	Testing the Necessity of Coefficients	280
11.2.4	Assessing the Quality of a Fit	281
11.3	Preprocessing	282
11.4	Multiple Linear Regression.....	283
11.4.1	Testing the Necessity of Coefficients	284
11.4.2	Assessing the Quality of a Fit	285
11.5	Diagnosing Linear Models	285
11.5.1	Error Assumptions	285
11.5.2	Linearity Assumption of the Model.....	288
11.5.3	Leverage Points	289
11.5.4	Outliers	289
11.5.5	Collinearity	290
11.5.6	Discussion.....	291
11.6	Advanced Topics	292
11.6.1	Interactions	292
11.6.2	Nonlinearities	292
11.6.3	Categorical Predictors	294
11.6.4	Generalized Linear Models.....	297
11.7	Summary	306
11.8	Exercises	307
12	Model Selection	309
12.1	Introduction	309
12.2	Difference Between Model Selection and Model Assessment....	309
12.3	General Approach to Model Selection	310
12.4	Model Selection for Multiple Linear Regression Models	312
12.4.1	R^2 and Adjusted R^2	312
12.4.2	Mallow's Cp Statistic	313
12.4.3	Akaike's Information Criterion (AIC) and Schwarz's BIC	314
12.4.4	Best Subset Selection	316
12.4.5	Stepwise Selection	316

12.5	Model Selection for Generalized Linear Models	318
12.5.1	Negative Binomial Regression Model	318
12.5.2	Zero-Inflated Poisson Model	320
12.5.3	Quasi-Poisson Model	321
12.5.4	Comparison of GLMs	322
12.6	Model Selection for Bayesian Models	324
12.7	Nonparametric Model Selection for General Models with Resampling	326
12.8	Summary	328
12.9	Exercises	330

Part III Advanced Topics

13	Regularization	333
13.1	Introduction	333
13.2	Preliminaries	334
13.2.1	Preprocessing and Norms	334
13.2.2	Data	335
13.2.3	R Packages for Regularization	336
13.3	Ridge Regression	336
13.3.1	Example	338
13.4	Non-negative Garrote Regression	339
13.5	LASSO	339
13.5.1	Example	341
13.5.2	Explanation of Variable Selection	341
13.5.3	Discussion	344
13.5.4	Limitations	344
13.6	Ridge Regression	345
13.7	Dantzig Selector	345
13.8	Adaptive LASSO	346
13.8.1	Example	347
13.9	Elastic Net	348
13.9.1	Example	349
13.9.2	Discussion	350
13.10	Group LASSO	352
13.10.1	Example	353
13.10.2	Remarks	354
13.11	Discussion	355
13.12	Summary	356
13.13	Exercises	357
14	Deep Learning	359
14.1	Introduction	359
14.2	Architectures of Classical Neural Networks	360
14.2.1	Mathematical Model of an Artificial Neuron	360
14.2.2	Feedforward Neural Networks	362

14.2.3	Recurrent Neural Networks	363
14.2.4	Overview of General Network Architectures	364
14.3	Deep Feedforward Neural Networks	365
14.3.1	Example: Deep Feedforward Neural Networks	366
14.4	Convolutional Neural Networks	372
14.4.1	Basic Components of CNN	375
14.4.2	Important Variants of CNN	379
14.4.3	Example: CNN	382
14.5	Deep Belief Networks	382
14.5.1	Pre-training Phase: Unsupervised	384
14.5.2	Fine-Tuning Phase: Supervised	387
14.6	Autoencoder	390
14.6.1	Example: Denoising and Variational Autoencoder	392
14.7	Long Short-Term Memory Networks	398
14.7.1	LSTM Network Structure with Forget Gate	400
14.7.2	Peephole LSTM	402
14.7.3	Applications	404
14.7.4	Example: LSTM	404
14.8	Discussion	410
14.8.1	General Characteristics of Deep Learning	410
14.8.2	Explainable AI	410
14.8.3	Big Data vs Small Data	415
14.8.4	Advanced Models	416
14.9	Summary	416
14.10	Exercises	417
15	Multiple Testing Corrections	419
15.1	Introduction	419
15.2	Preliminaries	420
15.2.1	Formal Setting	420
15.2.2	Simulations Using R	423
15.2.3	Focus on Pairwise Correlations	423
15.2.4	Focus on a Network Correlation Structure	424
15.2.5	Application of Multiple Testing Procedures	424
15.3	Motivation of the Problem	425
15.3.1	Theoretical Considerations	426
15.3.2	Experimental Example	428
15.4	Types of Multiple Testing Procedures	428
15.4.1	Single-Step vs Stepwise Approaches	428
15.4.2	Adaptive vs Nonadaptive Approaches	431
15.4.3	Marginal vs Joint Multiple Testing Procedures	431
15.5	Controlling the FWER	431
15.5.1	Šidák Correction	431
15.5.2	Bonferroni Correction	432
15.5.3	Holm Correction	434

15.5.4	Hochberg Correction	435
15.5.5	Hommel Correction	436
15.5.6	Westfall-Young Procedure	439
15.6	Controlling the FDR	442
15.6.1	Benjamini-Hochberg Procedure	442
15.6.2	Adaptive Benjamini-Hochberg Procedure	443
15.6.3	Benjamini-Yekutieli Procedure	445
15.6.4	Benjamini-Krieger-Yekutieli Procedure	446
15.6.5	Blanchard-Roquain Procedure	447
15.7	Computational Complexity	448
15.8	Comparison	449
15.9	Summary	451
15.10	Exercises	452
16	Survival Analysis	453
16.1	Introduction	453
16.2	Motivation	454
16.2.1	Effect of Chemotherapy: Breast Cancer Patients	454
16.2.2	Effect of Medication: Agitation	455
16.3	Censoring	455
16.4	General Characteristics of a Survival Function	457
16.5	Nonparametric Estimator for the Survival Function	458
16.5.1	Kaplan–Meier Estimator for the Survival Function	458
16.5.2	Nelson-Aalen Estimator for the Survival Function	459
16.6	Comparison of Two Survival Curves	459
16.6.1	Log-Rank Test	460
16.7	Hazard Function	461
16.7.1	Weibull Model	463
16.7.2	Exponential Model	464
16.7.3	Log-Logistic Model	464
16.7.4	Log-Normal Model	465
16.7.5	Interpretation of Hazard Functions	466
16.8	Cox Proportional Hazard Model	467
16.8.1	Why Is the Model Called a Proportional Hazard Model?	469
16.8.2	Interpretation of General Hazard Ratios	470
16.8.3	Adjusted Survival Curves	471
16.8.4	Testing the Proportional Hazard Assumption	471
16.8.5	Parameter Estimation of the CPHM via Maximum Likelihood	474
16.9	Stratified Cox Model	477
16.9.1	Testing No-Interaction Assumption	477
16.9.2	Case of Many Covariates Violating the PH Assumption	479

16.10	Survival Analysis Using R	479
16.10.1	Comparison of Survival Curves	479
16.10.2	Analyzing a Cox Proportional Hazard Model	481
16.10.3	Testing the PH Assumption	482
16.10.4	Hazard Ratios	483
16.11	Further Reading	484
16.12	Summary	484
16.13	Exercises	485
17	Foundations of Learning from Data	487
17.1	Introduction	487
17.2	Computational and Statistical Learning Theory	488
17.2.1	Probabilistic Learnability	488
17.2.2	Probably Approximately Correct (PAC) Learning	490
17.2.3	Vapnik Chervonenkis (VC) Theory	498
17.3	Importance of Bias for Learning	501
17.4	Learning as Optimization Problem	502
17.4.1	Empirical Risk Minimization	502
17.4.2	Structural Risk Minimization	503
17.5	Fundamental Theorem of Statistical Learning	504
17.6	Discussion	505
17.7	Modern Machine Learning Paradigms	505
17.7.1	Semi-supervised Learning	506
17.7.2	One-Class Classification	508
17.7.3	Positive-Unlabeled Learning	509
17.7.4	Few/One-Shot Learning	510
17.7.5	Transfer Learning	512
17.7.6	Multi-Task Learning	515
17.7.7	Multi-Label Learning	516
17.8	Summary	517
17.9	Exercises	518
18	Generalization Error and Model Assessment	519
18.1	Introduction	519
18.2	Overall View on Model Diagnosis	520
18.3	Expected Generalization Error	521
18.4	Bias-Variance Trade-Off	523
18.5	Error-Complexity Curves	528
18.5.1	Example: Linear Polynomial Regression Model	529
18.5.2	Example: Error-Complexity Curves	531
18.5.3	Interpretation of Error-Complexity Curves	533
18.6	Learning Curves	535
18.6.1	Example: Learning Curves for Linear Polynomial Regression Models	535
18.6.2	Interpretation of Learning Curves	537

Contents	xvii
18.7 Discussion	538
18.8 Summary	540
18.9 Exercises	541
References	543
Index	565

AUTHOR QUERY

AQ1. Please check if the updated author names in the sentence “In particular, would...” is fine

Chapter 1

Introduction to Learning from Data



We are living in a data-rich era, where every field of science or industry sector generates data in a seemingly effortless manner [156, 389]. In order to emphasize the importance of this, data have been called the “oil of the twenty-first century” [228]. To deal with this flood of data, a new field has been established called *data science* [80, 146, 224]. Data science combines the skill sets and expert knowledge from many different fields, including machine learning, artificial intelligence, statistics, and pattern recognition [146, 216, 478].

The availability of data provides new opportunities in all fields and industries for gaining new information and for tackling difficult problems. However, data alone do not provide information; instead, they need to be analyzed to unlock the answers to questions buried within them. This is what we call *learning from data*. To motivate the importance of learning from data, let’s look at three examples from genomics, finance, and Web applications.

Traditionally, biology has been a field one would not associate with technology. However, in the last 30 years, significant advances in experimental techniques have been made, allowing nowadays an easy and affordable generation of different types of data from measurements of various aspects of biological cells. The most prominent example is probably the sequencing of the DNA. The collection of these different data types is summarized under the term omics or sometimes genomics data. Importantly, genomics technology is not only used in research but also in hospitals to generate patients’ data. This enables such data to be used for diagnostic, prognostic, and therapeutic applications with a direct benefit for patients. A second source of big data is from finance. Nowadays, there is a myriad of different financial markets, including stock exchanges, providing temporal information about the market value of companies even on a sub-minute scale. This information can be used by investors to select an optimal portfolio that is resilient to economic crisis. As a third example of mass data, we would like to mention Web applications, e.g., for online shopping or social networking. Such Web applications utilize the Internet,

which became available to the public in the 1990s, for placing orders or exchanging information about all aspects of our private and professional life.

In this book, we discuss the basic tools enabling data scientists to learn from data. As we will see, it takes a journey to gain an understanding of all the different methods and approaches and become proficient.

1.1 What Is Data Science?

From time to time, new scientific fields emerge to adapt to a changing world. Examples of newly established academic disciplines include economy (the first professorship in economics was established at the University of Cambridge in 1890 and was held by Alfred Marshall [325]), computer science (the first department of computer science in the United States was established at Purdue University in 1962, whereas the term “computer science” appeared first in [160]), bioinformatics (the term was first used in [244] in 1978), and most recently data science [95, 316, 389].

The first appearance of the term “data science” is ascribed to Peter Naur in 1974 [353], but it took nearly 30 years until there were callings for an independent discipline with this name [80]. Since then, the first Research Center for Dataology and Data Science was established at Fudan University in Shanghai, China, in 2007, and Harvard Business Review even called data scientist “The Sexiest Job of the twenty-first century” [376].

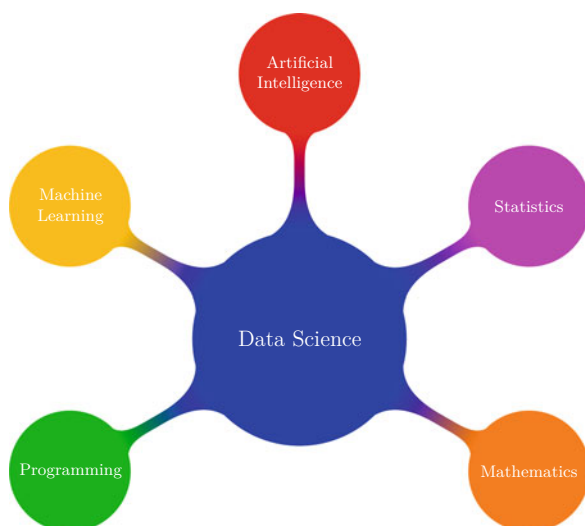
A natural question to ask is, “What is data science?” In [142], a data-driven scientometrics analysis of this question has been presented by studying data of publication statistics from scientists provided by Google Scholar. As a result, the top 20 most influential fields were all found to contribute significantly to data science. Those fields include machine learning, artificial intelligence, data mining, and statistics.

An important consequence resulting from the above analysis is that data science is not a monolithic field. Instead, it consists of many different approaches and concepts having their origin in entirely different fields and communities, e.g., machine learning, artificial intelligence, or statistics. From the perspective of a learner, this is unfortunate because the learning process will not be monolithic but undulating. However, due to its inclusive nature, i.e., covering methods irrespective of the field of origin, in our opinion, data science provides the most comprehensive toolbox for analyzing data.

Regardless of its inter- and multidisciplinary nature, data science consists of five major components (see Fig. 1.1):

- : machine learning
- : artificial intelligence
- : statistics
- : mathematics
- : programming

Fig. 1.1 Data science is composed of five major components, and each one makes a unique contribution



The first three components, ■ machine learning, ■ artificial intelligence, and ■ statistics, provide all the methods used in data science to analyze data. Representative methods therefore are support vector machines (SVMs), neural networks (NNs), and generalized linear models (GLMs). Each of these methods is based on ■ mathematics, providing the fundamental methodology for formulating such methods. Finally, ■ programming connects everything with each other. It is important to realize that programming is a glue skill that (1) enables the practical application of methods from machine learning, artificial intelligence, and statistics, (2) allows the combination of different methods from different fields, and (3) provides practical means for developing novel computer-based methods (using, e.g., resampling methods or Monte Carlo simulations). For clarity reasons, we want to emphasize that here when we speak about “programming skills,” we mean “scientific” and “statistical” programming rather than general purpose programming skills. All of these points are of major importance for data science. In some sense, mathematics and programming form the root of data science (see Fig. 1.1), whereas machine learning, artificial intelligence, and statistics provide the methodological realizations, establishing the roofing.

It is this multicomponent nature of data science which explains why the field is usually taught at a graduate level, whereas mathematics and programming are taught at the undergraduate level. The reason therefore is that learning basics about mathematics and programming requires already a considerable amount of time if one wants to acquire a higher level of proficiency. Furthermore, it is clear that one book alone can neither cover all relevant topics of these five components nor introduce them to a sufficient detail as needed for the beginner. For this reason,

we refer to our introductory textbook for learning basics in mathematics and programming [149].

Regarding useful programming languages, R and Python are very popular nowadays. However, while both provide similar capabilities, there are differences in certain situations. In this book, we prefer R over Python due to its statistical origin. In fact, R was developed to provide a “statistical programming language.” We will see the benefits of this when discussing the topics on hypothesis testing (Chap. 10), resampling methods (Chap. 4), or linear regression (Chap. 11), where R provides excellent functionalities.

Although this book does not provide an introduction to programming and mathematics (this is provided in [149]), we present examples in R for the methods from  machine learning,  artificial intelligence, and  statistics. A pedagogical side effect of this presentation is to enhance the computational thinking. This has a profound influence on the analytical problem-solving capabilities because it enables to think in guided ways, which can be computationally realized in a practical manner, rather than coming up with ungrounded solution proposals that are intractable.

1.2 Converting Data into Knowledge

In data science, our main aim is to *learn about a phenomenon* that underlies available data. In general terms, this means that we want to extract *reliable information* from the data through the application of *appropriate analysis methods*. In the following, we discuss this in more detail.

1.2.1 Big Aims: Big Questions

In order to learn about a given problem, we need to be able to answer important relevant questions by interrogating related data in a way that allows us to enhance our current knowledge about the problem. Ideally, we would like to ask “big questions” in a sense that their answers will put us into a position to solve the problem entirely. The following questions provide some examples for such big questions:

- What is a cure for breast cancer?
- What will be the stock price of Alphabet (parent company of Google) on the 27th of May 2057?
- What products will a customer order from Amazon the next time?
- What is the meaning of life?

Without long explanations, it is easy to see for everyone that an answer to any of the above questions would have a huge impact—on different levels. In the first case, you

would certainly be awarded the Nobel Prize in Medicine and Physiology, whereas in the second and third case, you could become rich. Finally, in the fourth case, you might not earn scientific merits but would probably make a lot of people happy because the search would have an end.

From the above examples, there are two important lessons to be learned. First, there are usually no analysis methods available that provide a direct answer to any of the above questions. Even worse, the results which can be obtained from such analysis methods are usually not even close to answers for “big questions.” The reason for this is that data science methods work differently, as we will see below. Second, questions of the fourth type are out of the scope of this book as we are only dealing with questions that can be addressed by the *analysis of data*.

The second lesson seems trivial; however, there are related cases that are not so easy to spot in practice because they come disguised. For instance, a couple of years ago, we were analyzing proteomics data from SELDI-TOF (surface-enhanced laser desorption/ionization time-of-flight) experiments (providing information about proteins), and to our surprise, we were not able to detect anything by any analysis method. Curiously, also, independent analysis attempts by several other teams confirmed our negative findings. Later, we found out that the experiments conducted were corrupted, hence, confirming that the data did not contain any meaningful information.

The first lesson above is also not trivial either, but its underlying argument is different. In order to understand this, we show in Fig. 1.2 an overview of six principal categories of analysis methods. For each of these categories, we provide information about the data type such an analysis is based on, the question the method addresses, and some example of methods, which are discussed in later chapters of this book. In order to simplify the discussion, we present only simplified versions for, e.g., the data types that can be handled, without effecting the following discussion. It is important to note that for each method-category, the question addressed is of a *simple nature* compared to any of the 3 big questions above. The reason for this is not that we intentionally selected only method-categories that address such simple questions, but it is a characteristic of all data analysis methods. Furthermore, it is important to emphasize that the questions that can be addressed by the six principal method-categories are representative for the categories as a whole and not just for particular methods of a category. However, this means that any “big question” needs somehow to be related back to such “simple questions” for which analysis methods are available. A further consequence of this is that in order to study “big questions,” one needs not only one but several different methods applied sequentially. Hence, studying “big questions” requires a *data analysis process* where a multitude of different methods are applied in a synchronized way [146].

In Fig. 1.3, we visualize the steps needed to reformulate a “big question” in order to arrive to a question that can be addressed by means of data science. The reformulation of the question requires expert knowledge and statistical thinking because essential elements of the problem need to be conserved, whereas unimportant ones can be neglected. For the reformulated question, there may be methods applicable. This requires computational thinking to select and implement the appropriate

Preprocessing	Preprocessing	Preprocessing
Clustering	Hypothesis Testing	Linear Regression
Data type: $X = \{(x_i)\}_{i=1}^p$ with feature vector $x_i \in \mathbb{R}^n$ Question addressed: Is there a 'structure' between variables? Example methods: k-means Hierarchical clustering	Data type: $X = \{x_i\}_{i=1}^n$ with $x_i \in$ Question addressed: Is there a difference for a test statistic? Example methods: t-test Fisher's exact test	Data type: $X = \{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^p$ and $y_i \in$ Question addressed: Is there a dependency between variables? Example methods: Simple Regression Multiple Regression
Preprocessing	Preprocessing	Preprocessing
Deep learning	Classification	Survival Analysis
Data type: X multiple data types are possible Question addressed: Multiple Example applications: Parameter estimation Clustering Classification Regression Time Series Analysis	Data type: $X = \{(x_i, y_i)\}_{i=1}^n$ with feature vector $x_i \in \mathbb{R}^p$ and class label $y_i \in \{-1, 1\}$ Question addressed: In what class should a data point be placed? Example methods: Logistic Regression SVM	Data type: $X = \{(t_i, c_i)\}_{i=1}^n$ with survival time $t_i \in \mathbb{R}_+$ and censoring $c_i \in \{0, 1\}$ Question addressed: How is the survival be effected? Example methods: Kaplan-Meier curves Cox Proportional-Hazards Model

Fig. 1.2 Six principal method-categories discussed in this book allowing to address a large variety of different application-specific questions

one. Potentially, this method needs to be adapted to fit the reformulated question optimally. This requires mathematical thinking to redesign the algorithm. From the results of this analysis, we can then draw conclusions back to the original “big question.” In general, in science, the steps along the orange triangle are iteratively repeated, establishing what is called *scientific discovery*.

In summary, in order to analyze data, one needs to learn how to (re)formulate questions so that we can analyze the data in a problem-oriented manner. The key for this step is a thorough understanding of principal analysis categories and their methods. This requires especially statistical thinking for the reformulation of a question itself, the analysis of the data, and the conclusions that can be drawn thereof.

Another point of caution to mention is that *simple* does not mean *trivial*. This is visualized in Fig. 1.4, where we show the three main components for analyzing

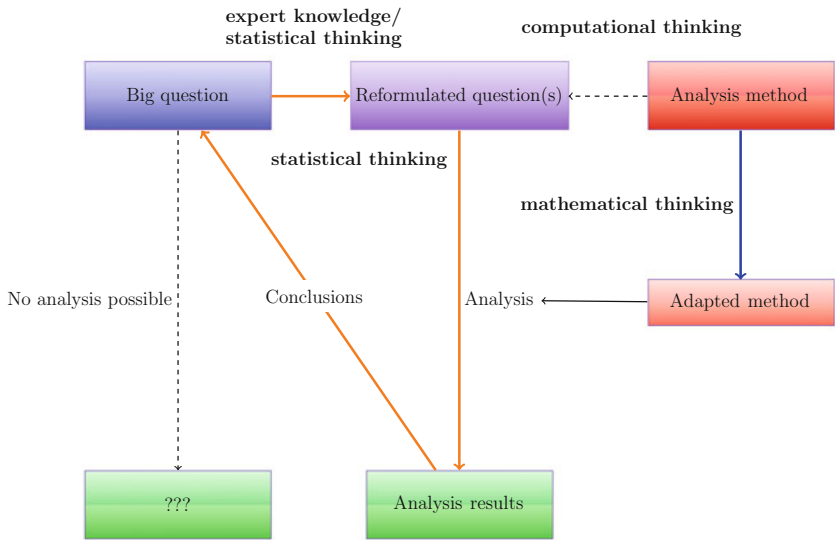


Fig. 1.3 In order to study a “big question” amenable to an analysis method from data science, one needs to reformulate it. This requires expert knowledge, statistical thinking, computational thinking, and mathematical thinking

data (data, methods, and results) and the corresponding topics one needs to address in order to specify a data analysis project. This involves preprocessing, modeling, analysis, and design. The numbers in bracket indicate the chapters, in this book, discussing the respective subjects. For the beginner, it may be interesting to see the iteration arc from results to data. This means even a simple data analysis does not consist of running the analysis once but many times, e.g., for estimating learning curves (discussed in Chap. 18). Overall, to specify a data analysis, one needs to address each aspect of the categories shown in Fig. 1.4.

From an educational perspective, the above described complexity of data science projects poses challenges, because one cannot address all issues at the same time. Instead, it is easier to learn the *elements of data science* step-by-step to gain a thorough understanding of its components. In this book, we will also follow this approach.

1.2.2 Generating Insights by Visualization

It is important to note that in addition to the six quantitative method-categories listed in Fig. 1.2, there is one further principal method-category that is of qualitative nature. This category comprises *visual exploration methods*. The conceptual idea behind this was introduced in the 1950s by John Tukey, who advocated widely the

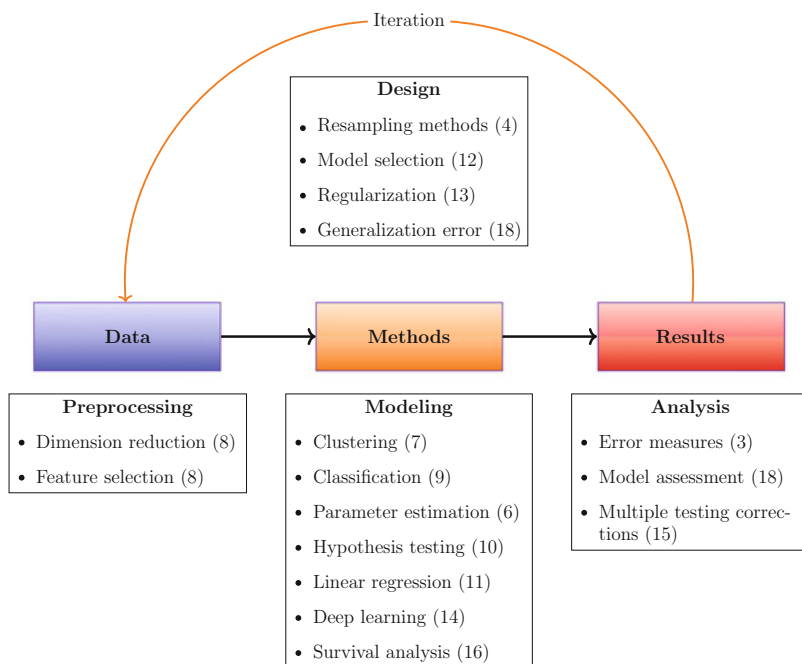


Fig. 1.4 The three main components for analyzing data (data, methods, and results) and corresponding topics one needs to address (non-exhaustive list) in order to specify a data analysis project

idea of data visualization as a mean to generate novel hypothesis about data. In the statistics community, such an approach is called *exploratory data analysis (EDA)*. In general, EDA uses data visualization techniques, e.g., box plots, scatter plots, etc. as well as summary statistics like the mean or the variance, to get either an overview of the characteristics of the data or to generate a new understanding. For instance, a first step in formulating a question, which can be addressed by a quantitative analysis method (see Fig. 1.3), would consist of visualizing the data.

Overall, this means that there are seven principal method-categories of data science. In Chap. 6, we discuss a variety of different visualization methods, which can be utilized for the purpose of an exploratory data analysis.

1.3 Structure of the Book

In the following, we discuss the structure and the content of this book briefly. Overall, the book is structured into three main parts.

1.3.1 *Part I*

In Part I, we start with a discussion of general prediction models and their categorizations as well as the difference between inferential models and predictive models. In Chap. 3, we discuss general error measures one can use for supervised learning problems, which are later discussed in Part II of the book. In Chap. 4, we introduce resampling methods, e.g., cross validation, and show how they are used for error estimation. Furthermore, we discussed related topic, e.g., subsampling and sampling from a distribution. This chapter shows that error estimates of learning models are random variables, requiring estimates of their variability, e.g., in the form of the standard error. The last chapter in Part I is about different data types frequently encountered in data science. This chapter is important because methods do not operate in isolation but are always applied to data. Hence, data are of course an important part of data science, and their sufficient understanding is required. In this chapter, we provide five examples of different data types (genomic data, network data, text data, time-to-event data, and business data) to show that data structures can be complex.

1.3.2 *Part II*

In Part II, we present core analysis methods of data science. Specifically, we discuss statistical inference, clustering, dimension reduction, classification, hypothesis testing, linear regression, and model selection. Each of these chapters presents the methods side by side with examples using the statistical programming language R.

Part II starts with a chapter on statistical inference. First, we present an overview of descriptive statistics and *exploratory data analysis* (EDA). As briefly mentioned above, EDA advocates the idea to visualize data as a mean to generate insights about the underlying problem [237, 470]. That means the visualization of data is very important because it represents actually a form of data analysis. Further topics discussed in this chapter are Bayesian inference, maximum likelihood estimation, and the expectation-maximization (EM) algorithm. In Chap. 7, we discuss clustering methods, which can be used when only unlabeled data are available. Also, Chap. 8 presents unsupervised learning methods but for use in dimension reduction. In contrast, in Chap. 9, we discuss supervised learning methods for classification. There, we present a variety of different approaches for classification including naive Bayes classifier, logistic regression, and decision tree. In Chap. 10, we introduce hypothesis testing and a number of important hypothesis tests that find frequent application in practice, e.g., Student's t-test or Fisher's exact test. Chapter 11 presents again supervised learning methods; however, unlike in Chap. 9, the focus is on a gradual output instead of a categorical one. Data of such a type can be studied using regression methods. Finally, Part II ends with a chapter about model selection.

The order of the chapters in Part II follows largely a logical order when conducting a data science project and considers also the dependency between the methods. For instance, for linear regression models, one can use a hypothesis test to check whether a regression coefficient vanishes. Hence, hypothesis testing needs to be presented first in order to understand this. Another example is the topic of model selection, which aims to choose the best model among a family of prediction models, e.g., regression models. Hence, regression is presented before model selection.

1.3.3 *Part III*

In Part III, we discuss advanced topics of data science. Specifically, we introduce regularization, deep learning, multiple testing corrections, and survival analysis models. Furthermore, we discuss theoretical foundations of learning from data and the generalization error. All of these topics and methods provide powerful approaches for conducting complex data science projects in a sound way. It is important to emphasize that many of those methods are not independent but provide extensions of methods discussed in Part II. For instance, regularization (Chap. 13) builds upon linear regression models (Chap. 11), and multiple testing corrections (Chap. 15) extend statistical hypothesis testing (Chap. 10). Also survival analysis (Chap. 16) is not a stand-alone subject but contains elements of linear regression models (Chap. 11) and statistical hypothesis testing (Chap. 10).

The dependency among the chapters reflects also a general characteristic of data science. This means when selecting a particular method for an analysis, this requires usually a multitude of “other” considerations beyond the analysis method itself. Hence, data science projects are typically complex, resisting a cookbook-style presentation.

Finally, we would like to remark that each chapter ends with a brief summary. The summary contains also a learning outcome box, highlighting the most important lesson from a chapter. In general, a chapter will contain many learning outcomes; however, by highlighting one, we want to encourage the reader to reflect about its content from a bird’s-eye view. This is important because in order to understand any topic, one needs to be able to switch between the technical aspects offered by a method to the general perspective it provides. This mental switching allows to avoid getting lost in either the nitty-gritty or a generic outlook.

1.4 Our Motivation for Writing This Book

Conducting research projects in data science for science or industry typically requires the application of *appropriate* methods. This involves model selection, model assessment, and problem visualization. For each subtask, decisions need

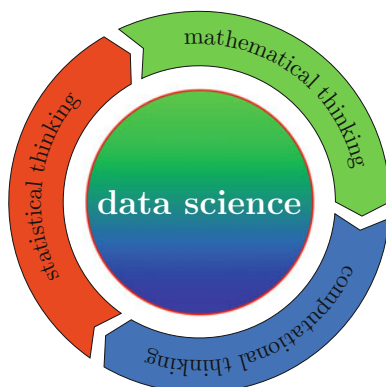
to be made, e.g., “what to study” and “how to study.” Taken together, this makes data science projects complex, requiring a multitude of different skills and knowledge from a variety of different areas. In order to succeed, one needs a detailed understanding of all those methods and approaches, which is far beyond a cookbook thinking. The goal of this book is to provide a data science toolbox with the most important methods from machine learning, artificial intelligence, and statistics to analyze data. We provide detailed information about such methods on an abstraction level that balances a mathematical understanding with an application-oriented desire. Overall, we do not only aim to provide information about analysis methods but do advocate statistical thinking. The latter allows the learner to advance from individual methods to complex data science projects.

From our experience in supervising data science projects, we learned over the years that there are three key components for learning how to analyze data: (1) a thorough understanding of methods and data, (2) an easy way to apply a method to data, and (3) iterative modification of (1) and (2) to study a problem. For (1), sufficient explanations of the methods and the data are needed to have a good knowledge of how a method works and what the data mean. For (2), computational implementations of methods are needed because essentially all data science projects require a computer. In this book, we use the statistical programming language R because it has a long history in the statistics community and it is increasingly dominating machine learning and artificial intelligence. For (3), only your creativity is the limit, which makes data science an art. More formally, statistical thinking is needed to connect all parts with each other.

From the above discussion, it follows that for beginners, books offering a comprehensive coverage may be not very beneficial because such presentations usually lack a thorough discussion of methods, neglect computational discussions of their applications, and do not guide the reader toward statistical thinking. Instead, such books are great reference books for experts. Also, books that focus only on methods from particular fields, e.g., from statistics or machine learning, have a limited utility in learning data science, because for an optimal analysis, the most appropriate methods need to be used, regardless of which field introduced them.

Our book is intended for the beginner trying to learn the basics of data science and to acquire an understanding. Hence, we emphasize the interplay between mathematical thinking, computational thinking, and statistical thinking, as shown in Fig. 1.5, throughout the book when discussing the different topics. We favor basics over a comprehensive presentation because once the basics are learned and understood, all advanced methods can be self-learned. Finally, we present important methods from essentially any field, including machine learning, artificial intelligence, and statistics, because the selection of methods needs to be made eclectically to obtain the best results.

Fig. 1.5 Data science requires proficiency in mathematical thinking, computational thinking, and statistical thinking. Only by integrating those skills and knowledge can one obtain the optimal results when working on a data science project



1.5 How to Use This Book

Our textbook can be used in many different areas related to data science including computer science, information technology, and statistics. The following list gives some suggestions of different courses at the (advanced) undergraduate and graduate level for which selected chapters of our book could be used:

- Artificial intelligence
- Big data
- Bioinformatics
- Business analytics
- Computational biology
- Computational finance
- Computational social science
- Data analytics
- Data mining
- Data science
- Deep learning
- Machine learning
- Natural language processing
- Statistical data analytics

The target audience of the book is graduate students in computer science, information technology, and statistics, but it can also be used for advanced undergraduate courses in related fields. For students lacking a thorough mathematical understanding of basic topics, including probability, analysis, or linear algebra, we recommend “Mathematical Foundations of Data Science Using R” [149] as an introductory textbook. This textbook provides also an introduction to the programming language R on the level required for the following chapters.

The order of the chapters in this book follows a natural progression of the difficulty level of the topics (see Fig. 1.6). Hence, the beginner can follow the

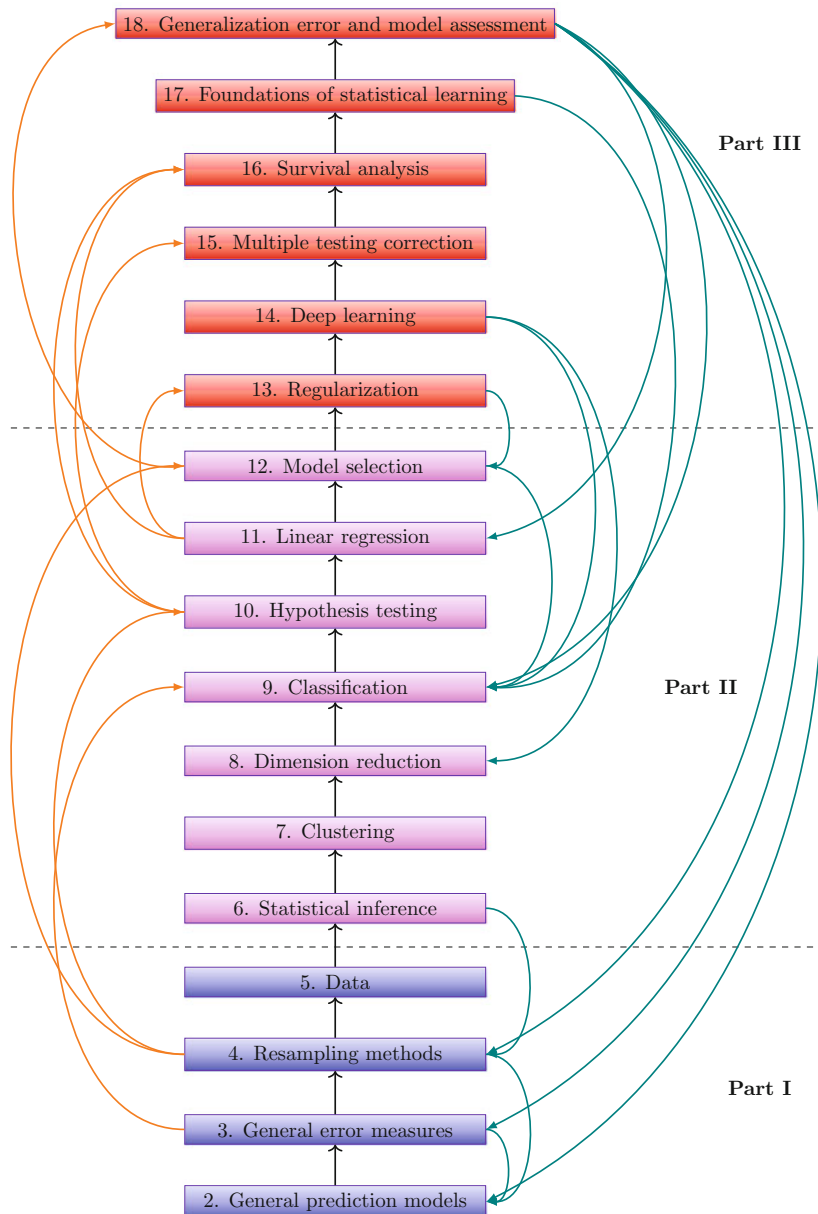


Fig. 1.6 Interconnectedness of the chapters in this book. The chapters in Part I, II, and III are shown in blue, purple, and red, respectively, while the shown links correspond to major dependencies among the topics. This indicates that data science forms a complex network of methods, concepts, and algorithms

chapters in the presented order to gain a progressive understanding about data science. However, we would like to note that there is not one size that fits all, and the same is true for learning data science. For instance, the topics of Chap. 18 about the generalization error can be seen either as a conceptual roof for all the previous chapters or as a conceptual root. The difference is the order of the presentation. We decided to present this topic at the end of the book because in our experience, many students ask, during the learning process, “what is it good for?” This question is naturally answered when presenting first practical applications of the generalization error. However, other students may prefer, first, to obtain theoretical insights before knowing how to realize them practically. In this case, the last chapter should be read at an earlier stage. Due to the fact that all chapters have their own major focus, the intermediate and advanced reader can choose an individual reading order for a personalized learning experience.

1.6 Summary

During the course of this book, we will see that the topics defining data science are very interconnected with each other. In Fig. 1.6, we show a visualization of this interconnectedness for the chapters in this book. The shown links correspond only to the major dependencies among the topics; however, many more connections exist. As one can see, there are forward connections (shown in orange) and backward connections (shown in green) among the topics. Overall, this highlights that data science forms a complex network of interconnected topics.

Learning Outcome 1: Data Science

Data science forms a complex network of methods, concepts, and algorithms, which defies a linear learning experience.

In order to achieve the best learning outcome, we advise the reader to work through the book multiple times and in different orders; for an example of these orders, see Fig. 1.6. Metaphorically, one can see data science as a language rather than a single method that needs to be wrapped creatively around a data-based problem in order to *communicate* efficiently with the information buried within the data.

Part I

General Topics

Chapter 2

General Prediction Models



2.1 Introduction

This chapter provides a general overview of prediction models. We present three different categorizations of methods commonly used to organize the various methods in data science. This will show that there is more than one way to look at prediction models, but none is superior to the others. In addition, we present our own pragmatic organization of methods, we use for the following chapters, which is formed by a mixture of application-based and model-based categories. Finally, we discuss a fundamental statistical characteristic which holds for every prediction model. We will see that every output of a prediction model is a random variable. In later chapters, we will utilize this in various ways, e.g., when discussing resampling methods (see Chap. 4) or the expected generalization error (see Chap. 18).

2.2 Categorization of Methods

Data science is an interdisciplinary field. That means its methods have not been developed by one research community but many. In the previous chapter, we mentioned already that its major contributions come from machine learning, artificial intelligence, statistics, and pattern recognition. A consequence of this is that there is no common criterion that can be used to categorize methods, e.g., which has been used to derive learning algorithms. Instead, there are considerable differences among the methods from data science, and there is no universal concept from which all methods could be derived. For this reason, depending on the perspective, various categorizations of methods are possible.

In Fig. 2.1, we show an overview of three such categorizations. The three columns in this figure emphasize different aspects of methods and their properties. Specifically, the first column emphasizes properties of the data, the second column

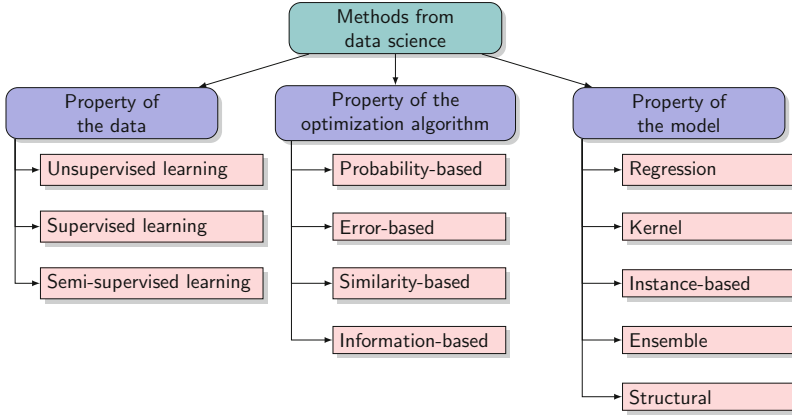


Fig. 2.1 Categorization of methods from data science. The three columns emphasize different aspects of the methods and their properties

properties of the optimization algorithm, and the third column properties of the model itself. Neither of these perspectives is superior to the others, but each is valid in their own right. In the following, we discuss each of the three main categories briefly.

2.2.1 *Properties of the Data*

This category emphasizes a data perspective. A data perspective is particularly practical because it allows the selection of a method based on the properties of the available data. For instance, if data for the variables X (input) are available without response variable(s) Y (output), methods for unsupervised learning need to be used. In contrast, if data for the variables X are available in combination with real valued response variable(s) Y , then supervised methods, e.g., for a regression model, can be used.

In general, one can distinguish between three major types of data, as defined in Fig. 2.2. Due to the fact that each data type contains a different type of information, different learning paradigms have been developed to deal with such data. The corresponding learning paradigms are called unsupervised learning, supervised learning, and semi-supervised learning.

For *supervised learning*, we distinguish between two different data characteristics (A and B); see Fig. 2.2. This shall indicate that there are further subcategories of the learning paradigm. For Data A, one uses a regression model to estimate the effect of explanatory variables (X , also called predictors) on the dependent variable(s) (Y). Another example of supervised learning is reinforcement learning [452].

Unsupervised learning	Supervised learning	Semi-supervised learning
Data: $X = \{(x_i)\}_1^n$ with feature vectors $x_i \in \mathbb{R}^m$ Question addressed: Is there a 'structure' between the variables? Example methods: Non-hierarchical clustering: K-means K-medoids Hierarchical clustering Demdograms Partitioning around medoids (PAM)	A: Data: $D = \{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^m$ and $y_i \in \mathbb{R}$ Question addressed: What is the effect of explanatory variables on the dependent variable? Example methods: Simple linear regression Multiple linear regression B: Data: $X = \{(x_i, y_i)\}_{i=1}^n$ with feature vectors $x_i \in \mathbb{R}^m$ and class labels $y_i \in \{-1, 1\}$ Question addressed: In what class should a data point be placed? Example methods: Logistic Regression SVM	Data: $D = \left\{ \{(x_i, y_i)\}_{i=1}^l, \{(x_j)\}_{j=1}^u \right\}$ with feature vector $x_i \in \mathbb{R}^m$ and class label $y_i \in \{-1, 1\}$ Questions addressed: Transductive learning (goal is to learn missing labels) or inductive learning (goal is to learn mapping $X \rightarrow Y$). Example methods: Transductive learning: Graph-based methods Inductive learning: Semi-supervised SVM

Fig. 2.2 Properties of the data. Shown are the three major learning paradigms, namely, unsupervised learning, supervised learning, and semi-supervised learning, as well as the characteristics of the data types they are based on. The questions addressed are for illustration purpose only, and other questions are possible

We want to note that methods from classical statistics, e.g., for parameter estimation or statistical hypothesis testing, are based on the assumption of the form of data used for *unsupervised learning*. Despite this fact, such methods are usually not designated as unsupervised learning methods. However, this is merely due to the jargon used by different research communities.

The third data characteristic we distinguish in Fig. 2.2 is called *semi-supervised learning*. Semi-supervised learning is a mixture of data for supervised learning and unsupervised learning providing labeled and unlabeled data at the same time. It is easy to imagine that the analysis of such data requires other methods than those for supervised learning and unsupervised learning. For completeness, we would like to add that the terms unsupervised, supervised, and semi-supervised learning have their origin in machine learning.

Finally, we would like to add that there are more advanced learning paradigms, e.g., transfer learning or one-shot learning. Those are derived from the basic data characteristics in Fig. 2.2, however, providing additional structure. In Chap. 17, we will discuss these learning paradigms in detail.

2.2.2 Properties of the Optimization Algorithm

Unlike using the properties of the data to categorize methods, utilizing properties of the optimization algorithms provides a categorization from a theoretical perspective. For instance, a Naive Bayes classifier estimates probability distributions and identifies the maximum posterior for making predictions, whereas k-Nearest Neighbor assesses the similarity of a feature vector compared to a set of reference vectors for assigning a prediction based on voting. Examples of error-based optimization methods are neural networks or regression models. In contrast, a decision tree or random forests are examples of information-based methods.

These are just four examples which demonstrate that the internal working mechanisms of the different methods from data science can be quite different from each other from a theoretical perspective, hence disallowing to find a common denominator. As discussed above, this is another indicator of the interdisciplinarity of the field, because different areas follow different approaches. Categorizing methods according to the properties of optimization algorithms is more of theoretical than practical interest, because this does not necessarily imply specific application areas where a method could be used.

2.2.3 Properties of the Model

Finally, categorizing prediction methods according to the properties of the model is similar to the properties of optimization algorithms, but it assumes the perspective of the model structure itself rather than the means of optimization. Hence, it provides a functional point of view on the working mechanism of a model as a whole.

For instance, support vector machines (SVMs) are kernel-based methods projecting feature vectors into a high-dimensional space where, e.g., the classification is performed. In contrast, deep neural networks learn the structure of a network to, first, re-represent the feature vectors in a new space (representation learning) and, second, to classify them. Both properties (kernel and structural) provide a metaphorical visualization of the working principles of the methods. Similarly, regression, e.g., multiple linear regression, emphasizes that an input is mapped onto an output, and instance-based learning, e.g., k-Nearest Neighbor, does not store an internal model but investigates each new feature vector (instance) by a comparison with a set of reference vectors. Finally, ensemble methods, e.g., random forests, utilize the same base classifier multiple times and achieve the final decision by combining the classification results from the individual trees (many trees make a forest).

2.2.4 Summary

From the discussion above, it follows that there is no grand underlying principle unifying all prediction methods, which would enable their objective categorizations. Instead, many perspectives are possible, and each has its own merits. Hence, every categorization used is neither absolute nor unique but presents a certain perspective. In the following, we present yet another perspective on the categorization of prediction models, which we follow in this book.

2.3 Overview of Prediction Models

The number of different methods provided by data science is vast. Hence, listing them one next to another does not allow for an easy overview, nor do the three categorizations presented in the previous section. Instead, in Fig. 2.3, we show such an overview based on a mixture of application-based and model-based groups. This implies that there are cross connections between the categories. For instance, an autoencoder is a deep learning model, which can be used for clustering problems, and logistic regression is a generalized linear model (GLM), which can be used for classification problems. Similarly, survival analysis is a special case of a GLM.

We used the above categories to structure the presentation of models discussed in this book because, in our opinion, this provides the most intuitive overview of the methods. Also it allows us to emphasize the underlying systematics, e.g., of generalized linear models or hypothesis tests, from which follow many models as special cases of a larger category.

For completeness, we would like to mention that there are advanced prediction models we did not include in the above overview. For instance, ensemble methods like boosting utilize many “weak” classifiers and combine them in a way that improves the individual classifiers. Other advanced methods include adversarial networks [201]. These are deep neural networks, and the idea of this method is to let two neural networks compete with each other in a game-like setting. Overall, this establishes a generative model that can be used for producing new data in a simulation-like framework. Further methods are for causal inference and network inference, which aim to reveal structural information among features or variables.

Finally, there are methods for the modeling of data. For instance, graphical models are methods for the representation of high-dimensional probability distributions [267, 285]. Importantly, these methods are *probabilistic* (and not statistic) models, which aim, first of all, to describe a problem rather than to make predictions. This distinction is very important because from this, it follows that graphical models are not data-driven. For this reason, they do not provide core methods for data science.

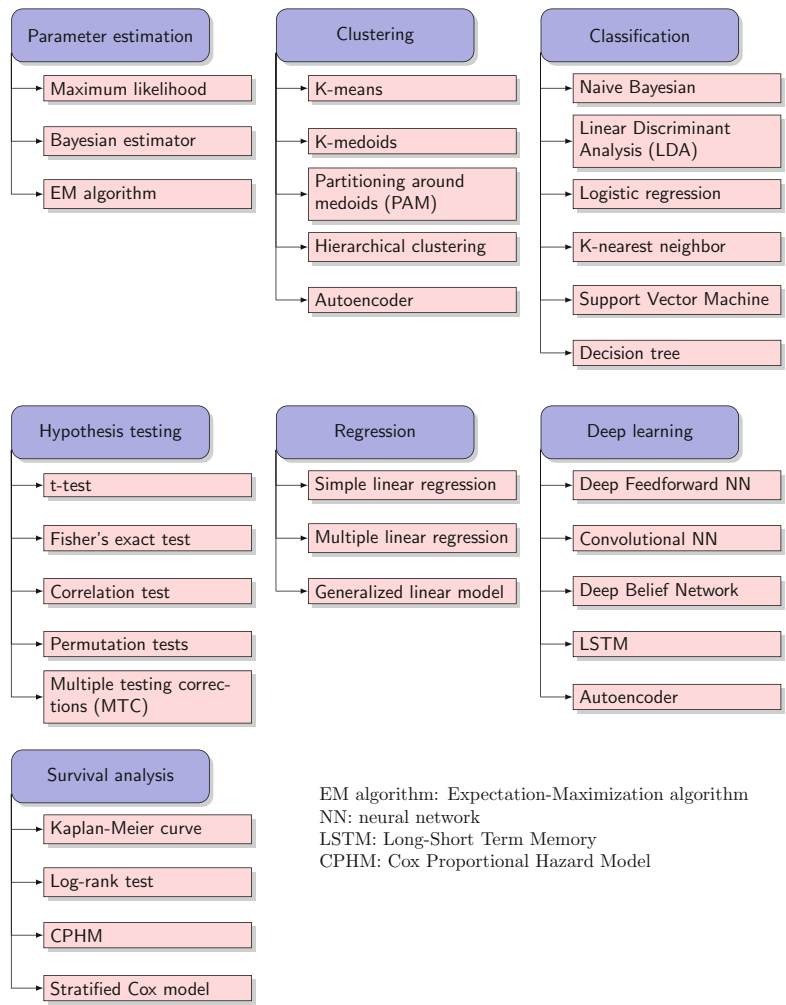


Fig. 2.3 Overview of prediction models where the main categories form a mixture of application-based and model-based groups. This allows a more intuitive overview of prediction models

2.4 Causal Model vs Predictive Model

We would like to note that there are two further model-categories originating from the statistics community [58] that are commonly distinguished. The first model type is called *causal model* (also known as inferential or explanatory model). Such a model provides a causal explanation of the data generation process. The second model type is called *predictive model*. The purpose of such models is to make forecasts for unseen instances (data points), e.g., by performing a classification or regression [432].

Certainly, an inferential model is more informative than a predictive model because an explanatory model can also be used to make predictions, but a predictive model does not provide (causal) explanations for such predictions. An example of an inferential model is a causal Bayesian network, whereas a decision tree (discussed in Chap. 9) is a prediction model. Due to the complementary capabilities of predictive and causal models, they coexist next to each other, and each is useful in its own right.

In this book, our focus is on predictive models because the beginner needs to start with intuitive and simpler models before learning advanced models. Causal models are advanced models that require a solid foundation about prediction models in order to fully appreciate their functionality. Besides this, in many practical situations, it is either impossible to estimate a causal model (for instance, due to data limitations), or it is unclear how to estimate such a model in a sound way. Hence, despite the obvious theoretical advances of a causal model over a predictive model, practically, the former is not attainable for every problem.

2.5 Explainable AI

Very recently, yet another model type emerged commonly summarized under the term *explainable AI (XAI)* [14, 150]. The goal of XAI is to generate human-understandable models, because applications in medicine, politics, or finance require the communication of the working mechanism of a model to stakeholders. Interestingly, every causal model is also an explainable model; however, the reverse is not true. Hence, this type of model is situated between a causal model and a predictive model. A practical example of an explainable model is a decision tree.

Considering the goal of XAI, one realizes that this type of model comes with a certain subjectivity, because different humans can exhibit a different understanding. Another complication comes from the fact that there are several closely related concepts to “explainability” which, however, add further subtleties [14]. Such concepts are:

1. Understandability
2. Comprehensibility
3. Interpretability

Research about XAI is currently at the very beginning, and at present many approaches are being tested, e.g., SHAP (SHapley Additive exPlanations) [318], but so far there is no generally accepted gold standard. At the moment, it is even unclear if it is always possible to substitute a *black-box* model with an explainable model or if there are possible approximations that provide a compromise between a “good prediction” and a “good explanation.”

2.6 Fundamental Statistical Characteristics of Prediction Models

In Chap. 1, we emphasized at several occasions on the importance of *statistical thinking*. This does not only refer to methods from statistics but is equally important for methods from machine learning and artificial intelligence. In this section, we want to elaborate on this by showing that the output of any prediction method is a random variable.

In Fig. 2.4 (top), we show our setup by outlining the general framework when analyzing data. This framework consists of the following three components. First, an experiment (Ex) is conducted leading to the generation of data (D). Then the data are analyzed using a method (M), leading to results (R). If one has just one data set, say D_1 , one obtains just one result, R_1 . Here, R_1 can correspond to an error measure (E), e.g., accuracy, F-score, or any other error score (for a discussion of general error measures, see Chap. 3). For the following discussion, the specific measure is not crucial, we only need to decide which one of the above we want to use. Obviously, it is possible to repeat the same experiment a second time, giving us data set D_2 and result R_2 . What would we expect from such repeated results?

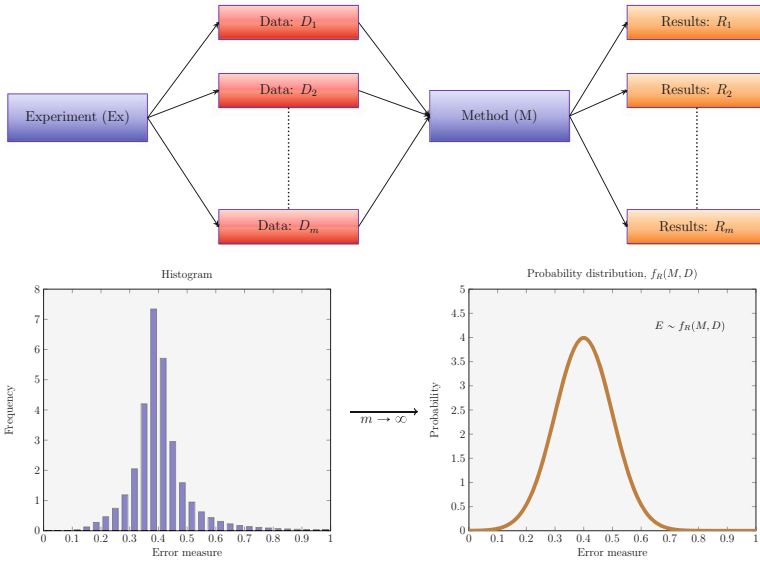


Fig. 2.4 Fundamental characteristics of prediction models. Top: An experiment (Ex) is repeated m times leading to m different data sets. Each data set is analyzed with the same method (M) leading to m different results. Bottom: The histogram shows a summary of an error measure (E), e.g., accuracy, F-score, or any other error score, while the distribution on the right-hand side is obtained in the limit $m \rightarrow \infty$, i.e., for an infinite number of experiments

Repeating an experiment m times allows us to obtain a histogram of the values of the error measure. From such a histogram (see Fig. 2.4 (bottom) for an example), one can already see a distribution emerging as a result from the repeated experiments. If one goes to the limit $m \rightarrow \infty$, conducting an infinite number of experiments, this histogram becomes actually a probability distribution for our error measure. If we call the resulting probability distribution $f_R(M, D)$ one can write

$$E \sim f_R(M, D) \quad (2.1)$$

meaning that the observed results (R) of a prediction method (M) for data (D) lead to a particular value of the error measure (E). In Eq. 2.1 ‘ $\sim$ ’ means the value of E is drawn (or sampled) from the distribution $f_R(M, D)$ (the sampling from a probability distribution is discussed in detail in Chap. 4), providing an abstract formulation for the visualization in Fig. 2.4. In statistics, a variable with this property is called a *random variable*. Importantly, this implies that the output of any prediction model, E , is a random variable.

One may wonder where does this randomness come from? From statistics, we know that when we have a random variable as input of a deterministic function, we get another random variable as output. Specifically, given x drawn from distribution f_x , i.e., $x \sim f_x$, and a deterministic function g given by

$$g : x \rightarrow y, \quad (2.2)$$

which maps x onto y , then y is also a random variable.

Given the distribution $f_R(M, D)$, one can ask which of the two input variables, i.e., M and D , of the function is the random variable? Since the method (M) is fixed, it must be the data (D). In fact for a given data set $D = \{x_i\}_{i=1}^n$ with n samples, each data point x_i is drawn from a distribution determined by the experiment (Ex), and it is given by

$$x_i \sim f_D(Ex). \quad (2.3)$$

That means the randomness in the measurement of each data point x_i translates into the randomness of the error measure E . This translation of randomness between the data and an error measure is fundamental, and it applies to any prediction model.

An important consequence of the above discussion is that now that we are aware that the output of any prediction model is a random variable, with associated underlying (but usually unknown) probability distribution $f_R(M, D)$, we need to interpret the results accordingly. This effects profoundly the way we conduct a data analysis. In Chap. 4, we return to this issue when discussing resampling methods.

2.6.1 Example

In order to gain a practical understanding of this theoretical result, let's study a numerical example of a prediction model to show that its output is a random variable. Listing 2.1 shows such an example for a t-test (discussed in detail in Chap. 10) as a prediction model.

The first part of Listing 2.1 defines our experiment. Here, we generate normal distributed data with a mean of $\mu = 0.4$ and a standard deviation of $\sigma = 0.1$. That means $x_i \sim N(\mu, \sigma)$ for $i \in \{1, \dots, n\}$, where n is the sample size. As a prediction model, we want to use a t-test for testing the following hypothesis:

Null hypothesis: The mean value of the population is 0.5: $\mu = 0.5$.

Alternative hypothesis: The mean value of the population is not 0.5: $\mu \neq 0.5$.

Listing 2.1: The output of a hypothesis test is a random variable

```
# Experiment 1
D1 <- rnorm(n = 5, mean = 0.4, sd = 0.1)
D1
[1] 0.3943265 0.4072219 0.3003603 0.2182522 0.2353866

# Experiment 2
D2 <- rnorm(n = 5, mean = 0.4, sd = 0.1)
D2
[1] 0.4282351 0.6338003 0.3208837 0.2422102 0.2464795

res1 <- t.test(D1, alternative = "two.sided", mu = 0.5)
res1$p.val
[1] 0.008483764

res2 <- t.test(D2, alternative = "two.sided", mu = 0.5)
res2$p.val
[1] 0.1607129
```

As one can see from Listing 2.1, the p-values resulting from the application of a t-test to D_1 and D_2 are $p_1 = 0.0084$ and $p_2 = 0.1607$, respectively. First, we notice that p_1 and p_2 are not identical. Second, we notice that both p-values are not even close because p_2 is almost 20 times larger than p_1 . Third, using a significance level to make a decision about the statistical significance of the results (discussed in detail in Chap. 10), we find that for $\alpha = 0.05$, p_1 is statistically significant, whereas p_2 is not. Hence, both p-values result in different decisions because we need to reject the null hypothesis based on $\alpha = 0.05$ and $p_1 = 0.0084$, whereas for $\alpha = 0.05$ and $p_2 = 0.1607$, we cannot reject the null hypothesis.

Overall, from the above numerical example, one can not only see different numerical values but different decisions that follow from declaring significance. In order to understand the severity of this, we would like to mention that hypothesis tests are frequently used to study the effect of medications (with the help of survival

analysis discussed in Chap. 16). In such a context, the above results could be interpreted as “the medication has an effect,” corresponding to rejecting the null hypothesis, or “the medication has no effect,” corresponding to not rejecting the null hypothesis. Apparently, both statements are opposing each other, and hence, both cannot be correct. Instead, one must be wrong. Again considering the fact that for both decisions the same method has been used, but different data (from the same experiment), this underlines the need for treating the output of a prediction model as random variable.

2.7 Summary

In this chapter, we presented different views on prediction models. We have seen that there are different perspectives possible, and each one is beneficial in its own right; however, none is perfect nor complete. This is a reflection of the interdisciplinary nature of data science and the lack of one underlying and guiding theoretical principle. Interestingly, this is in contrast with physics, which is based on the Hamiltonian theory allowing to derive fundamental laws in mechanics, electrodynamics, and quantum mechanics.

One important difference between statistical models is to distinguish a causal model from a predictive model. While a causal model is superior from a theoretical point of view (it provides an explanation and predictions), it is problematic from a practical point of view. One reason for the latter is that in practice, we start from data, and this requires us to learn a causal model from data. However, this turned out to be very difficult and is not even feasible in all circumstances, e.g., when there are limitations in the available data [361, 378, 445]. Hence, from a practical perspective as well as from the perspective of a beginner, predictive models are the first step when learning data science.

Furthermore, we discussed a numerical example, showing that the outcome of a prediction model is a random variable. We are aware that we may have used various terms and methods a beginner may not be familiar with. For this reason, we suggest to return to the setting outlined in Fig. 2.4 after reading about the corresponding topics in the following chapters and to repeat a similar analysis for other methods. This will allow to generate an important learning experience that holds for all prediction models.

Learning Outcome 2: Prediction Models

The output of a prediction model is a random variable, which is associated with an underlying probability distribution.

In our experience, truly understanding the meaning of this is an important step in comprehending data science in general and for working on the practical analysis of a project, because one needs to think in terms of a probability distribution when

talking about the output of a model. Hence, the above observation can serve as a guiding principle for looking at prediction models.

2.8 Exercises

1. Repeat the analysis provided by Listing 2.1.
 - Conduct this analysis for new randomly sampled data and for D_1 and D_2 shown in Listing 2.1.
 - Repeat the analysis, and record your results by plotting the percentage of rejection/acceptance as a function of the standard deviation σ .
 - What is the influence of m (number of experiments—see Fig. 2.4) on the results?
2. Modify the previous analysis, using any prediction model you are familiar with. For this you need to generate new data, which are appropriate for your method (e.g., for a classification, you need labeled data).
3. Write an R program that maps the values of $x = \{4, 2, -1, 5\}$ into y via the deterministic function

$$g : x \rightarrow y. \quad (2.4)$$

Modify this program by adding a noise term ε with $\varepsilon \sim N(\mu = 0, \sigma = 0.01)$. In R, values from a normal distribution can be drawn using the command ‘`rnorm(n=1, mean=0, sd=0.01)`’.

4. In Sect. 2.6, we distinguished between results (R) and the values of an error measure (E). Discuss this difference between R and E by explicating the relation between the contingency table and the F1-score. Hint: See Chap. 3.
5. For a medical example, in order to see the importance of explainable models (see Sect. 2.5) in the context of biomarkers, read the article [322]. Summarize and interpret these results. Hint: See a discussion in [152].

Chapter 3

General Error Measures



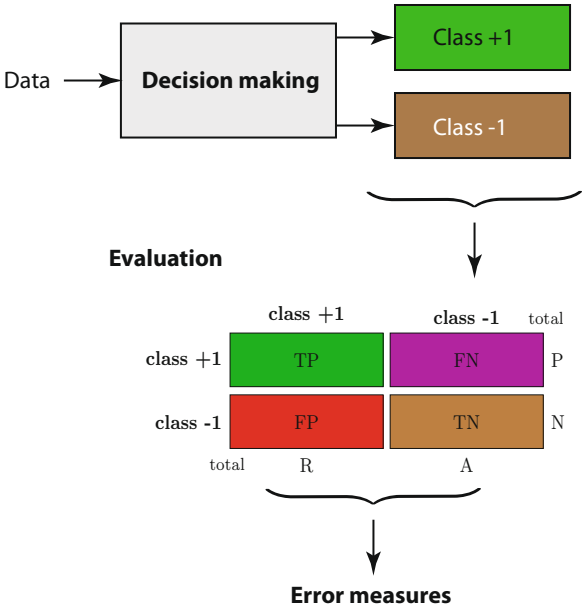
3.1 Introduction

When using a prediction method to analyze data, one needs to evaluate the outcome corresponding to the prediction. Depending on the nature of the method, there are different types of error measures one can use. In this chapter, we focus on error measures that can be used to evaluate classification methods. Classification methods are supervised learning methods that require labeled data. This allows a straightforward evaluation, because for every prediction, there is a label available, which can be used to assess if the prediction is true or false. This allows the quantification of the results of a prediction model. In this chapter, we will see that there are many error measures that can be used, and each one focuses on a particular aspect of the prediction.

There are different forms of classification, depending on the number of classes considered. The simplest and most widely used one is a binary classification (also known as two-class classification), where a data point is either assigned to class $+1$ or class -1 . In general, one can view binary classification as binary decision-making, because each sample requires us to make a decision regarding its classification. Binary decision-making is a topic of great interest for many fields, including biomedical sciences, economics, management, politics, medicine, natural sciences, and social sciences. Despite considerable differences of the problems studied within these fields, we can summarize the discussion of classification errors in terms that are useful for all application domains.

This chapter provides a comprehensive survey of well-established error measures for evaluating the outcome of binary decision-making problems. We will start by introducing four so-called *fundamental errors* from which most error measures are derived. Then, we discuss 14 different error measures. Finally, we discuss the evaluation of the outcome of a single method and of multiple methods showing that such an evaluation is a complex task, requiring the interpretation of an analysis.

Fig. 3.1 Overview of three main steps for obtaining error measures. First, binary decision-making categorizes instances either in class +1 or class -1. The result of this classification is either correct or false. Second, a repeated application allows to obtain the values in the contingency table evaluating the classification results. Third, from this, one can derive a variety of different error measures



3.2 Motivation

For the discussion of error measures, we start with a visualization of the overall problem, which is shown in Fig. 3.1. This figure shows three steps. First, data are used for *decision-making*, leading to a binary classification. Second, a repeated application of this allows to obtain the values in a so-called *contingency table*. A contingency table provides an *evaluation* of the classification results, providing a numerical summary information about the true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) (discussed in detail below). Third, based on the values in the contingency table, one can derive a variety of different *error measures*, which are functions of TP, FP, TN, and FN.

In the next section, we discuss the contingency table and its constituents, and in the following sections, we discuss error measures and their definitions.

3.3 Fundamental Error Measures

A *contingency table* (also called a confusion matrix) allows a summarization of binary decision-making. In the following, we assume that we have two classes called +1 and -1. Here, the indicators of the two classes are labels or nominal numbers (also called categorical numbers). That means the class labels do not provide numerical values but just names to distinguish the classes.

In general, the outcome of a decision-making can be one of the following four cases:

1. The actual outcome is class +1, and we predict +1.
2. The actual outcome is class +1, and we predict -1.
3. The actual outcome is class -1, and we predict +1.
4. The actual outcome is class -1, and we predict -1.

It is convenient to give these four cases four different names. We call them:

1. True positive: TP
2. False negative: FN
3. False positive: FP
4. True negative: TN

A summary of the outcomes is provided by the contingency table, shown in Fig. 3.2. If we repeat such an analysis multiple times, making predictions for a number of instances, one obtains integer values for each of the above four measures, i.e., TP, FN, FP, TN $\in \mathbb{N}$. Hence, summing over the rows or columns in a contingency table provides information about the following:

$$\text{Total number of instances in class +1 : } P = TP + FN. \quad (3.1)$$

$$\text{Total number of instances in class -1 : } N = FP + TN. \quad (3.2)$$

$$\text{Total number of instances predicted +1 : } R = TP + FP. \quad (3.3)$$

$$\text{Total number of instances predicted -1 : } A = FN + TN. \quad (3.4)$$

It is easy to see that these four quantities characterize the outcome of a binary decision-making completely. For this reason, we are calling them the four *fundamental error measures*. All the error measures we will discuss in the following sections will be based on these four fundamental errors. In order to facilitate the understanding, we will utilize the contingency table whenever beneficial for explaining the different error measures.

We would like to note that we use the term “fundamental error measures” to emphasize the importance of these four measures from all other error measures discussed in the subsequent sections. Mathematically, we will see that all the other

Fig. 3.2 Summary of binary decision-making in the form of a contingency table

		prediction outcome		
		class +1	class -1	total
actual outcome	class +1	TP	FN	P
	class -1	FP	TN	N
total		R	A	

measures are functions of the four fundamental error measures. Hence, for those measures, TP, FP, TN, and FN are *independent variables*.

3.4 Error Measures

Having the contingency table available, we have all we need for discussing the error measures. In Fig. 3.3, we show an overview of the error measures we will discuss in the following. In general, the error measures can be grouped into three main categories. The first group focuses on correct outcome, the second on incorrect outcome, and the third group on both outcomes. In the following sections, we will discuss measures from each of these categories.

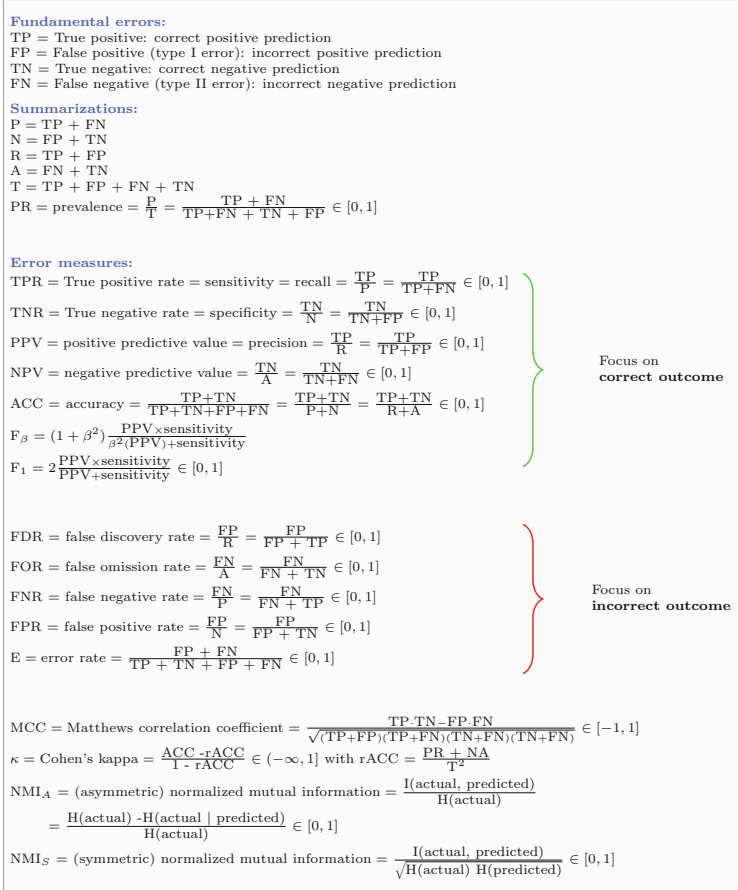


Fig. 3.3 Overview of error measures for binary decision-making and two-class classification

3.4.1 True-Positive Rate and True-Negative Rate

The true-positive rate and true-negative rate are defined by:

$$\text{TPR} = \text{True positive rate} = \text{sensitivity} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}} \in [0, 1] \quad (3.5)$$

$$\text{TNR} = \text{True negative rate} = \text{specificity} = \frac{\text{TN}}{\text{N}} = \frac{\text{TN}}{\text{TN} + \text{FP}} \in [0, 1] \quad (3.6)$$

The definitions ensure that both measures are bound between zero and one. For an error-free classification, we obtain $\text{FN} = \text{FP} = 0$, which implies $\text{TPR} = \text{TNR} = 1$. On the other hand, for $\text{TP} = \text{TN} = 0$, we obtain $\text{TPR} = \text{TNR} = 0$.

In the literature, the true-positive rate is also called *sensitivity*, and the true-negative rate is called *specificity* [167].

It is important to note that both quantities utilize only half of the information contained in the confusion matrix. The TPR uses only values from the first row and the TNR only values from the second row. In Fig. 3.4, we highlight this by encircling the used fundamental errors. For simplicity, we refer to the first row as P-level and the second row as N-level. Hence, the TPR uses only values from the P-level and the TNR values from the N-level.

From Fig. 3.4, it is clear that both measures are symmetric with respect to the utilized information, i.e., TPR and TNR merely exchange the roles of both classes. This formulation allows to remember the measures more easily.

3.4.2 Positive Predictive Value and Negative Predictive Value

The positive predictive value and negative predictive value are defined by:

$$\text{PPV} = \text{positive predictive value} = \text{precision} = \frac{\text{TP}}{\text{R}} = \frac{\text{TP}}{\text{TP} + \text{FP}} \in [0, 1]. \quad (3.7)$$

$$\text{NPV} = \text{negative predictive value} = \frac{\text{TN}}{\text{A}} = \frac{\text{TN}}{\text{TN} + \text{FN}} \in [0, 1]. \quad (3.8)$$

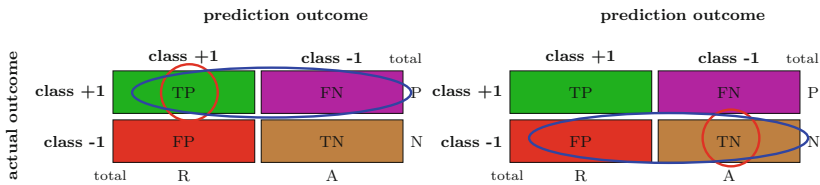


Fig. 3.4 Left: The true-positive rate (TPR) uses only information from the P-level. Right: The true-negative rate (TNR) uses only information from the N-level

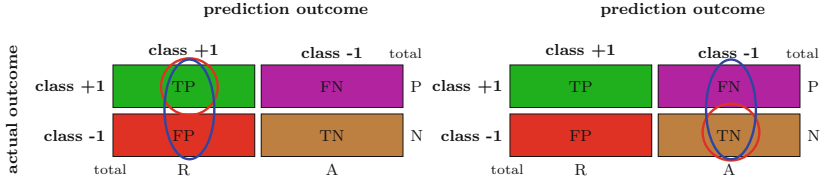


Fig. 3.5 Left: The positive predictive value (PPV) uses only information from the R-level. Right: The negative predictive value (NPV) uses only information from the A-level

The definitions ensure that both measures are bound between zero and one. For an error-free classification, we obtain $FN = FP = 0$, which implies $PPV = NPV = 1$. On the other hand, for $TP = TN = 0$, we obtain $PPV = NPV = 0$.

In the literature, the positive predictive value is also called *precision* [158].

Similar to TPR and TRN, PPV and NPV are also estimated using only half of the information contained in the confusion matrix. The PPV uses only values from the first column, and the NPV uses only values from the second column. In Fig. 3.5, we highlight this by encircling the used fundamental errors. For simplicity, we refer to the first column as R-level and the second column as A-level. Hence, the PPV uses only values from the R-level, and the NPV uses values from the A-level.

From Fig. 3.4, it can be observed that both measures are again symmetric with respect to the utilized information, and just the roles of the classes are exchanged.

3.4.3 Accuracy

The accuracy is defined as:

$$ACC = \text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{P + N} = \frac{TP + TN}{R + A} \in [0, 1]. \quad (3.9)$$

This definition ensures that the accuracy is bound between zero and one. For an error-free classification, we obtain $FN = FP = 0$, which implies $ACC = 1$. On the other hand, for $TP = TN = 0$, we obtain $ACC = 0$. Another terminology used to refer to the accuracy, in the context of clustering evaluation, is the *Rand index*.

In contrast with the quantities TPR, TNR, PPV, and the NPV, the accuracy uses all values in the confusion matrix.

3.4.4 F-Score

The general definition of the F-score is:

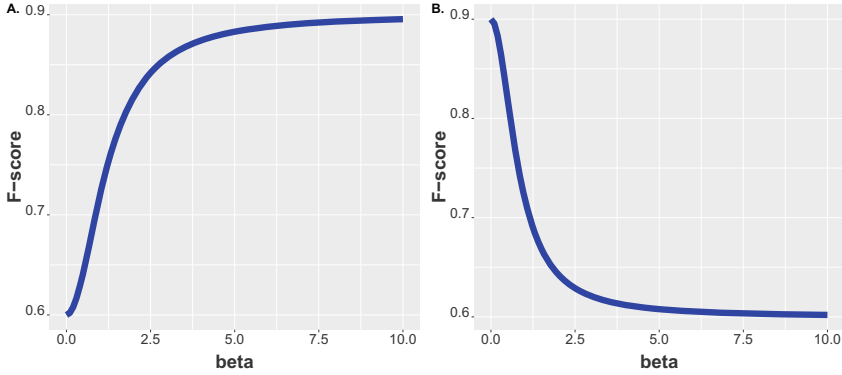


Fig. 3.6 Behavior of the F_β -score depending on the parameter β . The results are shown for PPV = 0.60 and sensitivity = 0.90 (left figure) and PPV = 0.90 and sensitivity = 0.60 (right figure)

$$F_\beta = (1 + \beta^2) \frac{\text{PPV} \times \text{sensitivity}}{\beta^2(\text{PPV}) + \text{sensitivity}}. \quad (3.10)$$

In this equation, the parameter β is a weighting parameter which can assume values in the interval $[0, \infty]$. The parameter β allows to weight the importance of the PPV and the sensitivity. Hence, the F-score is a family of measures and not just one error measure.

In Fig. 3.6, we show an example for two different value pairs of PPV and sensitivity. One can observe that for $\beta = 0$, the F-score corresponds to the PPV, whereas for $\beta \rightarrow \infty$, it corresponds to the sensitivity. Intermediate values of β enable to obtain “averaged” F-score values.

For $\beta = 1$, one obtains the *F₁-score*,

$$F_1 = 2 \frac{\text{PPV} \times \text{sensitivity}}{\text{PPV} + \text{sensitivity}} \in [0, 1] \quad (3.11)$$

The F_1 -score is the harmonic mean of PPV and sensitivity, where the harmonic mean is defined as

$$\frac{1}{F_1} = \frac{1}{n} \left(\frac{1}{\text{PPV}} + \frac{1}{\text{sensitivity}} \right). \quad (3.12)$$

The F_1 -score uses three of the four fundamental errors, namely, TP, FP, and FN.

3.4.5 False Discovery Rate and False Omission Rate

The false discovery rate and false omission rate are defined by:

$$\text{FDR} = \text{false discovery rate} = \frac{\text{FP}}{\text{R}} = \frac{\text{FP}}{\text{FP} + \text{TP}} \in [0, 1]. \quad (3.13)$$

$$\text{FOR} = \text{false omission rate} = \frac{\text{FN}}{\text{A}} = \frac{\text{FN}}{\text{FN} + \text{TN}} \in [0, 1]. \quad (3.14)$$

The above definitions ensure that both measures are bound between zero and one. For an error-free classification, we obtain $\text{FN} = \text{FP} = 0$, which implies $\text{FDR} = \text{FOR} = 0$. On the other hand, for $\text{TP} = \text{TN} = 0$, we obtain $\text{FDR} = \text{FOR} = 1$.

FDR and FOR also utilize only half of the information contained in the confusion matrix. The FDR uses only values from the first column, and the FOR uses only values from the second column. In Fig. 3.7, we highlight this by encircling the used fundamental errors. That means the FDR uses information from the R-level, but in contrast to the PPV, which uses also information from this level, the FDR focuses on failure by forming the quotient of FP and R. Similarly, the FOR uses only information from the A-level, forming the quotient of FN and A. These can be compared with the PPV and the NPV in Fig. 3.5.

From Fig. 3.7, one sees again that both measures are symmetric with respect to the utilized information and just the roles of the classes are exchanged. Frequent application domains for the false discovery rate and the false omission rate are biology, medicine, and genetics [157, 181]. When discussing multiple testing corrections in Chap. 15, we will see further examples for the application of the FDR.

3.4.6 False-Negative Rate and False-Positive Rate

The false-negative rate and false-positive rate are defined by:

		prediction outcome				prediction outcome			
		class +1	class -1	total		class +1	class -1	total	
actual outcome	class +1	TP	FN	P	class +1	TP	FN	P	
	class -1	FP	TN	N	class -1	FP	TN	N	
total		R	A		total		A		

Fig. 3.7 Left: The false discovery rate (FDR) uses only information from the R-level. Right: The false omission rate (FOR) uses only information from the A-level. Both measures focus on failure only

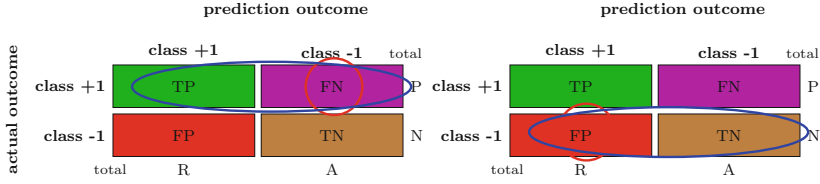


Fig. 3.8 Left: The false-negative rate (FNR) uses only information from the P-level. Right: The false-positive rate (FPR) uses only information from the N-level. In contrast with the TPR and the TNR, both measures focus on failure

$$\text{FNR} = \text{false negative rate} = \frac{\text{FN}}{\text{P}} = \frac{\text{FN}}{\text{FN} + \text{TP}} \in [0, 1]. \quad (3.15)$$

$$\text{FPR} = \text{false positive rate} = \frac{\text{FP}}{\text{N}} = \frac{\text{FP}}{\text{FP} + \text{TN}} \in [0, 1]. \quad (3.16)$$

By definition, both measures are bound between zero and one. For an error-free classification, we have $\text{FN} = \text{FP} = 0$, which implies $\text{FNR} = \text{FPR} = 0$. On the other hand, for $\text{TP} = \text{TN} = 0$, we obtain $\text{FNR} = \text{FPR} = 1$.

Both quantities are similar to the TPR and TNR, since they use only information from the first and second row of the contingency table, respectively. Specifically, the FNR uses only values from the first row, whereas the FPR uses only values from the second row. However, both measures focus on failure. In Fig. 3.8, we highlight this by encircling the used fundamental errors. These can be compared with the TPR and the TNR in Fig. 3.4.

From Fig. 3.8, it can be observed that both measures are symmetric with respect to the utilized information, and just the roles of the classes are exchanged.

3.4.7 Matthews Correlation Coefficient

A common issue when applying machine learning (ML) techniques to a real-world problem is to deal with an imbalanced target variable. In this case, the Matthews correlation coefficient (MCC) is a good measure [326]. The MCC score was first introduced by Matthews [326] to assess the performance of protein secondary structure prediction. It is defined as follows:

$$\text{MCC} = \frac{\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}} \in [-1, 1]. \quad (3.17)$$

MCC ranges from -1 to $+1$. A value of -1 indicates that the prediction is entirely wrong, whereas a value of $+1$ indicates perfect prediction. $\text{MCC}=0$ means that we have a random classification, where the model predictions have no detectable correlation with the true results.

In Fig. 3.9, we show some numerical results for the behavior of the MCC. For the shown simulations, we assumed a fixed prevalence ($=P/T$) of 0.1, a fixed sensitivity of 0.5, and $T = 10,000$. The values of the specificity were varied from 0.0 to 1.0. From these four measures, the four fundamental errors can be derived and, hence, Matthews correlation coefficient. The chosen value for the prevalence ensures a strong imbalance between both classes.

Listing 3.1 shows how to estimate Matthews correlation coefficient for given values of prevalence, specificity, sensitivity, and T (total number of instances). Since MCC is based on TP, TN, FP, and FN, these values needed to be estimated first by utilizing their corresponding definitions.

Listing 3.1: Matthews correlation coefficient given prevalence, specificity, sensitivity and T

```
MCC.fct <- function(TP, FP, FN, TN) {
  num <- (TP*TN) - (FP*FN)
  den <- (TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)

  if (den == 0) {
    return(0)
  } else {
    return(num / sqrt(den))
  }
}

T <- 10000
Prevalence <- 0.1
Specificity <- 0.5
Sensitivity <- 0.5

P <- round(T * Prevalence)
N <- T - P
TP <- round(Sensitivity * P)
FN <- P - TP
TN <- round(Specificity * N)
FP <- N - TN

# Results
Accuracy <- (TP + TN) / T
matt_corr_co <- MCC.fct(TP, FP, FN, TN)
```

In general, low specificity values correspond to poor classification results as indicated by very low ACC values and negative values of the MCC (see Fig. 3.9). With increasing values of the specificity, the ACC increases. However, due to the imbalance of the classes and the way we defined our model, keeping the TPR constant, the increasing values of ACC are misleading. Indeed, the MCC increases but not linearly and much slower than the ACC. Moreover, the highest achievable value for MCC is only 0.68 in contrast with 0.95 for ACC. Using ACC alone would not reveal this problem.

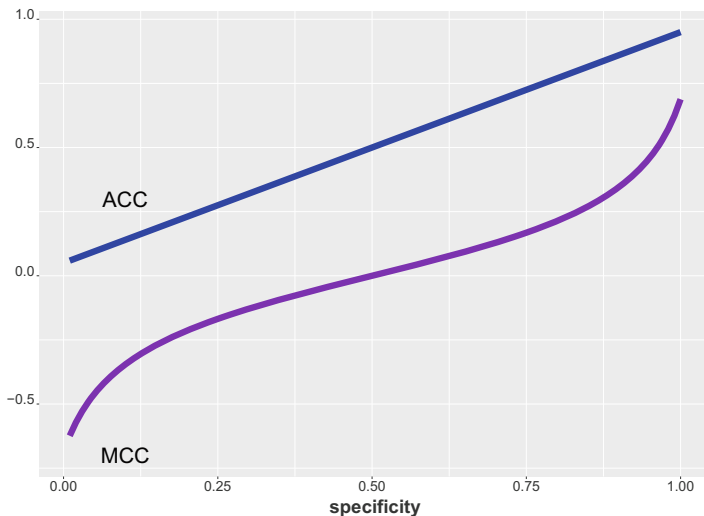


Fig. 3.9 Behavior of the MCC and ACC (accuracy) depending on the specificity and fixed values for sensitivity, prevalence, and T . For the shown results, we use a prevalence of 0.1, a sensitivity of 0.5, and $T = 10,000$

Due to the fact that an imbalance in the categories of the classes is a frequent problem, the Matthews correlation coefficient is used throughout all application domains of data science.

3.4.8 Cohen's Kappa

The next measure we discuss is called Cohen's kappa [82]. It has been essentially used as a measure to assess how well a classifier performs compared to how well it would have performed by chance. That means, a model has a high kappa value if there is a big difference in the accuracy between a model and a random model. Formally expressed, Cohen's kappa was defined in [82] as:

$$\kappa = \text{Cohen's kappa} = \frac{\text{ACC} - \text{rACC}}{1 - \text{rACC}} \in (-\infty, 1]. \quad (3.18)$$

Here, ACC is the accuracy, and rACC denotes the randomized accuracy defined as:

$$\text{rACC} = \frac{P \cdot R + N \cdot A}{T^2}. \quad (3.19)$$

Usually, Cohen's kappa is used as a measure of agreement between two categorical variables in various machine learning applications across many disciplines, including epidemiology, medicine, psychology, and sociology [409, 472].

3.4.9 Normalized Mutual Information

The normalized mutual information is an information-theoretic measure [405]. In most studies, the normalized mutual information is applied to classification tasks and other learning problems.

Specifically, Baldi et al. [20] defined the *(asymmetric) normalized mutual information* as follows:

$$\begin{aligned} \text{NMI}_A &= \frac{I(\text{actual}, \text{predicted})}{H(\text{actual})} \\ &= \frac{TP}{T} \log \frac{TP}{T} \frac{TN}{T} \log \frac{TN}{T} \frac{FP}{T} \log \frac{FP}{T} \frac{FN}{T} \log \frac{FN}{T} \\ &\quad - \frac{TP}{T} \log \left(\frac{TP+FP}{T} \frac{TP+FN}{T} \right) - \frac{FN}{T} \log \left(\frac{TP+FN}{T} \frac{TN+FN}{T} \right) \\ &\quad - \frac{FP}{T} \log \left(\frac{TP+FP}{T} \frac{TN+FP}{T} \right) - \frac{TN}{T} \log \left(\frac{TN+FN}{T} \frac{TN+FP}{T} \right) \end{aligned}$$

In fact, this expression can be simplified to [20]:

$$\begin{aligned} \text{NMI}_A &= \frac{TP}{T} \log \frac{TP}{T} \frac{TN}{T} \log \frac{TN}{T} \frac{FP}{T} \log \frac{FP}{T} \frac{FN}{T} \log \frac{FN}{T} \\ &\quad - \frac{TP}{T} \log \left(\frac{R}{T} \frac{P}{T} \right) - \frac{FN}{T} \log \left(\frac{P}{T} \frac{A}{T} \right) \\ &\quad - \frac{FP}{T} \log \left(\frac{R}{T} \frac{N}{T} \right) - \frac{TN}{T} \log \left(\frac{A}{T} \frac{N}{T} \right) \end{aligned}$$

Many variants of the normalized mutual information measure have been introduced and applied (see, e.g., [253, 388, 405, 433, 486]). For instance, Hu and Wang [253] defined the normalized mutual information measures on the so-called augmented confusion matrix. This matrix was defined by adding one column for a rejected class onto a conventional confusion matrix (see [253]). Wallach [486] considered normalized confusion matrices representing models and formulated the normalized mutual information measures on that matrix as well as other error measures such as precision and recall.

In [21], it was noted that the normalized mutual information defined above is asymmetric in the argument of the entropy because

$$\text{NMI}_A = \frac{I(\text{actual}, \text{predicted})}{H(\text{actual})} \neq \frac{I(\text{actual}, \text{predicted})}{H(\text{predicted})}. \quad (3.20)$$

For this reason, in [449], a *symmetric normalized mutual information* was suggested, and it is defined as

$$\text{NMI}_S = \frac{I(\text{actual}, \text{predicted})}{\sqrt{H(\text{actual})H(\text{predicted})}}. \quad (3.21)$$

3.4.10 Area Under the Receiver Operator Characteristic Curve

The final error measure we are presenting is the *area under the receiver operator characteristic* (AUROC) curve [53, 158]. In contrast with all the previous measures discussed so far, the AUROC curve can only be obtained via a construction process, rather than being derived directly from the contingency table. The reason for this is that the AUROC does not make use of an optimal threshold of a classifier to decide how to categorize data points (instances). Instead, this threshold can be derived, as we will show at the end of this section.

The first step of this construction process is to derive the ROC curve, and the second is the integration of this curve to obtain the area under it. To construct a ROC curve, we need to obtain pairs of the true-positive rate and the false-positive rate, i.e., $(\text{TPR}_i, \text{FPR}_i)$. That means, the ROC curve presents the TPR as a function of the FPR. Since the TPR is equivalent to the sensitivity, and the FPR is equivalent to $(1 - \text{specificity})$, an alternative representation would be the sensitivity as a function of $(1 - \text{specificity})$.

Let's assume that we have a data set with n samples. Regardless of the classification method, we can obtain either a score s_i or a probability p_i for every instance as an indicator of the membership for class +1 (and analogously values for class -1). In the following, we use probabilities, but the discussion for scores is similar. Based on these probabilities, a decision is obtained by thresholding the values. That means for

$$p_i > p^t, \quad (3.22)$$

we decide to place instance i into class +1; otherwise in class -1. Rearranging the values p_i in increasing order, we obtain

$$p[1] \leq p[2] \leq \dots \leq p[n]. \quad (3.23)$$

Now, we apply successively all possible thresholds to obtain two groups, one group corresponding to class +1 and the other group to class -1. This results overall in $n + 1$ different thresholds and, hence, groupings. This is visualized in Fig. 3.10. In

Ordered probabilities:	Thresholds:
$p[1] \leq p[2] \leq \dots \leq p[n]$	1: $p_1^t \leq p[1]$
$p[1] \leq p[2] \leq \dots \leq p[n]$	2: $p[1] < p_2^t \leq p[2]$
$\vdots$	$\vdots$
$p[1] \leq p[2] \leq \dots \leq p[n]$	$n+1: p[n] < p_{n+1}^t$
$p[1] \leq p[2] \leq \dots \leq p[n]$	

Fig. 3.10 Left: Ordered probabilities and the corresponding thresholds indicated by the vertical red line. Right: These thresholds are constrained by the equations shown above

this figure, the vertical lines in red correspond to the thresholds used to categorize instances.

It is interesting to note that these thresholds p_i^t are not unique but are given by constraints as shown on the right-hand side of Fig. 3.10. For instance, threshold p_2^t can assume any value between $p[1]$ and $p[2]$, i.e., $p[1] < p_2^t \leq p[2]$, and all such values will give the same classification results.

For each of these groupings, we can calculate the four fundamental errors and, based on these, every error measure shown in Fig. 3.3, including the TPR and FPR, which are needed for a ROC curve. Overall, this results in $n + 1$ pairs of $(\text{TPR}_i, \text{FPR}_i)$ for $i \in \{1, \dots, n + 1\}$ from which a ROC curve is constructed.

In Fig. 3.11, we show an example of a ROC curve (in purple), resulting from a logistic regression analysis. These results were generated with Listing 3.2, using the Michelin data discussed in Sect. 9.6, where we also introduce logistic regression in formally.

Here, we are only interested in the resulting ROC curve in Fig. 3.11, which shows the TPR (=sensitivity) as a function of the FPR (=1—specificity). As one can see in Listing 3.2, these values are provided by the function *roc()* included in the package *plotROC*.

The area under the receiver operator characteristic, called AUROC, in Fig. 3.11 is 0.804. For comparison purposes, we added also a diagonal line (in red). If the ROC curves would look like this, the AUROC would be 0.5, indicating a random classification performance. Furthermore, we added a blue curve, reaching to the top left corner. In this case, the AUROC would be 1.0, indicating a perfect classifier without error.

A ROC curve can be used not only to obtain the AUROC value to evaluate a classifier, but it can also be used to determine the optimal threshold for the classifier. Remember, to obtain ROC curves, such a threshold was not used. In order to obtain such an optimal threshold, we need to define an optimization function. In the literature, there are two frequent choices [473]. The first is the distance from the ROC curve to the upper left corner, i.e., $(\text{TPR} = 1, \text{FPR} = 0)$ given by

Listing 3.2: Logistic regression model for demonstrating ROC curves

```

library("plotROC") # includes the roc() function
library("marginalmodelplots") # obtain the data

data(MichelinNY)
df <- data.frame(MichelinNY)
L <- dim(df)[1]
df$Food <- df$Food + rnorm(L) # add some noise

df.train <- df[1:20,] # use as training data
df.test <- df[21:L,] # use as test data

res.logistic <- glm(InMichelin ~ Food, data=df.train,
                    family=binomial(link=logit))
pred.log <- predict(res.logistic, df.test, type = "response")

res.roc <- roc(df.test$InMichelin, pred.log)
ls(res.roc)
# output
Call:
roc.default(response = df.test$InMichelin, predictor = pred.log)

Data: pred.log in 79 controls (df.test$InMichelin No) < 65 cases (
      df.test$InMichelin Yes).
Area under the curve: 0.8039
>
> ls(res.roc)
[1] "auc"           "call"           "cases"
[4] "controls"      "direction"       "fun.sesp"
[7] "levels"        "original.predictor" "original.response"
[10] "percent"       "predictor"       "response"
[13] "sensitivities" "specificities"   "thresholds"

```

$$\text{D-ROC}_i = \sqrt{(1 - \text{TPR}_i)^2 + \text{FPR}_i^2}, \quad (3.24)$$

$$= \sqrt{\left((1 - \text{Se}_i)^2 + (1 - \text{Sp}_i)^2\right)}. \quad (3.25)$$

The second is called the Youden's Index [512] given by

$$\text{Youden-Index}_i = \text{TPR}_i - \text{FPR}_i \quad (3.26)$$

$$= \text{Se}_i - (1 - \text{Sp}_i). \quad (3.27)$$

Here, Se and Sp correspond to the sensitivity and specificity, respectively.

The optimal thresholds are then obtained by finding

$$i_{opt}^d = \underset{i}{\operatorname{argmin}}\{\text{D-ROC}_i\}, \quad (3.28)$$

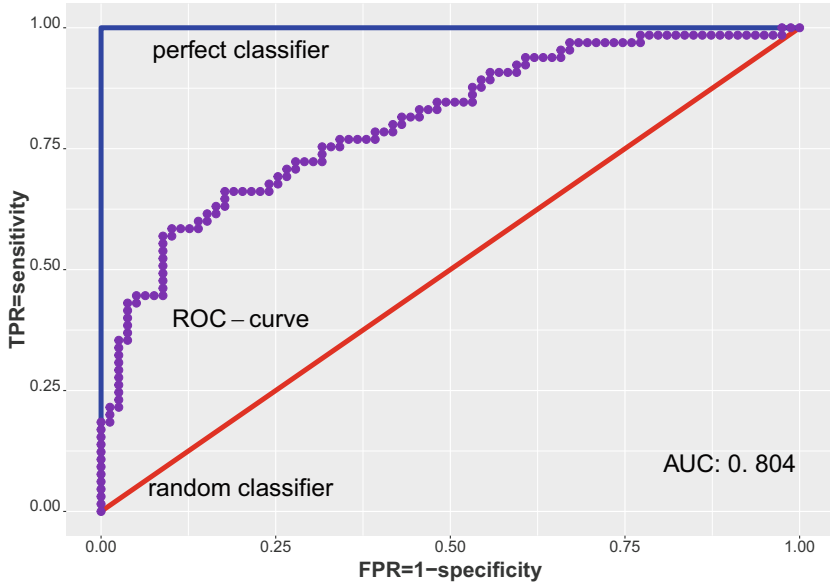


Fig. 3.11 ROC curve (in purple) for a logistic regression model. The red curve corresponds to a random classifier and the blue curve to a perfect classifier

$$i_{opt}^Y = \underset{i}{\operatorname{argmax}} \{ \text{Youden-Index}_i \}. \quad (3.29)$$

In general, these indices do not result in identical values; however, this is the case for the example shown in Fig. 3.11, as illustrated in Fig. 3.12. For the ROC curve in Fig. 3.11, this leads to $\text{TPR} = 0.66$ and $\text{FPR} = 0.17$.

3.5 Evaluation of Outcome

So far we discussed many different error measures, and one may ask, “do we really need all of them, or is there one measure that summarizes all others?” The answer to this question is that there is no single error measure that would be appropriate in all possible situations under all feasible conditions. Instead, one needs to realize that evaluating binary decision-making is a multivariate problem. This means, usually, we need more than one error measure to evaluate the outcome in order to avoid false interpretations of the prediction results.

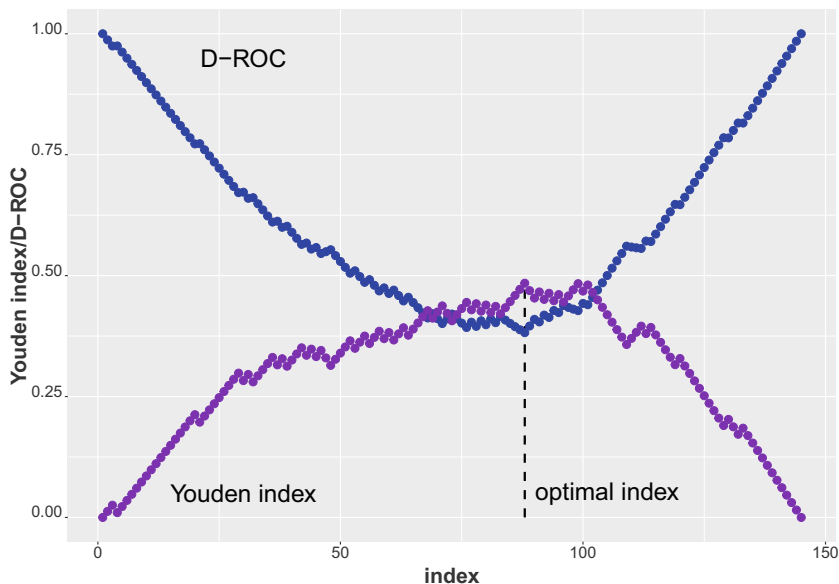


Fig. 3.12 Youden's Index (purple) and D-ROC curve (blue) depending on the index. For the logistic regression in Fig. 3.11, both measures result in the same optimal cutoff index

3.5.1 Evaluation of an Individual Method

In order to illustrate this problem, we present an example. For this example, we simulate the values in the contingency table according to a model. This means we are defining an error model. In order to simplify the analysis, we define the error model for the proportions of the four fundamental errors TP, FP, TN, and FN. Let us denote these proportions p_{TP} , p_{FP} , p_{TN} , and p_{FN} , respectively. Each of these proportions (probabilities) can assume values between zero, and one and the four quantities sum up to one:

$$p_{TP} + p_{FP} + p_{TN} + p_{FN} = 1 \quad (3.30)$$

If we multiply each of these proportions by T , the total number of instances, we recover the four fundamental errors:

$$TP = T \cdot p_{TP} \quad (3.31)$$

$$TN = T \cdot p_{TN} \quad (3.32)$$

$$FP = T \cdot p_{FP} \quad (3.33)$$

$$FN = T \cdot p_{FN} \quad (3.34)$$

Hence, there is no loss in generality by utilizing the proportions of the four fundamental errors.

In Fig. 3.13, we visualize our error model that defines the proportions of the four fundamental errors. Formally, we define the error model as shown in Listing 3.3.

Listing 3.3: Error model

```
pTP <- seq(0.2, 0.6, by=0.01)
pFN <- -(pTP)^2 + 0.4
pTN <- 1 - exp(-8*pTP) - 0.75
pFP <- 1 - (pTP + pFN + pTN)
```

The simulations start by assuming pTP takes its values in the interval from 0.2 to 0.6 in step sizes of 0.01. Based on this, the values of pFN and pTN are determined via functional relationships. Finally, the values of pFP ensure the conservation of the total probability. Overall, the model starts with a poor decision outcome, corresponding to low pTP and pTN values, and improves toward higher values corresponding to a better decision-making. Furthermore, the model is imbalanced in the sizes of the classes as can be seen from the values of P and N. It is also imbalanced in the predictions (see R and A).

In Fig. 3.14, we present results for seven error measures that are obtained using the values from our error model. The red points correspond to the starting value of the error model, i.e., pTP = 0.2, pFN = 0.36, pTN = 0.10, and pFP = 0.34 (see Listing 3.3).

All of these pairwise comparisons show nonlinearities, at least to some extent. The most linear (but not exactly linear) correspondence is exhibited between the ACC and the F-score, followed by the relationship between the FDR and the F-score. For the starting point, i.e., pTP = 0.2, etc., we obtain ACC = 0.18 and F = 0.33, indicating a poor classification performance. Taking, in addition, the value of the FDR = 0.68 into consideration, we see that 68% of all samples classified as class +1 are false. From this perspective, the classification results are even very poor. From the PPV vs NPV plot, one can see that class +1 is easier to recover compared to class -1 because there is a strong nonlinearity between these two error measures, with a faster increase in the values of the PPV (precision).

Overall, the interpretation of the classification results, given the error measures, is not straightforward because, usually, neither the best nor the worse classification results are observed. Instead, values of, e.g., the F-score or PPV are situated in between. Furthermore, it is not possible to use just one error measure to obtain the entire information of the classification performance. Instead, one needs to compare multiple error measures with each other to draw a conclusion about the performance. This will always require a discussion of the results.

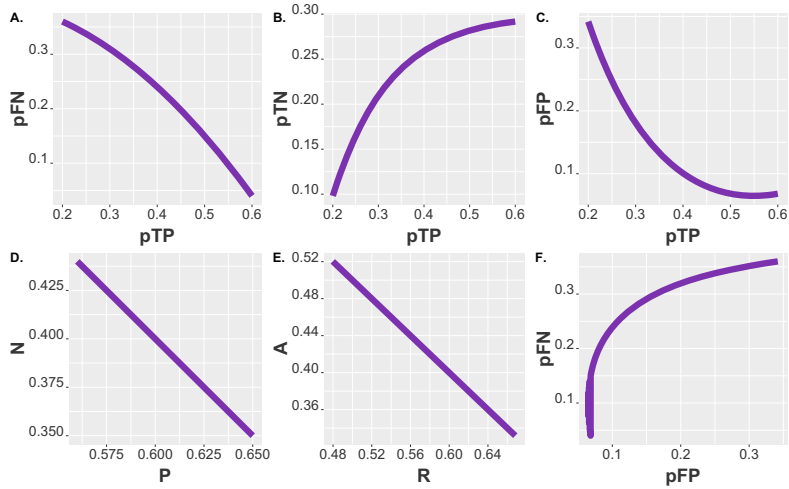


Fig. 3.13 Visualization of the error model that defines the proportions of the four fundamental errors

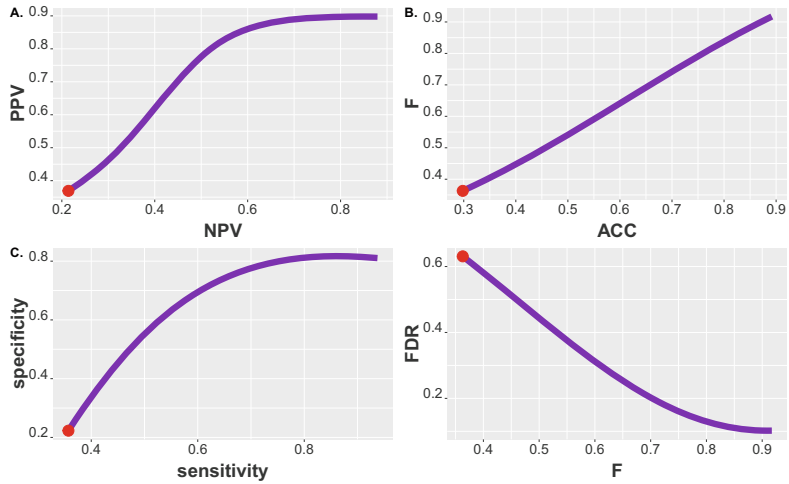


Fig. 3.14 Summary of seven error measures according to the error model shown in Fig. 3.13. The red points correspond to the starting value of the error model, i.e., $pTP = 0.2$, $pFN = 0.36$, $pTN = 0.10$, and $pFP = 0.34$

3.5.2 Comparing Multiple Binary Decision-Making Methods

In the above example, we have shown that changes in the four fundamental errors (as defined by an error model) can lead to nonlinear effects in the dependent errors. However, this type of issue is not the only problem in evaluating binary decision-

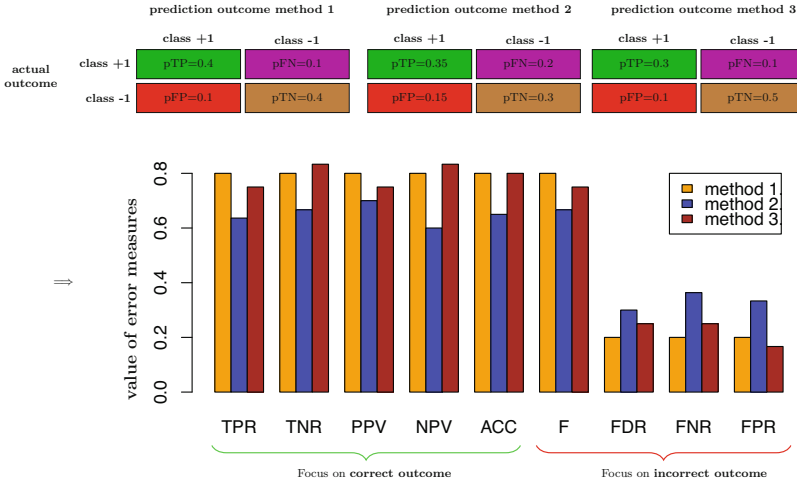


Fig. 3.15 Outcome of three binary decision-making methods. Top: Contingency tables for the three methods. Bottom: Nine further error measures for each method

making. Another problem arises when evaluating two (or more) binary decision-making methods. In order to demonstrate this type of problem, we show in Fig. 3.15 the outcome of three binary decision-making methods.

Specifically, in the first part of Fig. 3.15, we show the proportion of the four fundamental errors for three methods. Here, we assumed that the application of a method to a data set results in the shown errors, and all the three methods are applied to the same data set. In order to demonstrate this problem, we consider two scenarios. The first scenario corresponds to a comparison of the outcome of methods 1 with method 2 and the second scenario to a comparison of the outcome of method 1 with method 3.

For scenario one, from Fig. 3.15, we see that:

- $pTP_1 > pTP_2$,
- $pTN_1 > pTN_2$,
- $pFN_1 < pFN_2$,
- $pFP_1 < pFP_2$.

That means the true predictions for method 1 are always better compared to method 2, and the false predictions for method 1 are always worse compared to method 2. For this reason, it seems obvious that method 1 performs better than method 2 regardless of what fundamental error measure is used and no matter if one considers just one of these or their combinations. The result of this comparison shows that method 1 always performs better than method 2.

However, for scenario two, shown in Fig. 3.15, comparing method 1 with method 3 yields:

- $pTP_1 > pTP_3$,
- $pTN_1 < pTN_3$,
- $pFN_1 = pFN_3$,
- $pFP_1 = pFP_3$.

First, we observe that the false predictions are identical. Second, the proportion of true positives is higher for method 1, but the proportion of true negatives is higher for method 3. Third, the absolute value of the distance of the true predictions is equal for the positive and negative classes, i.e.,

$$\Delta pTP = pTP_1 - pTP_3 = 0.1, \quad (3.35)$$

$$\Delta pTN = pTN_1 - pTN_3 = -0.1, \quad (3.36)$$

but the sign is different. Overall, one can see that without further information, one cannot decide if method 1 is better than method 3 or vice versa.

This can be further clarified by looking at different error measures that are functions of the four fundamental errors. In Fig. 3.15, we show results for $N = 100$. For scenario one, we see that all error measures that focus on positive outcome are higher for method 1 (in orange) compared to method 2 (in blue) and all error measures that focus on negative outcome are lower for method 1 compared to method 2. In contrast, for scenario two, we see that there are some error measures that focus on positive outcome, which are higher for method 1 (in orange) compared to method 3 (in brown), and some are lower for method 1 (in orange) than for method 3. For instance,

$$TPR_1 > TPR_3, \quad (3.37)$$

$$TNR_1 < TNR_3. \quad (3.38)$$

Similar results hold for error measures that focus on negative outcome. For instance,

$$FDR_1 < FDR_3, \quad (3.39)$$

$$FPR_1 > FPR_3. \quad (3.40)$$

Hence, without additional information, one cannot decide which method is the best.

In summary, the above examples demonstrate the following: First, there is not one measure that summarizes the information provided by all error measures. Second, even using all of the above error measures does not guarantee the identification of the best method. As a consequence, domain-specific information about the problem, e.g., from biology, medicine, finance, or the social sciences, is needed to have a further criterion for assessing a method. The latter can be seen as introducing some further error measure(s).

3.6 Summary

In this chapter, we provided a comprehensive discussion of many error measures used in data science [146]. Such measures can be applied to common classification problems and hypothesis testing because both types of method conduct a binary decision-making.

We have seen that many error measures are based on a contingency table, which summarizes the outcome of the decision-making. Specifically, the four fundamental errors (true positive TP, false negative FN, false positive FP, and true negative TN) provide the base information from which the functional form of general error measures is derived. As we have seen, most error measures are rather simple in their definitions. On the other hand, the normalized mutual information and the receiver operator characteristic curve are more complex in their definition and estimation.

In Chap. 3.5, we have seen that the evaluation of one or more methods is not always straightforward. Despite the fact that there are many error measures, none is superior to the others in all situations, nor do all error measures taken together lead to an unique decision in all situations.

Learning Outcome 3: Error Measures

There are many error measures because they all provide a quantification for a different aspect, and the choice of the measure(s) to use for a particular analysis problem needs to be decided on a case-by-case basis.

While for many data analysis problems we will not encounter problems in interpreting the outcome of a classifier, there are cases which are hard to interpret. For those cases, domain-specific information about the problem, e.g., from biology, medicine, economy, or social sciences, needs to be taken into consideration when assessing a method. It is interesting to note that the latter can be seen as introducing further error measures. This is of course possible since the list of error measures we discussed in this chapter is not exhaustive. In fact, the number of error measures one can construct, based on the four fundamental errors, is not limited.

Finally, we would like to note that in practice, resampling methods [124, 341], e.g., cross validation, are used to estimate the variability of error measures. This is necessary because in Chap. 2, we have learned that the outcome of a prediction model is a random variable, which is associated with an underlying probability distribution. From this, one can estimate the mean value of error measures as well as the corresponding standard deviation (called the standard error). Hence, error measures are only one side of the medal when evaluating the outcome of binary decision-making. Resampling methods complement these error measures by enabling estimates of the distribution of an error measure. Resampling methods will be the topic of Chap. 4.

3.7 Exercises

1. Show that the following analytical result holds for the F-score.

$$\lim_{\beta \rightarrow \infty} F_\beta = \text{sensitivity}. \quad (3.41)$$

Hint: Expand the definition of the F-score.

2. Show numerically, by writing a program in R, that the above result is correct.
3. Extend Listing 3.1, used to estimate the Matthews correlation coefficient, in the following ways:
 - Plot the MCC for different values of the specificity.
 - Plot the MCC for different values of sensitivity.
 - Investigate the effect of the prevalence on the results. What are naturally occurring values of classification problems?
4. Write an R program to estimate the symmetric mutual information. What influence has the imbalance of classes on NMI_S ?
5. Compare the results for the symmetric mutual information with the asymmetric mutual information.
6. Construct a ROC curve manually by following the definition of its construction in Sect. 3.4.10. Start by randomly drawing 6 probabilities $p[i]$ corresponding to the membership of an instance for class +1.
 - Why are the thresholds p_i^t not unique?
 - Give some numerical examples for p_i^t and study their influence.

Chapter 4

Resampling Methods



4.1 Introduction

This chapter introduces resampling and subsampling methods. Both methods deal with the sampling of data. Due to the non-trivial nature of the latter topic, we also discuss this, in detail, in Sect. 4.7.

The methods discussed in this chapter are different from the other methods presented in this book. As we will see, resampling or subsampling methods allow the generation of “new” data sets, from any given data set, which can be used either for the assessment of a prediction model or for the estimation of parameters. Some important resampling methods, which will be discussed in this chapter, include holdout set, leave-one-out cross-validation (CV), k-fold CV, repeated k-fold CV, and bootstrap [125, 199, 422]. Subsampling is similar to resampling but for systematically reducing the number of samples. Such methods can be used for advanced applications, e.g., for the estimation of learning curves which will be discussed in detail in Chap. 18.

All resampling and subsampling methods are nonparametric methods, and this makes them flexible and easy to use. Here, nonparametric means that they are lacking sophisticated analytical formulations expressed via mathematical equations. Instead, such methods are realized numerically in a computational manner. This lack of mathematical elegance has the practical advantage of making such methods easy to understand and easy to implement computationally. Furthermore, despite this simplicity, resampling methods are powerful enough to enable us to estimate the underlying probability distribution associated with an outcome (error) of a prediction model, as discussed in Chap. 2.

The resampling methods discussed, in the following sections, can be categorized as follows:

- Resampling methods for error estimation
 - Holdout set
 - Leave-one-out CV
 - K-fold CV
- Extended resampling methods for error estimation
 - Repeated holdout set
 - Repeated k-fold CV
 - Stratified k-fold CV
- Resampling methods for parameter estimation
 - Bootstrap

We would like to highlight that in this chapter, we focus on the situation where one prediction model needs to be evaluated, and this is equivalent to model assessment. The situation where more than one model need to be evaluated will be discussed in Chap. 12 because this requires model selection and model assessment.

In addition to resampling and subsampling methods, this chapter discusses the meaning of sampling from a distribution and the standard error. Both concepts have wide implications in data science, and for these reasons, they deserve an in-depth discussion.

4.2 Resampling Methods for Error Estimation

In the following sections, we discuss resampling methods used for estimating errors of prediction models. That means such resampling methods can be applied for either classification or regression methods.

4.2.1 *Holdout Set*

The holdout set method is the simplest of all resampling methods. It is based on a two-step process, which works as follows. First, the original data are randomized, and then the data points are separated into two parts of equal size. One part is used as a training data set and the other as a testing data set. We would like to note that the proportion used as training and testing data can be a parameter, allowing other choices than 50%.

In practice, the holdout set method is not often used because its estimates are merely based on one testing data set. However, from a didactic point of view, it is instructive for the more complex methods described below. Besides this, there are extensions similar to the holdout set method, which are of practical relevance, as we will see below.

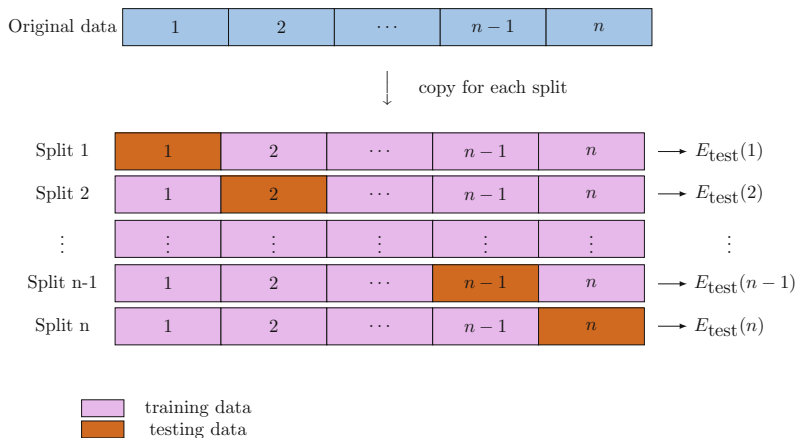


Fig. 4.1 Visualization of leave-one-out CV. The original data are copied n times, and then each copy is separated such that $n - 1$ points are used as training data and 1 point is used as test data. Importantly, the indicated test data are selected successively in a non-overlapping manner

Finally, we would like to note that there is one case in which the holdout set is the method of choice, namely, when the sample size is very large. In this case, splitting the data set in two parts leaves us with two still very large data sets which are sufficient for training and testing purposes.

4.2.2 Leave-One-Out CV

The leave-one-out cross-validation (LOO-CV) method is also called Jackknife. Assuming that the original data contain n samples, LOO-CV is a two-step process that, first, makes n identical copies of the original data and then splits each copy such that $n - 1$ points are used as training data and 1 point is used as test data, where the indicated test data are selected successively in a non-overlapping manner. The construction process of the LOO-CV method and the data splitting is visualized in Fig. 4.1.

In this figure, the original data are represented via the indices of the data points. That means for a data set $\{x_i\}_{i=1}^n$ with n samples, the shown folds (cells) contain the index of data point x_i , i.e., i . Furthermore, one can see that for each split i , a testing data set is available, and it is used to estimate the error $E_{\text{test}}(i)$ for a given method, e.g., a classifier. After all these individual test errors, for the n splits, have been estimated, they are summarized using the mean error, given by

$$E_{\text{test}}(\text{LOO-CV}) = \frac{1}{n} \sum_{i=1}^n E_{\text{test}}(i). \quad (4.1)$$

The reason why it makes sense to average over results of different data sets (given by the splits) is that $E_{\text{test}}(i)$ is a random variable, which changes its value (slightly) for different data sets. This means that there is an underlying distribution from which this random variable is drawn. Furthermore, it means that in addition to a mean value, one needs to quantify the variability of the estimate of E_{test} . This is done using the standard error, given by

$$s(E_{\text{test}}(\text{LOO-CV})) = \sqrt{\frac{\sum_{i=1}^n (E_{\text{test}}(i) - E_{\text{test}}(\text{LOO-CV}))^2}{n-1}}, \quad (4.2)$$

$$SE(E_{\text{test}}(\text{LOO-CV})) = \frac{s(E_{\text{test}}(\text{LOO-CV}))}{\sqrt{n}}, \quad (4.3)$$

where $s(E_{\text{test}}(\text{LOO-CV}))$ is the standard deviation of $E_{\text{test}}(\text{LOO-CV})$. In Sect. 4.8, we will provide an in-depth discussion of the standard error and its meaning in general.

4.2.3 K-Fold Cross-Validation

K-fold cross-validation is an extension of the LOO-CV method that increases the size of the test data at the prize of having fewer splits, resulting in fewer training data points. Specifically, k-fold cross-validation, first, randomizes the original data and makes k copies. Here, it is important to note that $k < n$. Then, each split is separated into k -folds, where $k-1$ folds are used as training data and 1 fold is used as test data. The folds corresponding to the test data are selected successively in a non-overlapping manner.

This is visualized in Fig. 4.2. As one can see, in contrast to LOO-CV, for k-fold CV, the test data sets consist of more than one data point. Consequently, the training data contain less data points compared to LOO-CV. Furthermore, it is important to note that each fold contains multiple data points as indicated by the indices of the data points. From these indices, one can also see that, for different splits, the indices in the folds do not change.

After all the individual test errors, for the k splits, have been estimated, they are summarized using the mean error, given by

$$E_{\text{test}}(\text{CV}) = \frac{1}{k} \sum_{i=1}^k E_{\text{test}}(i), \quad (4.4)$$

and the standard error, given by

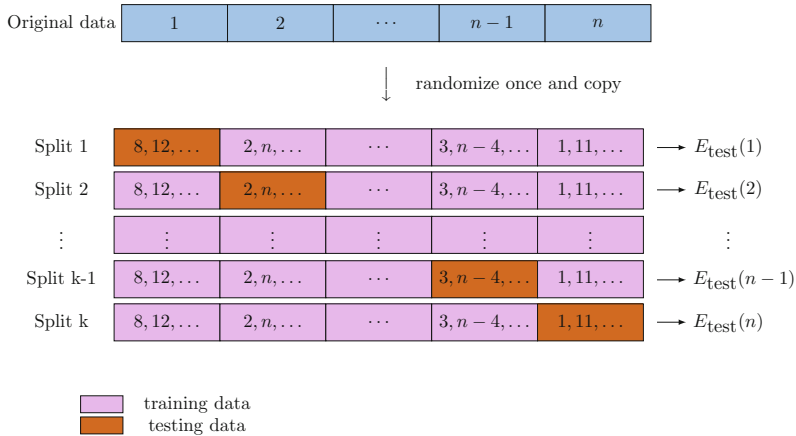


Fig. 4.2 Visualization of k-fold cross-validation. Before splitting the data, the original data points are randomized but then kept fixed for all splits

$$s(E_{\text{test}}(CV)) = \sqrt{\frac{\sum_{i=1}^k (E_{\text{test}}(i) - E_{\text{test}}(CV))^2}{k-1}}, \quad (4.5)$$

$$SE(E_{\text{test}}(CV)) = \frac{s(E_{\text{test}}(CV))}{\sqrt{k}}, \quad (4.6)$$

where $s(E_{\text{test}}(CV))$ is the standard deviation of $E_{\text{test}}(CV)$. Importantly, in contrast with LOO-CV, for k-fold CV, n has to be substituted by k , i.e., the number of folds.

On a technical note, we would like to remark that a LOO-CV corresponds to an n -fold CV where n is the number of samples in the data.

In Listing 4.1, we show an example, using R, for the randomization of data corresponding to the first step in Fig. 4.2. Specifically, Listing 4.1 shows the randomization of the indices of n data points.

Listing 4.1: Randomization of data

```
n <- 100 # sample size

ind <- sample.int(n, size = n, replace = FALSE) # indices of data
```

We would like to emphasize that it is important to set the option “replace = FALSE” to ensure each index is selected just once. This is different to the bootstrap discussed in Sect. 4.4. The difference between both is due to *resampling with replacement* and *resampling without replacement*, which will be discussed in Sect. 4.4.1.

4.3 Extended Resampling Methods for Error Estimation

The following two resampling methods provide extensions of the holdout set and the k -fold CV to improve the accuracy of the estimated errors. In contrast, the stratified k -fold CV deals with the problem of imbalanced samples.

4.3.1 *Repeated Holdout Set*

The repeated holdout set method is just a repeated application of the holdout set method, described above. Importantly, each application performs a new randomization of the original data. Also, the data separation into training and testing data is usually not done with equal proportions; instead, $2/3$ of the data points are frequently used for training and $1/3$ for testing.

The advantage of the repeated holdout set over the ordinary holdout set, discussed in Sect. 4.2.1, is that it can average over R repeats. This allows also to estimate a standard error, while this was not possible for the holdout set method.

4.3.2 *Repeated K-Fold CV*

The repeated k -fold CV set method is a repeated application of the k -fold CV method, described above. Each application of the k -fold CV method performs a new randomization of the original data.

The purpose of the repeated holdout set and the repeated k -fold CV is the same, namely, to reduce the variability of the error estimator. The choice of the most appropriate method depends on the underlying data and the sample size [278].

4.3.3 *Stratified K-Fold CV*

The stratified k -fold CV can be used for data with an additional structure. Such a structure could be provided by labeled data, i.e., instead of data of the form $\{x_i\}_{i=1}^n$, we have $\{(x_i, y_i)\}_{i=1}^n$. The simplest example would be a (multiple) classification, where the y_i corresponds to the labels of the data points x_i .

The problem is that in this case, we have at least two types of data points, one for class $+1$ and one for class -1 , but one needs data points from both classes for training and testing. However, due to the randomization of the data, it may happen that either data points from one class are completely missing in the training or testing data or they are disproportionally represented. This is especially a problem when the

classes are already imbalanced, and the randomization can amplify this imbalance even further.

For this reason, a stratification of the data points can be necessary. Here, stratification just means to perform a k-fold CV for each class (stratum) separately to ensure the same proportion of data points from the strata for the training and testing data sets.

A comparative analysis showed that stratified k-fold CV has a lower bias and lower variance compared to regular k-fold CV [284]. Similar to the repeated holdout set and the repeated k-fold CV method, these results also depend on the data and the sample size.

4.4 Bootstrap

Now we come to a resampling method that is different from all the previous methods. The bootstrap method was introduced by Efron in the 1970s, and it is one of the first computer-intensive approaches in statistics [128, 496]. In general, the bootstrap method does not lead to training and testing data. For this reason, it is not used for error estimation, which requires first to estimate the parameters of the model and then to estimate the error, but it is used for parameter estimation instead.

The working mechanism of bootstrap is as follows. The method generates B new data sets with $B \in \mathbb{N}$ that can be even larger than n . Each of these B data sets is generated by drawing n samples with replacement. This means that it is possible that data points can appear multiple times in a new data set. This implies that the number of unique data points in each new data set can be smaller than n . This is illustrated in Fig. 4.3, where in Set 1, data point 6 appears twice.

In Listing 4.2, we show an example for generating one bootstrap set. Due to the fact that we are resampling with replacement, the unique number of indices in the variable *ind* will be usually smaller than the sample size n . This can be seen using the command *unique(ind)*. It is important to note that for the bootstrap method, one does use not only the unique data points but also the duplicated ones. This duplication induced a weighting when the data are used for estimations.

Listing 4.2: Bootstrap of data

```
n <- 100 # sample size
ind <- sample.int(n, size = n, replace = TRUE) # indices of data
```



Fig. 4.3 Visualization of bootstrap. The method generates B new data sets with $B \in \mathbb{N}$ that can be larger than n . Each of these B data sets is generated by drawing n samples with replacement. This means that it is possible for data points to appear many times in a new data set (see the value “3” in Set 2)

4.4.1 Resampling With vs Resampling Without Replacement

To clarify the difference between *resampling with replacement* and *resampling without replacement*, we show, in Fig.4.4, a visualization of both resampling approaches. The top row (A) shows resampling with replacement and the bottom row (B) resampling without replacement. In the left column, the data points available before the sampling are shown. In our case, the sample size of the data is $n = 7$.

Let’s suppose we sample $m = 6$ instances from these with replacement and without replacement. The data points available after the sampling and the sampled data points are shown in the middle and the right column, respectively, in Fig.4.4. Due to the fact that resampling with replacement will replace every drawn instance, the data points after resampling (middle column) and the data points before sampling (left column) are the same. In contrast, resampling without replacement does not replace drawn instances, and for this reason, the data points after resampling (middle column) and the data points before sampling are *not* the same. Hence, the middle column shows the instances that are available for further sampling after *six* instances have been already drawn.

Also, the sampled instances, shown in the right column, are different. Resampling with replacement enables duplicate instances (see, e.g., the orange triangles), while for resampling without replacement, this is not possible. As a consequence, for resampling with replacement, the number of unique instances can be smaller than the number of drawn instances due to duplications (see Fig.4.4). In contrast, for resampling without replacement, the number of unique instances corresponds to the number of drawn samples, i.e., m . The latter means also that for resampling without replacement, m needs always to be smaller (or equal) to n .

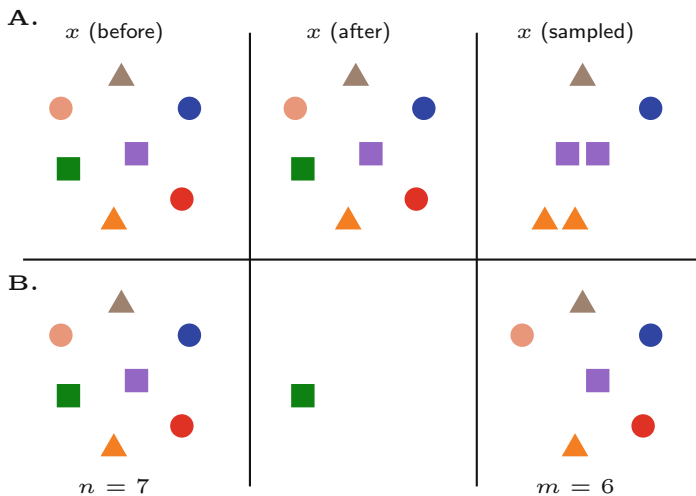


Fig. 4.4 Comparison of resampling with replacement (a) and resampling without replacement (b). The left column shows the data points available before sampling, the middle column shows the data points available after sampling *six* instances, and the right column shows the sampled data points

This example shows that there are crucial differences between both resampling approaches, and for this reason, it is important to ensure that the appropriate one is used when conducting an analysis.

In R, resampling with replacement is realized via the function `sample()` using the option “replace = TRUE,” whereas for resampling without replacement, the option “replace = FALSE” is required.

4.5 Subsampling

To estimate errors of learning algorithms as well as for parameter estimation, it is important to know how these estimates depend on the sample size. For this purpose, subsampling can be used.

The basic idea of subsampling is to systematically reduce the sample size of the original data set. Specifically, given a data set with sample size n , one reduces successively the number of samples by randomly selecting $x\%$ of the data points without replacement (see Fig. 4.5). For instance, one can obtain data sets with $\{90\%, 75\%, 50\%, 25\%\}$ of the original sample size n . These data sets can then be used with any of the resampling methods discussed above, e.g., to estimate the classification error of a method.

Due to the random selection of $x\%$ of the data points without replacement, this procedure needs to be repeated m times in order to obtain an appropriate averaging.

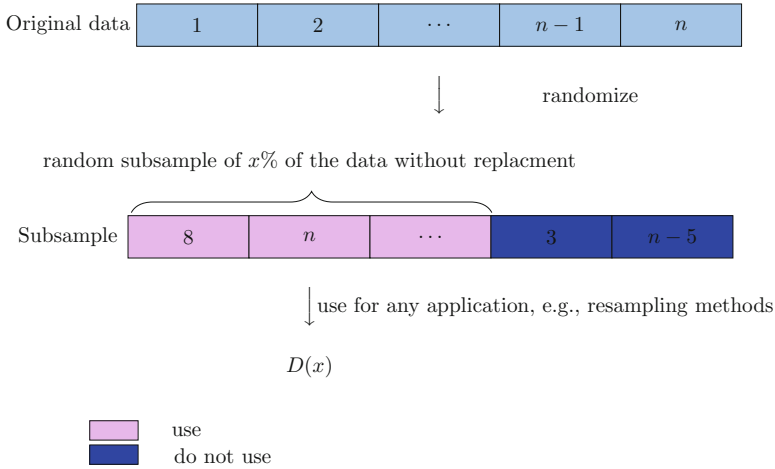


Fig. 4.5 Visualization of subsampling. The method uses a random subsample of the original data. Hence, the resulting data set contains $x\%$ of the data points without replacement

Frequently, one uses a value of m between 10 and 100, depending on the underlying data and the computational complexity of the involved methods.

In Listing 4.3, we illustrate how to subsample a data set, using R. The resulting variable *ind* contains the indices of the original data points and not their values (similar to all previous examples).

Listing 4.3: Subsampling of data

```
n <- 100 # sample size
x <- 0.7 # x% of data
m <- round(x*n)

ind <- sample.int(n, size = m, replace = FALSE) # indices of data
```

4.6 Different Types of Prediction Data Sets

In the previous sections, we distinguished between two different data sets: training data and test data (also called testing data). In addition, we would like to note that there are two further data sets to distinguish (see also Chap. 18):

- In-sample data
- Out-of-sample data

Both of these data sets are associated with the predictions of a model. Specifically, if one uses the training data set as the testing data set, i.e., if one makes double use of the training data, then such a set is called *in-sample data*. In contrast, if the training data set is different from the testing data set, then the testing data are also called *out-of-sample data*. Hence, the distinction does not introduce a new data set, but it just clarifies the role of a data set with respect to predictions of a model. Hence, a prediction is based on either in-sample data or out-of-sample data.

4.7 Sampling from a Distribution

As seen in the previous sections, when applying any resampling method, some form of random assignment of the data points is required. However, formally, this means that one needs to draw values from a probability distribution. In this section, we want to take a more detailed look at what it means to draw a sample from a probability distribution. In statistics, this is called sampling from a distribution. In the following, we start with the simplest way of sampling from a distribution which can be even done without a computer. Then we show how to translate this strategy into a computational form, which can be easily executed using R or any other programming language.

In Fig. 4.6, we show an example of a continuous probability distribution f . In order for f to be a probability distribution, it needs to hold that $f(x) \geq 0$ for all x within its domain D_x and $\int_{D_x} f(x')dx' = 1$. It is always possible to divide the domain D_x into a grid of $m + 1$ points of equal distance $\Delta x = x_{i+1} - x_i$ as shown in Fig. 4.6. This can be used to convert the continuous probability distribution into a discrete probability distribution, where each of the m intervals, Δx , is assigned a probability according to

$$p_i = \int_{x_i}^{x_{i+1}} f(x')dx' \quad (4.7)$$

with $i \in \{1, \dots, m\}$. By defining $x'_i = (x_i + x_{i+1})/2$, one can assign

$$p_i = \text{Prob}(X = x'_i) \quad (4.8)$$

to all $i \in \{1, \dots, m\}$, which defines a discrete probability distribution formally. It is clear that in the limit $m \rightarrow \infty$, the width of the intervals goes to zero, i.e., $\Delta x \rightarrow 0$, and one recovers the original continuous probability distribution.

We performed the above derivation to show that it is sufficient to have a sampling procedure for a discrete probability distribution because every continuous probability distribution can be approximated with a discrete one.

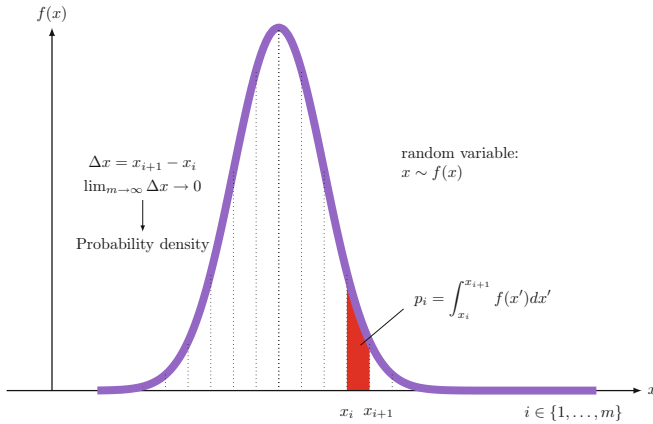


Fig. 4.6 Given a continuous probability distribution f , and using a grid, one can convert f into a discrete probability distribution

Now, let's assume that we have such a discrete probability distribution as defined in Eq. 4.8. The simplest way to utilize this discrete probability distribution is in a physical sampling process, e.g., via an urn experiment. Specifically, let's assume that we have an urn with N balls. Since our discrete probability distribution assumes m different values because $i \in \{1, \dots, m\}$, we need to label $N_i = N \cdot p_i$ balls with label "i." Here, a label could be either a name or a color. Either way, the N_i balls need to be recognizable as similar. Due to the fact that, usually, $N_i = N \cdot p_i$ will not result in an integer number, we need to round N_i toward the nearest integer. This labeling process is repeated for all $i \in \{1, \dots, m\}$ until all N balls received one label. Finally, we put all balls in an urn. By drawing one ball from this urn blindfoldedly, we simulate $x_i \sim P$, which corresponds to sampling from distribution P with probabilities given by Eq. 4.8.

The above procedure can be summarized as follows:

1. N : number of balls
2. Number of balls with label "i": $N_i = N \cdot p_i$ (rounding needed)
3. Place all balls in an urn
4. Draw one ball randomly (blindfolded) from the urn

As one can see, the above sampling procedure does not require a computer but can be done with balls and an urn. Hence, it is a purely physical (mechanical) process. However, the above procedure can be easily converted into a computational form. This is shown in Listing 4.4.

Listing 4.4: Drawing samples from a distribution with R

```

P <- c(0.15, 0.3, 0.2, 0.35)
m <- length(P)
N <- 101
Np <- round(N*P) # labeling of the balls
N <- sum(Np) # in case the rounding causes a difference

coin.aux <- runif(1)
coin <- N*coin.aux # random drawing

label <- c()
flag <- 1
i <- 1
cs <- cumsum(Np)
while(flag){
  if(coin <= cs[i]){
    label <- i
    flag <- 0
  }
  i <- i + 1
}
print(c(label, coin))

```

Following Listing 4.4 line by line, one can see how the instructions of the above (mechanical) procedure are realized.

Despite the simplicity of Listing 4.4, it can be further simplified by using the function `sample()` available in R. Specifically, by using the option “prob,” one can draw a sample from an arbitrary discrete probability distribution, as illustrated in Listing 4.5. When using the function `sample()`, it is important to set the option “replace = TRUE” to enable the draw of the same value more than once.

Listing 4.5: Drawing samples from a distribution with `sample` function

```

P <- c(0.15, 0.3, 0.2, 0.35)
sample(1:4, 10, replace = TRUE, prob = P)

```

Finally, we would like to note that R offers of course predefined functions to sample from, and some examples are given in Listing 4.6. However, these functions hide the complexity and make it unclear for the beginner how sampling actually works. For this reason, we presented the above details.

Listing 4.6: Functions in R for sampling from specific distributions

```

n <- 3
rnorm(n, mean = 2, sd = 1) # normal distribution
rgamma(n, shape = 2, rate = 1) # Gamma distribution

```

Further examples of probability distributions available in R can be found in [149].

4.8 Standard Error

In the above sections, we have seen that resampling methods provide means to estimate the probability distribution associated with the output of a prediction model since such an output is a random variable. Usually, this probability distribution is not estimated explicitly, but it is characterized by its mean and its variance.

For instance, a k -fold CV estimates the mean error based on k folds, i.e.,

$$E_{\text{k-CV}} = \frac{1}{k} \sum_{i=1}^k E_i. \quad (4.9)$$

This means that the E_i for $i \in \{1, \dots, k\}$ are drawn from an underlying probability distribution P , i.e., $E_i \sim P$, and by changing the training and testing data, the statistical model will yield (slightly) different error values.

In the following, we will derive a general result for the standard deviation of the mean in Eq. 4.9, which is called the *standard error*. Since the standard deviation (σ) corresponds to the square root of the variance (σ^2), we are using both terms interchangeably.

In order to derive the standard error, let's define a formal setting because the results will not only hold for the mean of errors but for general mean values. For this reason, let's define the mean and variance by

$$Y = \frac{1}{n} \sum_{i=1}^n X_i, \quad (4.10)$$

$$\mu_Y = \mathbb{E}[Y], \quad (4.11)$$

$$\sigma_Y^2 = \mathbb{E}[(Y - \mu_Y)^2]. \quad (4.12)$$

Here, Y is the sample mean of the data $\{X_1, \dots, X_n\}$ and μ_Y its expectation value, σ_Y^2 the variance of Y , and σ_Y the standard deviation of Y . For clarity reasons, we would like to highlight that the expectation value of the mean μ_Y is also called the population mean.

The difference between the *sample mean* Y and the *population mean* μ_Y is that the former is based on a finite sample of size n given by $\{X_1, \dots, X_n\}$, whereas the latter is evaluated by using the entire population. This requires knowledge about the probability distribution P from which the Y is drawn, i.e., $Y \sim P$, which is only known in theory. That means a data sample is not sufficient.

There is one further distribution, Q , from which the data points are drawn $X_i \sim Q$. This distribution Q is characterized by

$$\mu = \mathbb{E}[X] \quad (4.13)$$

$$\sigma^2 = \mathbb{E}[(X - \mu)^2]. \quad (4.14)$$

Here, μ is the mean value of X and σ^2 is the variance of X .

First, let's derive a result for the population mean of Y .

$$\mu_Y = \mathbb{E}[Y] = \mathbb{E}\left[\frac{\sum_{i=1}^n X_i}{n}\right] \quad (4.15)$$

$$= \frac{1}{n} \mathbb{E}\left[\sum_{i=1}^n X_i\right] = \frac{1}{n} \sum_{i=1}^n \mathbb{E}[X_i] \quad (4.16)$$

$$= \frac{1}{n} \sum_{i=1}^n \mathbb{E}[X] = \frac{n}{n} \mathbb{E}[X] \quad (4.17)$$

$$= \mu. \quad (4.18)$$

This shows that the population mean of Y is the same as the population mean of X .

At first, it may appear confusing why we estimate a “mean of a mean,” but $\mathbb{E}[Y]$ is exactly this. The reason for this is that since the X_i are random variables, Y is also a random variable, and for both variables, a mean can be estimated. The difference is that the X_i are drawn from Q , whereas Y is drawn from P . Hence, the underlying probability distributions of these two random variables are different.

Now, let's repeat such a derivation for the (population) variance of Y .

$$\sigma_Y^2 = \mathbb{E}\left[(Y - \mu_Y)^2\right] = \mathbb{E}\left[(Y - \mathbb{E}[Y])^2\right] \quad (4.19)$$

$$= \mathbb{E}\left[\left(\frac{1}{n} \sum_{i=1}^n X_i - \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^n X_i\right]\right)^2\right] \quad (4.20)$$

$$= \frac{1}{n^2} \mathbb{E}\left[\left(\sum_{i=1}^n X_i - \mathbb{E}\left[\sum_{i=1}^n X_i\right]\right)^2\right] \quad (4.21)$$

$$= \frac{1}{n^2} \mathbb{E}\left[\left(\sum_{i=1}^n X_i - \sum_{i=1}^n \mathbb{E}[X_i]\right)^2\right] \quad (4.22)$$

$$= \frac{1}{n^2} \mathbb{E}\left[\left(\sum_{i=1}^n X_i - \sum_{i=1}^n \mathbb{E}[X]\right)^2\right] \quad (4.23)$$

$$= \frac{1}{n^2} \mathbb{E}\left[\left(\sum_{i=1}^n X_i - \sum_{i=1}^n \mu\right)^2\right] \quad (4.24)$$

$$= \frac{1}{n^2} \mathbb{E} \left[\left(\sum_{i=1}^n (X_i - \mu) \right)^2 \right] \quad (4.25)$$

$$= \frac{1}{n^2} \mathbb{E} \left[\sum_{i=1}^n (X_i - \mu)^2 \right] \quad (4.26)$$

$$= \frac{1}{n^2} \sum_{i=1}^n \mathbb{E} \left[(X_i - \mu)^2 \right] \quad (4.27)$$

$$= \frac{1}{n} \mathbb{E} \left[(X - \mu)^2 \right] = \frac{1}{n} \sigma^2 \quad (4.28)$$

In summary, the above result means that there is a general relationship between the (population) standard deviation of an observation X and the (population) standard deviation of Y , which is given by

$$\sigma_Y = \frac{\sigma}{\sqrt{n}}. \quad (4.29)$$

It is important to emphasize that this result is based on expectation values, which provide population estimates. However, when dealing with data, we need to have sample estimates for the corresponding entities. For μ and σ , the sample estimates are given by

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i, \quad (4.30)$$

$$s = \frac{1}{n-1} \sqrt{\sum_{i=1}^n (X_i - \bar{X})^2}. \quad (4.31)$$

Overall, this gives the sample estimate of Eq. 4.29 given by

$$\text{SE} = \frac{s}{\sqrt{n}}. \quad (4.32)$$

Due to the importance of this result, the sample estimate of σ_Y has its own name, and it is called the *standard error*.

The important result of the above derivation is that whenever we estimate the mean value, e.g., based on prediction errors $\{E_i\}_{i=1}^n$, one can estimate the (sample) standard deviation of this mean, which corresponds to the standard error. For practical applications, the results from resampling methods should be always summarized using the mean prediction error and its standard error.

4.9 Summary

In this chapter, we distinguished two applications for resampling methods: (A) error estimation and (B) parameter estimation. Regarding error estimation, the results from the resampling methods can be summarized as follows.

$$E_{\text{HOS}} = E \quad (4.33)$$

$$E_{\text{LOO-CV}} = \frac{1}{n} \sum_{i=1}^n E_i \quad (4.34)$$

$$E_{\text{k-CV}} = \frac{1}{k} \sum_{i=1}^k E_i \quad (4.35)$$

$$E_{\text{r-HOS}} = \frac{1}{m} \sum_{i=1}^m E_i \quad (4.36)$$

$$E_{\text{r-k-CV}} = \frac{1}{m} \sum_{i=1}^m E_{\text{k-CV},i} \quad (4.37)$$

Here, E_i are the errors obtained from the testing data set i , n corresponds to the sample size of the original data, and m is the number of repetitions of a resampling method. As one can see, the holdout set (HOS) is the only method that does not average over errors since this method allows only to estimate one such error.

From the discussion on the various resampling methods, we have seen that there are many similarities. For instance, when setting $k = n$ (the number of samples) for the k -fold CV, this results in LOO-CV. In general, when choosing k , there are two competing effects. For large values of k , the bias of the true error estimate is small but the variance of the true error estimator is large, whereas for small values of k , the situation is reversed. In practice, common choices are $k = 5$ or $k = 10$. The general drawbacks of cross-validation can be summarized as follows:

- The computation time can be long because the entire analysis needs to be repeated k times for each model.
- The number of folds (k) needs to be determined.
- For a small number of folds, the bias of the estimator will be large.

For situations where the data are very limited, leave-one-out CF (LOO-CV) has been found to be advantageous [341].

For repeated resampling methods, e.g., repeated k -fold CV or repeated holdout set, the problems are similar because a large number of repetitions m reduce the variance of the estimates but introduce a computational burden.

Learning Outcome 4: Resampling Methods

Resampling and subsampling methods do not form standalone analysis methods, but they complement others (for instance, prediction models) by allowing to “simulate” repeated experiments.

This implies that resampling and subsampling methods allow to estimate the underlying probability distribution associated with the outcome of a prediction model (see Learning Outcome 2 in Chap. 2) for estimating the (mean) error and the standard error.

Finally, we would like to remark that despite the fact that there are many technical variations of cross-validation and other resampling methods (e.g., bootstrap) to improve the estimates [13, 290, 341], the discussed approaches in this chapter are frequently utilized in practical data science projects.

4.10 Exercises

1. Study the number of unique instances obtained from resampling with replacement. Start with $n = 100$ data points, and estimate the percentage of unique instances in dependence on m drawn samples for $m = \{10, 20, 50, 75, n\}$. In order to obtain stable estimates, an averaging is needed. How do the results change when varying n ? Hint: Extend the code shown in Listing 4.7.

Listing 4.7: Resampling with replacement

```
n <- 100 # sample size
m <- 10
ind <- sample.int(n, size = m, replace = TRUE) # indices of data
```

2. Generate a data set with $n = 100$ samples. For this data set, perform a subsampling by drawing $x = \{90\%, 60\%, 40\%\}$ samples.
3. Convince yourself that the standard error is a standard deviation, however, not for one observation but for the sample mean.
4. Given the standard deviation of X and of Y as in Eqs. 4.14 and 4.12, what is the limit of the corresponding sample estimates for the standard error and the sample standard deviation?

$$\lim_{n \rightarrow \infty} SE = ? \quad (4.38)$$

$$\lim_{n \rightarrow \infty} s = ? \quad (4.39)$$

Chapter 5

Data



5.1 Introduction

When learning how to analyze data, it may appear natural to focus entirely on methods. However, methods provide only one part of the story since they cannot be used without data. Unfortunately, until recently, when the field data science emerged, “data” were severely underappreciated by the research communities giving the false impression that knowledge about data is less important than the understanding of a method. In order to counteract this impression, we dedicate a chapter exclusively to data at the beginning of this book.

In this chapter, we provide an overview of different data types. We will see that the data generation processes (for various data types) can be fairly different. Here, we are not interested in acquiring enough knowledge to conduct experiments by ourselves. Instead, our focus is on a theoretical understanding that explains the generation of the data and the underlying processes behind them. We believe that a sensible data analysis is only feasible if one has a sufficient understanding of the underlying phenomena, the data generation process, and the related experimental measurements. For this reason, we describe in this chapter five different data types and the fields from which they come. The descriptions will be brief, and for professional data analysis projects, much more details may be needed. However, the discussed examples should be sufficient to see a general pattern when dealing with data.

In general, when working on a data analysis project, the data always require some work regarding:

- gathering
- loading/storing
- cleaning
- preprocessing
- understanding

This can become fairly complex and time-consuming, requiring weeks or even months of work. Hence, the discussion in this chapter is only exemplary for general data science projects to demonstrate the importance of the above steps.

At the end of this chapter, we will realize that “data” come in many shapes and forms and, hence, are very heterogeneous and situational with respect to the phenomena they are associated with. All of this will make it clear that the understanding of data needs to be taken seriously when conducting a data science project.

5.2 Data Types

In the following, we discuss five different data types: genomic data, network data, text data, time-to-event data, and business data. Some of these data types are field-specific, whereas others can occur in many different fields. For instance, genomic data occur in biology, medicine, or pharmacology which are all life sciences. The life sciences are fields that study living organisms, like plants, animals, or human beings. In contrast, network data are not field-specific, but they can be found in any field, including life sciences, physics, chemistry, social sciences, finance, business, and economy.

5.2.1 Genomic Data

The Human Genome Project did not only result in the sequencing of the human genome, but it sparked also the development of molecular measurement devices [391]. These measurement devices transformed biology in the early 2000s into a technological field with the capability to generate large amounts of data. In contrast with traditional experiments in biology, targeting only one or a few molecular entities within biological cells at a time, the new measurement devices perform high-throughput recordings of thousands or even tens of thousands of genes or proteins. Importantly, such experiments can be performed not only in biology but also in medicine and pharmacology to study human diseases or effects of drugs on treatment. Hence, nowadays, the research in life sciences is usually data-driven due to the flood of available data resources enabled by powerful high-throughput technologies.

According to the central dogma of molecular biology [89], every biological cell consists of three fundamental information carrying levels: genes, mRNAs (messenger ribonucleic acids), and proteins. The genes provide information about functional units stored in the DNA (deoxyribonucleic acid) also called genome. If activated, genes are transcribed into mRNAs, which are then translated into proteins. This flow of information is visualized in Fig. 5.1, showing a simplified biological

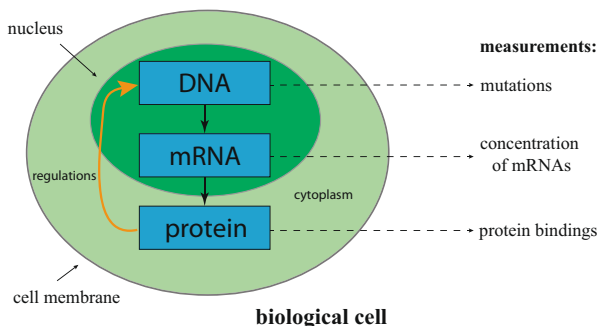


Fig. 5.1 Simplified visualization of a biological (eukaryotic) cell. Every cell contains three fundamental information carrying levels: DNA, mRNA, and protein. On each level, measurements about the state of a cell can be conducted

cell. Due to the fact that this is common to all biological cells, the information exchange between the three levels, i.e., DNA, mRNA, and protein, is fundamental.

We would like to highlight that there is also a connection between the protein level and the DNA level (see the orange arrow in Fig. 5.1). Biologically, this is provided by specific types of proteins, which are called *transcription factors* (TFs). A TF binds to a particular region on the DNA (called promoter region) to regulate the activation of a gene (for more information about this, see the discussion about gene regulatory networks in Sect. 5.2.2). Hence, the connections between the three levels are circular, which can give rise to very complex dynamical processes.

Regarding the generation of data, the central dogma of molecular biology informs us also about possible measurements, which can occur on these three levels. For instance, on the DNA level, we can obtain information about mutations of genes, i.e., changes in the sequence of nucleotides that are forming genes. On the mRNA level, one can measure the concentration of mRNAs within a cell, and on the protein level, one can either measure the binding among two or more proteins or their three-dimensional structure via crystallography. These are just a few examples, and for each of them, there are dedicated measurement devices, allowing to measure this molecular information for thousands of such entities. As an order of magnitude, we want to mention that humans have about 22000 genes (the exact number is, to this day, not known).

In the following, we will focus on data providing information about the measurement of the concentration of mRNAs. Such genomic (or omic) data are called *gene expression data*. There are two major technologies which can be used for measuring mRNAs: DNA microarrays and RNA-seq. The latter is a next-generation sequencing (NGS) technology [489], whereas the former is based on hybridization [425]. In general, measuring mRNAs is important because they allow studying the functioning of cells. The reason is that every cell of an organism contains the same amount of DNA (collection of all genes); however, not all genes are active at all times, neither are the same genes active in different cell types, e.g., breast cells

or neurons. mRNAs allow on the one hand to identify active genes and on the other hand to conclude on the presence of proteins. The latter is important because proteins are the active units in cells, which are required for performing all the work so that an organism can function properly.

Despite the fact that DNA microarray and RNA-seq technologies measure mRNAs, the preprocessing steps from raw data to normalized data, which can be used for an analysis, are considerably different. However, a commonality is that both preprocessing pipelines are very complex. This means that the corresponding pipelines can be structured into subproblems for which R packages exist for their processing. Note that each of these packages is typically the result of a PhD thesis. This should give an impression of the complexity of the preprocessing and explain why a detailed discussion of these steps is beyond the scope of this book. However, in Fig. 5.2, we show an example result of such an analysis.

Specifically, the figure shows a *heatmap* of expression data for 106 genes (rows) for 295 breast cancer patients (columns) [475]. A heatmap assumes a matrix form, M , where the rows correspond to genes and the columns to samples, which in our case are patients.

Definition 5.1 (Gene Expression Data) The expression matrix, M , is numerical, i.e., $M_{i,j} \in \mathbb{R}$, with $i \in \{1, \dots, g\}$ and $j \in \{1, \dots, s\}$ where g is the total number of genes and s is the total number of samples.

If expression data for all mRNAs (protein coding and non-coding) are available, such data are called *transcriptomics data* or transcriptome. For completeness, we would like to mention that the entire information about all proteins is called *proteomics data* and the entire information about all genes is called *genomics data* (classically called genetics).

The heatmap in Fig. 5.2 clusters (clustering is discussed in detail in Chap. 7) the patients according to the similarity of expression values of their genes. While the evaluation of a similar clustering for genes would be very complex, the evaluation of the clustering for the patients is simple, because each patient can be medically categorized according to a “good prognosis” (red) or “poor prognosis” (green) with respect to the overall survival. In Fig. 5.2, this is visualized on the top of the heatmap, where one can see that the two main clusters are fairly good but not perfect. Overall, this is a good example for an exploratory data analysis (EDA), already mentioned in Chap. 1 and further elaborated in Chap. 6. This also shows that a visualization is the first step for gaining deeper insights about the underlying meaning of data.

5.2.2 Network Data

Network data appear when a system contains variables that are heterogeneously connected with each other. In contrast to genomic data, this is not limited to a field but can occur in a variety of fields. Examples of areas where network

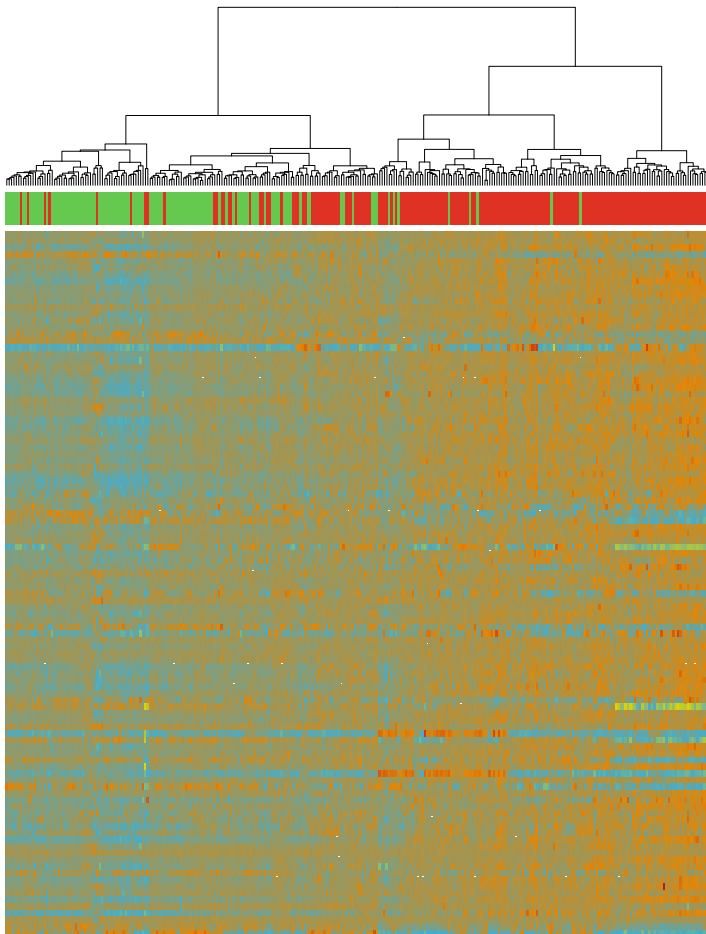
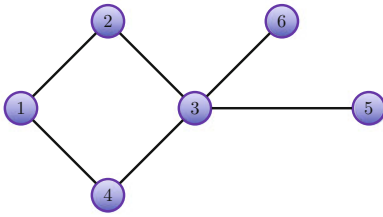


Fig. 5.2 Heatmap of gene expression data. Shown are the expression values of 106 genes (rows) for 295 breast cancer patients (columns) [475]. The patients are categorized on the top according to “good prognosis” (red) and “poor prognosis” (green) with respect to survival

data are observed include chemistry, biology, economy, finance, and medicine [22, 48, 93, 134, 137, 147]. Of course, the meaning of such networks is domain-dependent and requires additional information.

In order to define a graph (or network) formally, we need to specify its set of vertices or nodes, V , and its set of edges, E . That means any vertex $i \in V$ is a node of the network. Similarly, any element $E_{ij} \in E$ is an edge of the network, which means that the vertices i and j are connected with each other. In case of an undirected graph, there is no direction for this connection, which means node i is connected with j but also node j is connected with i .

Graph: G Adjacency matrix: A

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Fig. 5.3 An example for a simple graph. Left: Visualization of an undirected graph G . Right: The adjacency matrix, A , of the graph on the left-hand side

Figure 5.3 shows an example of a simple undirected graph with $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{E_{12}, E_{23}, E_{34}, E_{14}, E_{35}, E_{36}\}$. It is clear from Fig. 5.3 that in an undirected network, the symbol E_{ij} is symmetric in its arguments, i.e., $E_{ij} = E_{ji}$, since the order of the nodes is not important.

Definition 5.2 An undirected network $G = (V, E)$ is defined by a vertex set V and an edge set $E \subseteq \binom{V}{2}$.

$E \subseteq \binom{V}{2}$ means that all edges of G belong to the set of subsets of vertices with two elements.

In order to encode a network mathematically, a matrix representation can be used. The *adjacency matrix*, A , is a squared matrix with $|V|$ number of rows and $|V|$ number of columns. The matrix elements, A_{ij} , of the adjacency matrix provide the connectivity of a network.

Definition 5.3 The adjacency matrix, A , of an undirected network G is defined by

$$A_{ij} = \begin{cases} 1 & \text{if } i \text{ is connected with } j \text{ in } G, \\ 0 & \text{otherwise,} \end{cases} \quad (5.1)$$

for $i, j \in V$.

The adjacency matrix, A , of the graph in Fig. 5.3, is shown on the right-hand side. Since this network is undirected, its adjacency matrix is symmetric, i.e., $A_{ij} = A_{ji}$ holds for all i and j .

Definition 5.4 (Network Data) The adjacency matrix, A , represents network data.

In Fig. 5.4, we show four real-world examples of networks. The first example shows a *chemical graph*. In this case, chemical elements correspond to nodes, and the bindings between the elements correspond to edges. The chemical structure shown is serotonin.

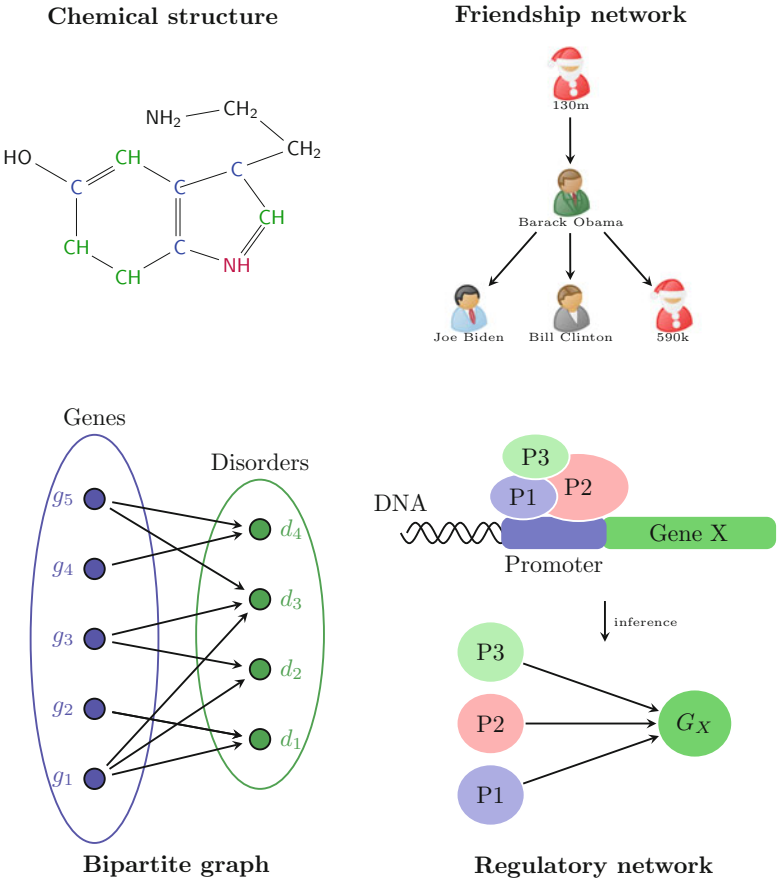


Fig. 5.4 Four examples for real-world networks. 1. Chemical structure of Serotonin. 2. Friendship network from Twitter. 3. Bipartite graph corresponding to the diseasesome. 4. Gene regulatory network providing information about the activation of genes in biological cells

The second example shows a *small* part of a friendship network or *acquaintance network*. Such networks can be constructed from social media data, e.g., Facebook or LinkedIn or, as is the case for Fig. 5.4, Twitter. A general problem with such networks and their visualization is that they can be very large. The shown subnetwork shows that Barack Obama follows Joe Biden and Bill Clinton. However, in addition, Obama follows 590,000 more Twitter users, and 130 million users follow Obama. From this information, it becomes clear that it is impossible to visualize all edges in the network, even when we focus only on Barack Obama. Despite this complexity, friendship networks can be easily constructed by collecting information about *who-follows-who*. That means such networks are constructed in an edge-by-edge manner.

The third network is a so-called *bipartite network*. In contrast to the networks discussed so far, a bipartite network consists of two types of nodes. In Fig. 5.4, the first type of nodes corresponds to genes and the second to disorders. For simplicity, let's call the first node type G and the second node type D . A bipartite network has the property that edges can only occur between nodes of different type, i.e.,

$$E_{ij} = 1 \text{ if } g_i \in G \text{ and } d_j \in D. \quad (5.2)$$

In order to distinguish such a network from a regular graph, we often write $G = (G, D, E)$, and the gene-disorder bipartite network was called *diseasome* [196]. Examples of such gene-disorder pairs are BRAC1 for breast cancer, KRAS pancreatic cancer, BRAC1 ovarian cancer, or C9orf72 ALS (amyotrophic lateral sclerosis). The *diseasome* is another example of a network that is constructed edge by edge [144], where the information about individual edges is obtained from databases, e.g., the Online Mendelian Inheritance in Man (OMIM) [370] providing information about thousands of individual experiments.

The fourth network in Fig. 5.4 is a regulatory network or *gene regulatory network (GRN)*. A GRN describes molecular activities within biological cells about the regulation of genes. Specifically, in order for genes to become activated, a certain number of proteins (called transcription factors) need to bind to the DNA to a promoter region. This is illustrated in Fig. 5.4 where the proteins P1 to P3 activate the gene X. This sounds similar to a friendship network; however, there is one crucial difference. While for a friendship network the “friends” are directly observable, e.g., from Twitter via “Followers” and “Following,” this information is only indirectly available for a GRN. Here, indirectly means this information needs to be inferred via statistical methods. One of these methods is BC3Net (bagging conservative causal core network) [105], which is based on an ensemble method (bagging) applied to statistical hypothesis tests.

The above examples show that the construction of a network can be simple, following deterministic rules, as for acquaintance networks of Twitter friends, or difficult requiring the statistical inference of connections via hypothesis testing, as for gene regulatory networks or financial networks [7, 323]. Hence, the generation of network data can be involved.

From a practical point, for large networks with many nodes, i.e., $|V| \gg 10000$, it can be preferable, in order to reduce the storage requirement on a computer, to use an edge list E instead of the adjacency matrix A . For a given adjacency matrix A , Listing 5.2.2 shows an example to obtain the edge list from A for the network in Fig. 5.3.

Listing 5.1: Edge list of an undirected network

```

A <- matrix(c(0,1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1,
             1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0), 6,
            6)
rownames(A) <- 1:6
colnames(A) <- 1:6

library(igraph)
g <- graph.adjacency(A, mode="undirected")
get.edgelist(g)
# output
      [,1] [,2]
[1,]  "1"  "2"
[2,]  "1"  "4"
[3,]  "2"  "3"
[4,]  "3"  "4"
[5,]  "3"  "5"
[6,]  "3"  "6"

```

The reduction in storage requirements comes from the fact that an edge list stores only information about existing edges and ignores all non-edges (non-existing edges).

5.2.3 Text Data

The third data type is very interesting because text data are not numbers. Instead, text data are symbol sequences that form a natural language, where each language has its own characteristic symbols (given by the alphabet). Hence, when analyzing text data, the first task is to convert symbol sequences into a representation that is amenable for machine learning methods. This requires a form of mathematical representation corresponding to, e.g., vectors of numbers.

One aspect that makes the analysis of text data a difficult task is that the conversion between symbol sequences and a (mathematical) representation is not unique. Instead, many different approaches have been introduced and used over the past decades. Here, we discuss five widely used text data representations: part-of-speech (POS), one-hot document (OHD), one-hot encoding (OHE), term frequency-inverse document frequency (TF-IDF), and word embedding (WE).

The idea of *part-of-speech (POS)* tagging is to assign each word (or lexical item) in a sentence to a grammatical category. Examples for such categories are noun (NOUN), determiner (DET), or auxiliary (AUX). For an example, see Fig. 5.5. Put simply, words in the same POS category show a similar syntactic behavior in a sentence by playing a similar role for the grammatical structure. For linguistics, POS tagging allows a systematic analysis of the grammatical structure of texts. For instance, in Fig. 5.5, we show the distribution of 49 categories as found in the novel *Moby-Dick* by Herman Melville published in 1851.

define TF-IDF, one needs two components, term frequencies and inverse document frequencies. The term frequencies are defined for each word (term), w_i , in a document, d_j , by

$$\text{TF}(w_i, d_j) = \frac{\#\{w_i | w_i \in d_j\}}{\text{len}(d_j)} \quad (5.3)$$

where $\text{len}(d_j)$ is the length of document d_j (total number of words—not the unique number of words) and $\#\{w_i | w_i \in d_j\}$ is the word frequency (how often does w_i appear in d_j).

The document frequencies give information about the fraction of documents containing a certain word, w_i , in all documents, and it is defined by

$$\text{DF} \sim \frac{\#\{d | w_i \in d\}}{|D|}, \quad (5.4)$$

where $|D|$ is the total number of documents, i.e., $D = \{d_1, \dots, d_{|D|}\}$. The inverse document frequencies is just the inverse of the above expression; however, this inverse is usually scaled using the logarithm as follows:

$$\text{IDF}(w_i) = \log\left(\frac{|D|}{\#\{d | w_i \in d\}}\right). \quad (5.5)$$

The resulting TF-IDF for each word and document is then given by

$$\text{TF-IDF}(w_i, d_j) = \text{TF}(w_i, d_j) \cdot \text{IDF}(w_i). \quad (5.6)$$

As an example, we consider the following (simple) documents, each consisting of one sentence:

$$d_1 : \text{one plus one is two;} \quad (5.7)$$

$$d_2 : \text{one times one is one.} \quad (5.8)$$

For this we obtain

$$\text{TF}(\text{'one'}, d_1) = 2/5 \quad (5.9)$$

$$\text{TF}(\text{'plus'}, d_1) = 1/5 \quad (5.10)$$

$$\text{TF}(\text{'is'}, d_1) = 1/5 \quad (5.11)$$

$$\text{TF}(\text{'two'}, d_1) = 1/5 \quad (5.12)$$

$$\text{TF}(\text{'one'}, d_2) = 3/5 \quad (5.13)$$

$$\text{TF}(\text{'times'}, d_2) = 1/5 \quad (5.14)$$

$$\text{TF}(\text{'is'}, d_2) = 1/5 \quad (5.15)$$

and

$$\text{IDF}('one') = \log(2/2) = 0 \quad (5.16)$$

$$\text{IDF}('plus') = \log(2/1) = 0.69 \quad (5.17)$$

$$\text{IDF}('is') = \log(2/2) = 0 \quad (5.18)$$

$$\text{IDF}('two') = \log(2/1) = 0.69 \quad (5.19)$$

$$\text{IDF}('times', d_2) = \log(2/1) = 0.69. \quad (5.20)$$

The resulting term frequency-inverse document frequencies for this example are shown in Listing 5.2, which also provides steps to calculate these values using R.

Listing 5.2: Term frequency-inverse document frequency (TF-IDF)

```
library(dplyr)
library(tidytext)

input.text <- c('one plus one is two', 'one times one is one')
df <- data.frame(text = input.text, class = c("D1", "D2"))

dat.text <- df %>%
  mutate(text = as.character(text)) %>%
  unnest_tokens(output = word, input = text) %>%
  count(class, word, sort = TRUE)

dat.text <- dat.text %>%
  bind_tf_idf(term = word, document = class, n)
dat.text
  class word n   tf      idf   tf_idf
1    D2  one 3  0.6 0.0000000 0.0000000
2    D1  one 2  0.4 0.0000000 0.0000000
3    D1  is  1  0.2 0.0000000 0.0000000
4    D1 plus 1  0.2 0.6931472 0.1386294
5    D1 two  1  0.2 0.6931472 0.1386294
6    D2  is  1  0.2 0.0000000 0.0000000
7    D2 times 1  0.2 0.6931472 0.1386294
```

Overall, a word that is representative of a document, e.g., because it appears often in this document but not in others, receives a high TF-IDF value, whereas a word that appears often in all documents receives a low TF-IDF value due to its lack of being representative for any document.

By using the TF-IDF values, one can now extend the definition of one-hot-document by substituting the “1”s with the values of the term frequency-inverse document frequencies while the zeros remain unchanged. This leads to real valued vectors that can be used to compare documents. For our example, this gives:

$$v = ('one', 'two', 'plus', 'times', 'is') \quad (5.21)$$

$$v_1 = (0, 0.13, 0.13, 0, 0) \quad (5.22)$$

$$v_2 = (0, 0, 0, 0.13, 0) \quad (5.23)$$

All above measures are defined for a bag-of-words. A natural extension of a bag-of-words approach is a bag-of-n-grams. In general, an n-gram is a contiguous sequence of n items from a given text. These items can be, for instance, letters, syllables, or words. Put simply, an n-gram is any sequence of n tokens. Using n-grams, e.g., of words, one can again determine the TF-IDF. This allows to study more complex entities, rather than single words, and their meaning for a given text.

The last text representation we will discuss is a *word embedding (WE)* model. In contrast with the previous representations, a WE model learns a real-valued vector representation of a word from text data. The learning considers the context of a word, at least to some extent, and hence, it is different to approaches based on bag-of-words and even bag-of-n-grams.

Technically, there are a number of different methods, frequently utilizing neural networks, for learning such a vector representation of a word, but word2vec was the first one introduced in 2013. The most popular architecture of word2vec is based on continuous bag-of-words (CBOW) [332]. The CBOW model predicts the current word from a window of surrounding context words, where the order of the context words has no influence on the prediction. The dimension of a vector is a parameter, and it is not determined by the size of the vocabulary. The learning vector representations of words allows to construct vectors for entities beyond words, e.g., phrases or entire documents, to study their similarity.

Definition 5.5 (Text Data) Text data are symbol sequences that do not correspond to numbers. In order to analyze text data mathematically, there are different text representations (for instance, POS, OHD, OHE, TF-IDF, WE), which have different interpretations and applications.

We conclude by remarking that the field dedicated to analyze text data is natural language processing (NLP), and recent years have seen many advances by utilizing methods from deep learning (see Chap. 14).

5.2.4 Time-to-Event Data

The next data type is called time-to-event data. Such data have yet another form compared to all other data types, which is very characteristic. In its simplest form, time-to-event data can be represented as a triplet given by:

$$\text{time-to-event data: (ID, } t: \text{time duration, } c: \text{censoring)} \quad (5.24)$$

Here, ID corresponds to the identification number of, e.g., a patient, time duration t is a time interval, and censoring c is a binary label, i.e., $c \in \{1, 2\}$ with 1 corresponding to censoring occurred and 2 indicating the event occurred. A crucial difference with many other data types is that the above information, i.e., t and

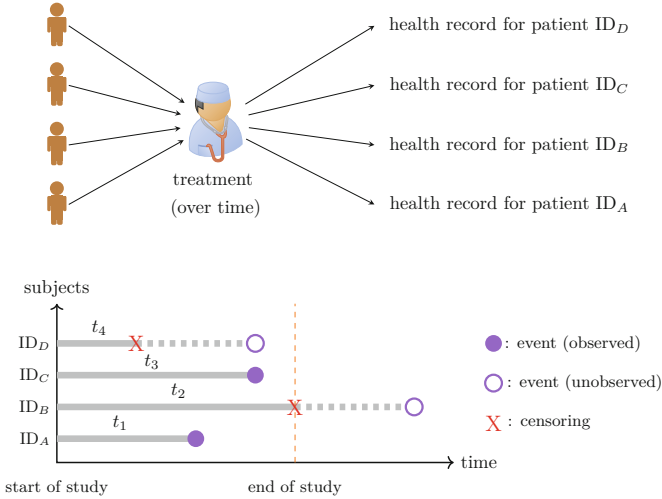


Fig. 5.6 Generation of time-to-event data in a medical context. Top: Patients receive treatment for a particular disease. This treatment is extended over a certain time duration. Bottom: A summary of time-to-event data extracted from health records of patients. Unobserved events lead to a complication requiring information about this which is called censoring

c , cannot be directly measured by an experiment. Instead, this information needs to be extracted from a certain process, and this process is domain or application dependent.

In order to provide a motivating example, let’s discuss time-to-event data in a medical context. In Fig. 5.6 (top), we show patients participating in a study and receiving a treatment at a hospital. Each treatment, e.g., a surgery, will lead to a patient-specific health record, which contains all the medical information about a patient. It is clear that not all patients receive the surgery at the same day at the hospital, but this occurs over a longer period, e.g., months or years, depending on the duration of the study. From the patient-specific health records, one can then extract the above described time-to-event data. However, this requires the definition of an “event.” In a medical context, an example of an event is “death.” Other examples are “relapse” or “exhibiting symptoms.” This allows to obtain t , e.g., corresponding to the time from surgery to the time of death.

The meaning of the third entity of time-to-event data, censoring, is described in Fig. 5.6 (bottom). Due to the fact that the collection of the health records of patients occurs during a period of time, one needs to distinguish between three different cases. For the first case, represented by patient ID_A and ID_C , the event “death” occurs either in hospital or outside the hospital, but the hospital is informed about the death of the patient, and this event happens within the time period of the study. In both cases, the health record of the patient can be updated by including this information. For the second case, represented by patient ID_B , the event “death” occurs at some time; however, the hospital is not informed, or the study ended

already. Hence, from the health record, only information about the last doctor visit of the patient is available but nothing after the last visit. The time of the last visit is called censored, and it indicates a patient lived at least till this day. For the third case, represented by patient ID_D , the event “death” occurs within the study; however, the patient decided before this to drop out of the study. Hence, this event despite occurring within the time frame of the study is not observed. This time is therefore labeled as censored.

Overall, for the example shown in Fig. 5.6, this gives the following time-to-event data for the four patients:

$$(ID_A, t_1, c_1 = 2); \quad (5.25)$$

$$(ID_B, t_2, c_2 = 1); \quad (5.26)$$

$$(ID_C, t_3, c_3 = 2); \quad (5.27)$$

$$(ID_D, t_4, c_4 = 1). \quad (5.28)$$

Despite their simple appearance, the analysis of these data is far from simple. In Chap. 16, we will see the analysis methods developed for such data, which are summarized under the term survival analysis.

For the description of time-to-event data, an “event” plays a pivotal role. Depending on the application, there are various definitions of “event,” and this makes survival analysis widely applicable. In an industrial context, an event could be “malfunctioning” or “failure” of a component/device of a machine, while in marketing, an event could be “purchase” of a product or “churn” of a customer.

5.2.5 Business Data

The last data type we are discussing in this chapter is business data. Business data are similar to expression data from genomic experiments, which assume a matrix form, but business data come in the form of a table. Although a matrix and a table may look similar at first glance, there is a crucial difference between both entities as discussed below. Examples of business data are customer, transaction, financial, product, or market research data. Hence, one encounters such data frequently in management, economy, or marketing.

Let’s look at an example. The dataset `UniversalBank.csv` contains data about 5000 customers of the Universal Bank. The data include customer demographic information (age, income, etc.), the customer’s relationship with the bank (mortgage, securities account, etc.), and the customer’s response to the last personal loan campaign (Personal Loan). In R, tables in csv format, e.g., used by Excel, can be easily loaded as the example in Listing 5.3 shows.

Listing 5.3: Loading business data as csv file from Excel

```
my.data <- read.csv("UniversalBank.csv")
head(my.data)
# output
  ID Age Experience Income ZIP.Code Family CCAvg Education
1  1  25           1    49   91107      4   1.6          1
2  2  45          19    34   90089      3   1.5          1
3  3  39          15    11   94720      1   1.0          1
4  4  35           9   100   94112      1   2.7          2
5  5  35           8    45   91330      4   1.0          2
6  6  37          13    29   92121      4   0.4          2
155
  PersonalLoan SecuritiesAccount CDAccount Online CreditCard
1             0                 1          0         0         0
2             0                 1          0         0         0
3             0                 0          0         0         0
4             0                 0          0         0         0
5             0                 0          0         0         1
6             0                 0          0         1         0
```

The shown table contains 14 columns, corresponding to 14 features. These features have the following meaning: ID, Customer ID; Age, Customer's age in completed year; Experience, number of years of professional experience; Income, annual income of the customer (1000); ZIP code, Home address ZIP Code; Family, Family size of the customer; CCAvg, Average monthly credit card spending (1, 000); Education (Education level), 1 for undergrad, 2 for Graduate, 3 for Advance/Professional; Mortgage, Value of house mortgage if any (1, 000); Personal loan (Did this customer accept the personal loan offered in the last campaign?), 1 for yes, 0 for no; Securities Acct (Does the customer have a securities account with the bank?), 1 for yes, 0 for no; CD Account (Does the customer have a certificate of deposit—CD—account with the bank?), 1 for yes, 0 for no; Online (Does the customer use internet bank facilities?), 1 for yes, 0 for no; Credit Card (Does the customer use a credit card issued by the Bank?): 1 for yes, 0 for no.

A potential problem with business data is that their features can be of a different *level of measurement* (see Sect. 10.6.4 for a detailed discussion of level (or scale) of measurement). Put simply, this means that there are different types of features as can be seen by comparing, e.g., the feature “Income” and “ZIP.Code.” While the former can assume numbers, the latter is merely a label. This means the number of a ZIP code could be replaced by any character string without losing information. Due to the label type of the ZIP code, the summation of two ZIP codes does not make any sense. Hence, despite the fact that a ZIP code is a number, its meaning indicates it is a label.

This is the main difference between a table and a matrix. While a table can contain different types of features, e.g., numbers, labels, character strings, etc., a

matrix contains only one type of features. This is a crucial difference between the gene expression data discussed earlier and the business data.

When analyzing data with mixed feature types, this needs to be taken into consideration because one cannot just treat a label as a number since both variables convey different information. For this reason, in statistics, such variables are called *nominal* or categorical. In general, when selecting a method, the level of measurement of the features needs to be considered, and not every method can be used indifferently. In Sect. 10.6.4, we will see that there are in total four different levels of measurement one needs to distinguish, and three of those can be found in the data from the Universal Bank (nominal, ZIP Code; ordinal, Education; and ratio, Income).

Definition 5.6 (Business Data) Business data are represented by tables. The entries in a table can correspond to different levels of measurement (mixed type of information).

Other types of data that usually come in the form of a table are from economy, finance, politics, psychology, or the social science. Since data from those domains contain usually a mixture of levels of measurement, the same arguments for business data are applicable to them.

5.3 Summary

The lessons learned from this chapter are the following: First, one should realize that data generating processes can be very complicated and they are usually highly domain specific. That means data generated in biology are very different from data coming from economy. However, to conduct a sensible analysis, one needs always sufficient background information and insights about the application domain, which generated the data. For the many fields including those discussed above, acquiring this knowledge is nontrivial and requires a substantial effort. Second, the resulting data types can also be quite different for different application domains. This effects also the selection of prediction models, which can be used for the analysis since different methods have different data requirements. Third, data may not even come in a numerical form, as we have seen for text data studied in the field natural language processing, where a mapping from “text to numbers” is required before any method can be used for an analysis.

Learning Outcome 5: Data

Data come in many different forms (data heterogeneity), which cannot be mapped into each other. This zoo of data types requires diverse families of prediction methods.

In the following chapters, where we focus on methods, the discussion about the data will be usually short. However, this is merely for practical reasons requiring us to limit the length of our discussions. For real-world data analysis projects, the analyst needs always to make sure that the data are addressed properly and with sufficient detail.

Part II

Core Methods

Chapter 6

Statistical Inference



The basic idea of statistics is to use a data sample to draw conclusions about the underlying population from which the data have been drawn. Since, in reality, a sample of data has always a finite size, our conclusions about the population are always uncertain to a certain degree. The goal of statistics is to quantify the amount of uncertainty around the conclusions we make based on a sample of data. In general, *statistical inference* is the (systematic) process of making predictions about a population, using data drawn from that population.

Figure 6.1 provides a visualization of the concept of statistical inference which connects properties in a theoretical world (upper part) described by probability theory with properties in reality (lower part) described by statistics. Assuming a population is given, such properties can be deduced probabilistically. To achieve this, probability theory, i.e., mathematics, can be used to study a population. On the other hand, the lower part, in Fig. 6.1, corresponds to a data sample and the properties that can be inferred thereof. Since a sample is always finite, such conclusions are only estimates of the population properties. Overall, the upper part belongs to a theoretical world in which the laws of mathematics hold by means of probability theory, whereas the lower part represents the reality, which requires methods for inferring conclusions by means of statistics or data science. Both worlds are separated from each other, and information can only be exchanged via a data sample which is not only small compared to the population itself but is usually corrupted by measurement errors and noise. That means even the small part of the population we can observe is potentially corrupted. From this description, the difficulty of the problem we are facing should become clear.

The above description is true for all fields dealing with the analysis of data, including machine learning, artificial intelligence, or pattern recognition, but statistics was the first field that articulated this explicitly. Hence, the term statistical inference is usually connected to the field of statistics, although all data science-related fields are facing the same issue.

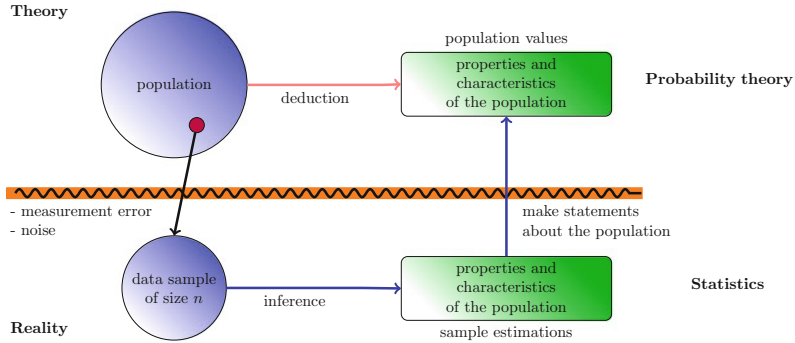


Fig. 6.1 Conceptual framework of statistical inference which connects properties in a theoretical world (upper part) with properties in reality (lower part)

We start this chapter by discussing *exploratory data analysis (EDA)* [470], including descriptive statistics and properties of sample estimators. Both topics provide valuable means for any data science project because methods thereof are universally applicable. Then, we discuss the two main approaches for parameter estimation, namely, Bayesian inference and maximum likelihood estimation (MLE), and their conceptual differences. Finally, we discuss the expectation-maximization (EM) algorithm as a practical way for finding the (local) maximum of a likelihood function or the maximum posteriori (maximum of the posterior distribution), which corresponds to the parameter estimate of a statistical model, using an iterative approach.

6.1 Exploratory Data Analysis and Descriptive Statistics

The first step of any statistical investigation consists of preprocessing the data, assessing its quality, analyzing its structure, and calculating the associated summary statistics.

6.1.1 Data Structure

Data can be defined as a collection of facts derived from experiments or observations. Aspects of the data, to be taken into consideration for a statistical investigation, include:

- *Sample size:* This is the total number of data points collected; a very small sample size is likely to lead to unreliable statistical results.
- *Number of variables:* For a large number of variables, one needs to investigate whether all the variables are necessary.

- *Type of variables*: Variables can be categorized into two main classes:
 - *Qualitative or categorical variable*
 - *Nominal*, e.g., patient gender (male, female)
 - *Ordinal*, e.g., response to a treatment (poor, mild, good)
 - *Quantitative variable*
 - *Continuous*, e.g., patient age (0–120 years)
 - *Discrete*, e.g., number of admissions to care (1, 2, 3, etc.)
- *Independence of the observations*: We need to investigate whether the observations from the variable of interest are independent from each other.
- *Distributions of the variables*: We need to investigate whether the variables are following classical distributions, e.g., the normal distribution.

We would like to remark that variables are sometimes also called features, especially in the machine learning community.

6.1.2 Data Preprocessing

Data quality is essential for an effective use of statistical inference techniques, and the data preprocessing step aims to assess the quality of the data.

There are four main stages in the preprocessing of the data:

- Data consolidation: data collection, data selection, and data integration;
- Data cleaning: data auditing, data de-cleansing, imputation of missing values or outliers, elimination of data inconsistency, and data quality validation;
- Data transformation: data discretization/aggregation, data normalization, and derivation of new attributes/variables;
- Data reduction: variable reduction, sample reduction, and data balancing.

6.1.3 Summary Statistics and Presentation of Information

Descriptive statistics can be defined as a summary statistics of the whole data set as well as the most important sub-groups. The statistics to be calculated depend on the type and number of variables under consideration, and they generally consist of:

- *Measures of location*: They describe the central tendency of the data.
- *Measures of scale*: They describe the spread of the data, i.e., the departure from the central tendency.
- *Measures of shapes*: They describe the form/shape of the distribution of the data.

To be more informative, the summary statistics need to be presented using the most suitable format, which can include:

- *Tables*: They are used for presenting data itself, summary statistics (e.g., contingency tables), and final results.
- *Graphs*: They are used for displaying broad qualitative information such as the shape of a distribution, the form of a relationship between a pair of variables, the presence of outliers, etc.; the most commonly used graphs for an exploratory data analysis include histograms, bar charts, boxplots, scatter plot, and pie charts.

Due to their importance, in the following, we discuss measures of location, measures of scale, and measures of shape in more detail.

6.1.4 Measures of Location

A measure of location attempts to describe a *typical individual sample* through a number. That means the possible complex characteristics of a population are summarized by a single number that should reflect the *centrality* of the population.

6.1.4.1 Sample Mean

The sample mean, also called the arithmetic sample mean or the average, denoted $\bar{x}$, is defined as the sum of the observations of the sample divided by the sample size. Therefore, the sample mean is estimated as follows:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (6.1)$$

One of the problems associated with the sample mean is that a small number of outliers can have a huge impact on its value. Such outliers could stem, e.g., from erroneous measurements. Importantly, it is possible that even just one measurement error could have a determining effect on the sample mean.

Let's illustrate this problem using a data sample with an outlier, which is 45, and a similar data sample without this outlier.

											Sample mean
Data sample with outlier	3	5	2	3	45	4	2	3	5	4	7.6
Data sample without outlier	3	5	2	3	4	4	2	3	5	4	3.5

The presence of the outlier has doubled the value of the sample mean without outlier, which is a significant error in the description of the sample.

6.1.4.2 Trimmed Sample Mean

The trimmed mean is the sample mean, which results from trimming a certain percentage from both ends of the ordered original data sample. Let's calculate the 10% trimmed mean of the following data sample containing an outlier, which is 45: 3, 5, 2, 3, **45**, 4, 2, 3, 5, 4.

											Sample mean
Original data sample	3	5	2	3	45	4	2	3	5	4	7.6
10% trimmed data sample		2	3	3	3	4	4	5	5		3.6

Therefore, the trimmed mean can be used to address one of the shortcomings of the sample mean, namely, its sensitivity to outliers. However, the threshold of the data to be trimmed needs to be chosen carefully to take advantage of this potential benefit of the trimmed mean without throwing away some valuable observations within the data.

6.1.4.3 Sample Median

The median of a data sample, denoted m_x , is defined as the middle point of the ordered observations from the sample. It is estimated by, first, ordering the data from the smallest to the largest value and then counting upward for half the observations. Let n denote the sample size of the data. Then, the value of the sample median depends on whether the number n is even or odd. The sample median is given by

$$\begin{cases} \text{the } \left(\frac{n+1}{2}\right)\text{th ordered value} & \text{if } n \text{ is odd,} \\ \text{the average of } \left(\frac{n}{2}\right)\text{th and } \left(\frac{n}{2} + 1\right)\text{th ordered values} & \text{if } n \text{ is even.} \end{cases}$$

Therefore,

$$m_x = x_{(n+1)/2} \quad \text{if } n \text{ odd,} \quad (6.2)$$

$$m_x = \frac{1}{2}(x_{n/2} + x_{n/2+1}) \quad \text{if } n \text{ even,} \quad (6.3)$$

where x_i , $i = 1, \dots, n$, are the ordered values of the observations from the sample data.

For the following data sample

$$(180, 175, 191, 184, 178, 188), \quad (6.4)$$

the ordered observations are

$$(175, 178, 180, 184, 188, 191).$$

(6.5)

Since the sample size is 6 (even), the median is the average of 180 and 184, which is 182.

If we have one more observation value, say 189, then the ordered observations are

$$(175, 178, 180, 184, 188, 189, 191)$$

(6.6)

and the median is given by 184, since the sample size is odd.

In contrast with the sample mean, the sample median is quasi-insensitive to outliers. In fact, the median is affected by at most two values in the sample data, and these are half-way the ordered observations data. Let’s illustrate this desirable property of the sample median using a data sample with an outlier, which is 45, and a similar data sample without this outlier.

											Sample median
Data sample with outlier	3	5	2	3	45	4	2	3	5	4	
Ordered observations	2	2	3	3	3	4	4	5	5	45	3.5
Data sample without outlier	3	5	2	3	4	4	2	3	5	4	
Ordered observations	2	2	3	3	3	4	4	4	5	5	3.5

The presence of the outlier didn’t affect the value of the sample median, which is the same for both data samples.

6.1.4.4 Quartile

A quartile is any of the three values, which divide an ordered data sample into four equal parts, so that each part forms a quarter of the sample. Therefore, the second quartile is nothing but the sample median.

For the following data sample

$$(180, 175, 191, 184, 178, 188, 189, 183, 197, 186, 172, 169, 181, 177, 170, 172),$$

the ordered observations are

$$(169, 170, 172, 172 \mid 175, 177, 178, 180 \mid 181, 183, 184, 186 \mid 188, 189, 191, 197).$$

The first, second, and third quartiles for this data sample are given below:

- 1st quartile = $\frac{172 + 175}{2} = 173.5$

- 2nd quartile (sample median) = $\frac{180 + 181}{2} = 180.5$
- 3rd quartile = $\frac{186 + 188}{2} = 187$.

6.1.4.5 Percentile

A percentile is the data value that is greater than or equal to a certain percentage of the observations in a data sample.

For the following sample

(180, 175, 191, 184, 178, 188, 189, 183, 197, 186, 172, 169, 181, 177, 170, 172),

the ordered data are

(169, 170, [172, 172], 175, 177, 178, 180, 181, 183, 184, 186, 188, 189, 191, 197).

The 20th percentile is the value that is greater or equal to 20% of the observations. Since the sample size is 16, then the rank of the 20th percentile is given by $20\% \times 16 = 3.2 \approx 3$. Therefore, the 20th percentile is 172.

6.1.4.6 Mode

The mode of a data sample is the observation that occurs most often in the sample, in other words, the observation that is more likely to be sampled.

As an example, let's consider the following data sample:

$$(1, 5, 2, 3, 4, 4, 2, 3, 5, 4) \quad (6.7)$$

for which the ordered sample is

$$(1, 2, 2, 3, 3, 4, 4, 4, 5, 5). \quad (6.8)$$

For this data sample, the mode is 4, since it is the most observed value.

6.1.4.7 Proportion

While the sample mean, sample median, quartile, and percentile are meaningful and can be readily calculated for quantitative data, the most representative measure of location for categorical data is the proportion or percentage of each of the category associated with the observations in the data sample.

For instance, “prevalence” is the proportion of patients with the disease of interest (see Chap. 3), whereas “fatality” is the proportion of people who died due to the event of interest, etc.

6.1.5 Measures of Scale

A measure of scale (or dispersion) attempts to describe the extent to which values in a sample differ from some measures of location, e.g., the sample mean, of the same sample.

6.1.5.1 Sample Variance

The sample variance describes the spread of the observations of the data sample around the sample mean. It is given by the average of the squares of the difference between the sample mean and each individual observation in the data sample. If $\bar{x}$ denotes the mean of the sample $x_1, \dots, x_n$, the sample variance, denoted s^2 , is given by

$$s^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2. \quad (6.9)$$

For statistical reasons, the following formulation of the variance is preferred due to its theoretical properties, namely, it is an unbiased estimation of the sample variance.

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2. \quad (6.10)$$

There are two other important measures of scale, directly derived from the sample variance, namely, the *sample standard deviation* and the *sample standard error of the mean*, which are defined as follows:

- The sample standard deviation denoted s is given by square root of the variance, i.e., $s = \sqrt{s^2}$.
- The standard error of the mean of a data sample, denoted SE , is obtained by dividing the sample standard deviation by the square root of the sample size, i.e., $SE = \frac{s}{\sqrt{n}}$. A discussion on the *standard error* is provided in Chap. 4.

As an example, let's consider the data sample (1, 5, 2, 3, 4, 4, 2, 3, 5, 4). For this data sample, the sample variance, the sample standard deviation, and the standard error are 1.8, 1.3, and 0.4, respectively.

Before we continue, we would like to highlight the difference between a sample variance, estimated using Eq. (6.10), and the population variance given by Eq. (6.9).

6.1.5.2 Range

The range of a data sample is given by the difference between the largest and the smallest observation in the sample. For instance, the range of the data sample (1, 5, 2, 3, 4, 4, 2, 3, 5, 4) is given by $5 - 1 = 4$.

6.1.5.3 Interquartile Range

The interquartile range, denoted IQR, for a data sample is given by the difference between the 3rd and the 1st quartiles.

For the following data sample

(180, 175, 191, 184, 178, 188, 189, 183, 197, 186, 172, 169, 181, 177, 170, 172),

the 1st and 3rd quartiles are 173.5 and 187, respectively. Hence, the corresponding interquartile range is 13.5.

6.1.6 Measures of Shape

A measure of shape attempts to describe some graphical properties of the distribution of the data sample through some numerical values. The main properties of interest include the symmetry of the distribution, its tendency to skew, its uniformity, and its number of modes (i.e., whether it is unimodal, bimodal, or multimodal).

6.1.6.1 Skewness

The skewness defines the asymmetry of a distribution, i.e., the degree to which a distribution is distorted either to the left or to the right.

If the values of the sample mean and sample median fall to the right of the mode, then the distribution is said to be skewed positively. Therefore, with positive skew, we have sample mean $>$ sample median $>$ sample mode.

If the values of the sample mean and sample median fall to the left of the mode, then the distribution is said to be skewed negatively. Therefore, with negative skew, we have sample mean $<$ sample median $<$ sample mode.

A measure of skewness is the Fisher-Pearson coefficient of skewness [379]. Let $x_1, x_2, \dots, x_n$ denote a data sample of size n . Then, the Fisher-Pearson coefficient of skewness, denoted γ , is given by

$$\gamma = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^3}{s^3}, \quad (6.11)$$

where $\bar{x}$ and s denote the sample mean and the sample standard deviation, respectively.

The following adjusted variant of the Fisher-Pearson coefficient of skewness (6.11) is also commonly used:

$$\gamma_{adj} = \frac{\sqrt{n(n-1)}}{n-2} \gamma. \quad (6.12)$$

Other measures of skewness include:

- The Pearson coefficients of skewness, denoted γ_{P1} and γ_{P2} , given by

$$\gamma_{P1} = \frac{\bar{x} - \bar{m}}{s}, \quad (6.13)$$

$$\gamma_{P2} = \frac{3(\bar{x} - m_x)}{s}, \quad (6.14)$$

where $\bar{x}$, $\bar{m}$, s , and m_x denote the sample mean, the sample mode, the sample standard deviation, and the sample median, respectively.

- The Galton measure of skewness also known as Bowley's measure of skewness or Yule's coefficient, denoted γ_G , given by

$$\gamma_G = \frac{Q_1 + Q_3 - 2Q_2}{Q_3 - Q_1}, \quad (6.15)$$

where Q_1 , Q_2 , and Q_3 denote the first, second, and third quartiles, respectively, of the data sample.

Remark 6.1 The skewness coefficient of a symmetrical distribution (e.g., the normal distribution) is near zero. Negative values for the coefficient of skewness indicate that the data are left skewed, whereas positive values indicate that the data are right skewed.

6.1.6.2 Kurtosis

The kurtosis is a measure of the peakedness for a symmetrical distribution. Let $x_1, x_2, \dots, x_n$ denote a data sample of size n . Then, the kurtosis of the sample, denoted η , is given by

$$\eta = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^4}{s^4}, \quad (6.16)$$

where $\bar{x}$ and s denote the sample mean and the sample standard deviation, respectively.

The following adjusted variant of the kurtosis (6.16) is also commonly used:

$$\eta_{adj} = \eta - 3. \quad (6.17)$$

Using the definition of kurtosis provided by Eq. (6.16), the kurtosis of the normal distribution has an expected value of 3. Therefore, using the definition of kurtosis provided by Eq. (6.17), the kurtosis of the normal distribution is approximately 0.

Remark 6.2 The skewness coefficient and the kurtosis can be used to quantify the extent to which a distribution differs from the normal distribution.

6.1.7 Data Transformation

Some data could benefit from transformations of the observations to get the sample “fit” for statistical analyses. For quantitative data, the transformation can be used for:

- Reducing the skewness:
 - To reduce right skewness, we can take square roots or logarithms or reciprocals of the observations in the sample.
 - To reduce left skewness, we can take squares or cubes or higher powers of the observations in the sample.
- Achieving approximate normality: the observations of the sample can be standardized by calculating the z -score for each observation, x_i , which is given by

$$z_i = \frac{(x_i - \bar{x})}{s}, \quad (6.18)$$

where $\bar{x}$ and s denote the mean and standard deviation of the sample.

- Stabilizing the variance: the form of the transformation is dictated by the dependence of the variance on the other sample characteristics, e.g., the sample mean.

Categorical variables can also benefit from special transformations. The most common transformation is made on the proportion or percentage values of the categories. This is referred to as the *logit* (or logistic) transformation, and it is defined as follows:

- $\text{logit}(p) = \log(p/(1 - p))$ for proportions,
- $\text{logit}(p) = \log(p/(100 - p))$ for percentages,

where p is a proportion or a percentage.

6.1.8 Example: Summary of Data and EDA

In the following, we provide a numerical example using data of the recurrence time to infection for kidney patients who are using a portable dialysis equipment. The time is measured from the point of the insertion of the catheter. The data set is available from the package “*survival*” available in R, and for the following analysis, we use only data from 18 patients experiencing a recurrent infection (corresponding to uncensored data; see Chap. 5).

In Table 6.1, we provide summary statistics for measures of location and measures of scale for the kidney patients’ data. The first part of the table presents the measures of location and the second half the measures of scale. Here, we assumed that the data for the 18 patients are in a vector called “dat.”

Listing 6.1 gives details of the different measures. It is important to note that there is no function for obtaining the mode in the base package of R. However, by estimating the density of the recurrence times, this information can be obtained in only two lines of code.

Listing 6.1: Summary statistics and EDA for the kidney data, see Fig. 6.2

```
library("survival")

data(kidney)
ind <- which(kidney[,3]==0) # uncensored events
dat <- kidney[ind,2]

mean(dat)
median(dat)
mean(dat, trim=0.1)
d <- density(dat)
d$x[which.max(d$y)] # mode

var(dat)
range(dat)
IQR(dat)

hist(dat, breaks=20, col="lavender")
plot(density(dat, from=0))
boxplot(dat, lwd=1, col="lavender")
plot(ecdf(dat))
```

In addition to the summary statistics, Listing 6.1 gives also four examples for a graphical summary of data using a histogram, density plot, boxplot, and empirical cumulative distribution function (ECDF). The results of these visualizations are depicted in Fig. 6.2. It is worth highlighting that the data underlying each of these four different graphs is identical. From the different shapes of the graphs, one can nicely see the power of the visualization of the data since each graph emphasizes a different aspect of the “distribution” of the data.

Table 6.1 Summary statistics for measures of location and measures of scale for the kidney patient data

Measure type	Name	R code	Value for the kidney data
Location	Sample mean	<code>mean(dat)</code>	46.38
	Sample median	<code>median(dat)</code>	23
	Trimmed mean (10%)	<code>mean(dat, trim=0.1)</code>	42
	Mode	<code>d <- density(dat)</code> <code>d\$x[which.max(d\$y)]</code>	14.58
Scale	Variance	<code>var(dat)</code>	2673.6
	Range	<code>range(dat)</code>	4159
	IQR	<code>IQR(dat)</code>	58

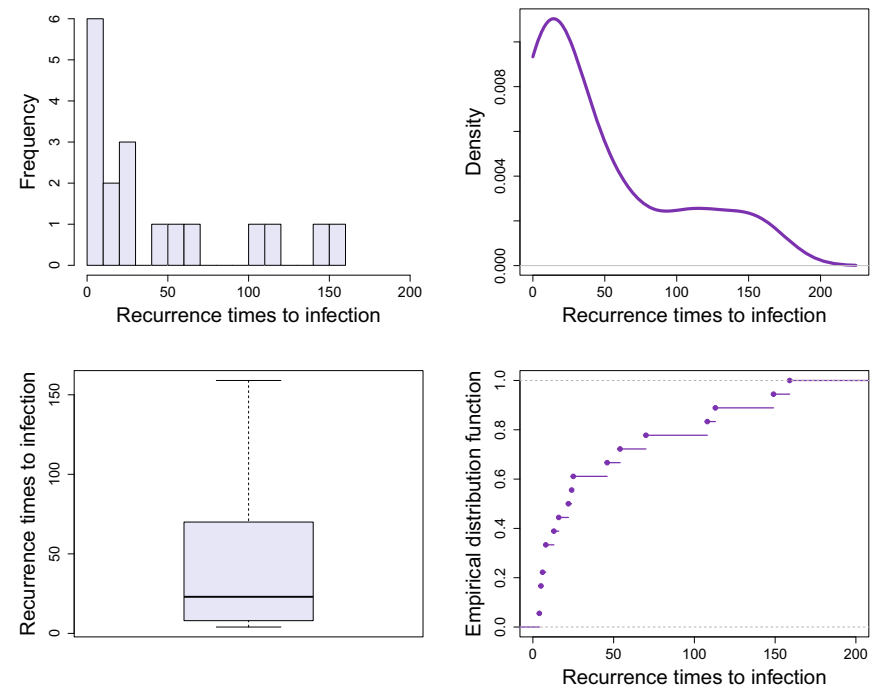


Fig. 6.2 Graphical summary of the kidney data using a histogram, a density plot, a boxplot, and an empirical cumulative distribution function (ECDF)

A histogram gives a comprehensive overview of the frequency of binned recurrence times corresponding to the number of patients in the respective bins. In contrast, a density plot shows a smoothed probability density, allowing to obtain a global overview of the entire population, whereas a boxplot focuses on the IQR corresponding to 50% of the population and its median value. Yet a different perspective is provided by the ECDF, which integrates over recurrence times to give

a global overview of the percentage of the population experiencing an event (in our case, a recurring infection).

Overall, this example shows how summary statistics, e.g., for descriptive statistics, and graphical visualizations are used and why such measures provide valuable information for an exploratory data analysis.

6.2 Sample Estimators

Statistics theory in general, and statistical inference in particular, is concerned with the effective use of a data sample, collected from a given population, to devise a model, which involves the estimation of some unknown parameters. The parameters of interest, which may have some physical interpretation, generally provide a concise description of relatively complex data or represent some essential properties of the system under consideration.

The form of the parameterization, i.e., the choice of the model, can be motivated by the following considerations:

1. The potential physical interpretation of the values of the parameters,
2. Some theoretical properties of the parameters, e.g., the stability of the values of the parameters,
3. The properties of the procedure used to estimate the parameters, e.g., the numerical stability of the procedure.

In the following, we will assume that we have a data sample $x_1, x_2, \dots, x_n$ with n independent observations on a random variable X defined on a probability space and the probability distribution of X depends on unknown parameters, say m , denoted by the vector $\theta = (\theta_1, \theta_2, \dots, \theta_m)$.

6.2.1 Point Estimation

Suppose that $m = 1$. Therefore, $\theta = \theta_1$, and we have only a single parameter to estimate. The point estimation of the parameter θ consists of constructing a function h such that $h(x_1, x_2, \dots, x_n)$ provides a “good” estimate of the parameter θ , i.e., it should be closely concentrated around the true value of θ .

To explain this, we state the following definitions.

Definition 6.1 The random variable $h(X_1, X_2, \dots, X_n)$ is called a point estimator of θ , and its value, $h(x_1, x_2, \dots, x_n)$, for the sample under investigation, is called the point estimate of θ .

Definition 6.2 For a given value of θ , the probability distribution of the random variable $h(X_1, X_2, \dots, X_n)$, denoted $f_h(\cdot|\theta)$, is called the *sampling*

distribution. Theoretically, the probability distribution, of the random variable $h(X_1, X_2, \dots, X_n)$, can be obtained as follows:

- Transform $\begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{pmatrix}$ to $\begin{pmatrix} Y_1 = h(X_1, X_2, \dots, X_n) \\ Y_2 = X_2 \\ \vdots \\ Y_n = X_n \end{pmatrix}$.
- Find the joint probability distribution of $Y_1, Y_2, \dots, Y_n$.
- Find the marginal probability distribution of Y_1 .

Note that the estimate of θ is likely to vary when we consider a different data sample. Therefore, it is more suitable to generalize the point estimation to interval estimation by constructing two functions, say h_1 and h_2 , with $h_1(x_1, x_2, \dots, x_n) < h_2(x_1, x_2, \dots, x_n)$, such that θ is likely to lie within the interval

$$[h_1(x_1, x_2, \dots, x_n); h_2(x_1, x_2, \dots, x_n)]. \quad (6.19)$$

The difference between point estimation methods revolves around the choice of the function h . For a fixed value of the parameter θ , the sampling distribution $f_h(\cdot|\theta)$ generally depends upon θ , the sample size n , as well as other unknown parameters, which may not be of interest.

The aim of statistical inference is to assess or estimate this variation from sample data. Some detailed discussions and illustrations of the most commonly used approaches are the purpose of the subsequent sections in this chapter as well as Chaps. 10, 11, 12, and 18.

6.2.2 Unbiased Estimators

The aim of a point estimation is to find a function h , which is “closely concentrated around the true value θ .” One of the most commonly used approaches to mathematically express “closely concentrated around the true value θ ” is to require that

$$\mathbb{E}[h(X_1, X_2, \dots, X_n)] = \theta, \quad (6.20)$$

and the variance $\mathbb{V}(h(X_1, X_2, \dots, X_n))$ is small for any value of θ .

To define this formally, we give a couple of definitions.

Definition 6.3 The estimator $h(X_1, X_2, \dots, X_n)$ is called an *unbiased estimator* of θ if

$$\mathbb{E}[h(X_1, X_2, \dots, X_n)] = \theta, \quad \forall \theta \in \Theta, \quad \forall n \in \mathbb{N},$$

where Θ denotes the parameter space, i.e., the set of all possible values of θ , and n is the size of the data sample.

Definition 6.4 The estimator $h(X_1, X_2, \dots, X_n)$ is called an *asymptotically unbiased estimator* of θ if

$$\mathbb{E}[h(X_1, X_2, \dots, X_n)] \longrightarrow \theta, \text{ as } n \longrightarrow \infty,$$

where n denotes the size of the data sample.

Definition 6.5 If $h_1(X_1, X_2, \dots, X_n)$ and $h_2(X_1, X_2, \dots, X_n)$ are two unbiased estimators of θ , then

- h_1 is said to be *more efficient* than h_2 if

$$\mathbb{V}[h_1(X_1, X_2, \dots, X_n)] < \mathbb{V}[h_2(X_1, X_2, \dots, X_n)], \quad \forall \theta \in \Theta, \quad \forall n \in \mathbb{N};$$

- h_1 is said to be *asymptotically more efficient* than h_2 if

$$\mathbb{V}[h_1(X_1, X_2, \dots, X_n)] < \mathbb{V}[h_2(X_1, X_2, \dots, X_n)], \quad \text{as } n \longrightarrow \infty.$$

In either cases, h_1 would be preferred compared to h_2 .

Definition 6.6 An unbiased estimator of θ , $h^*(X_1, X_2, \dots, X_n)$, such that

$$\mathbb{V}[h^*(X_1, X_2, \dots, X_n)] \leq \mathbb{V}[h(X_1, X_2, \dots, X_n)], \quad \forall \theta \in \Theta,$$

for any other unbiased estimator, h , of θ is called the *minimum variance unbiased estimator*. In other words, h^* is the “best” unbiased estimator of θ .

Although unbiased estimators have some desired properties, it may happen that:

- The unbiased estimator is difficult to obtain or doesn't exist.
- The unbiased estimator is not a satisfactory estimate for the problem of interest.
- The available unbiased estimator has large variances for some values of θ .

6.2.3 Biased Estimators

To introduce this concept, we start with some definitions.

Definition 6.7 Suppose that $h(X_1, X_2, \dots, X_n)$ is an estimator of θ such that

$$\mathbb{E}[h(X_1, X_2, \dots, X_n)] = \theta + g(\theta, n),$$

where n is the size of the data sample and $g(\theta, n)$ is called bias. If the bias $g(\theta, n) \neq 0$, then h is called a *biased estimator* of θ .

Note that if $\forall \theta \in \Theta, \lim_{n \rightarrow \infty} g(\theta, n) \rightarrow 0$, then h is an asymptotically unbiased estimator of θ ,

When $h(X_1, X_2, \dots, X_n)$ is a biased estimator of θ , the variance of h , $\mathbb{V}(h)$, is not a suitable measure of the concentration of the values of h around θ , and the *mean square error (MSE)* of h is preferred instead.

Definition 6.8 The *MSE* of an estimator of θ , $h(X_1, X_2, \dots, X_n)$, is defined as follows:

$$MSE(h(X_1, X_2, \dots, X_n)) = \mathbb{E} \left[(h(X_1, X_2, \dots, X_n) - \theta)^2 \right] \quad (6.21)$$

$$= \mathbb{E}(h(X_1, X_2, \dots, X_n)) + [g(\theta, n)]^2. \quad (6.22)$$

Note that for an unbiased estimator h , we have $MSE(h) = \mathbb{V}(h)$.

Definition 6.9 Let $h_1(X_1, X_2, \dots, X_n)$ and $h_2(X_1, X_2, \dots, X_n)$ be two estimators of θ . Then,

- h_1 is said to be more efficient than h_2 if

$$MSE[h_1(X_1, X_2, \dots, X_n)] < MSE[h_2(X_1, X_2, \dots, X_n)], \quad \forall \theta \in \Theta, \quad \forall n \in \mathbb{N};$$

- h_1 is said to be asymptotically more efficient than h_2 if

$$MSE[h_1(X_1, X_2, \dots, X_n)] < MSE[h_2(X_1, X_2, \dots, X_n)], \quad \text{as } n \rightarrow \infty.$$

6.2.4 Sufficiency

There is no general approach for finding a suitable estimator for a parameter θ , and we often resort to a specified technique, which provides a good estimator of θ . However, there is a general framework, known as the *principle of sufficiency*, which can be applied when estimating parameters from certain distributions.

Definition 6.10 Let $\xi(X_1, X_2, \dots, X_n)$ denote a statistic, i.e., a function of the random variables $X_1, X_2, \dots, X_n$, which doesn't depend upon any unknown parameter.

Then, the statistic $\xi(X_1, X_2, \dots, X_n)$ is said to be *sufficient* for the unknown parameter θ if the conditional distribution of $X_1, X_2, \dots, X_n$ given the value of ξ doesn't depend upon θ .

The principle of sufficiency [163] states that all the useful information in a random sample, for inference purposes, is contained in a sufficient statistic, if one exists. Therefore, any statistical procedure for making inferences about a parameter, θ , should be based on a sufficient statistic, if one exists.

Finding a sufficient statistic is not a trivial problem. However, there are some theoretical tools, which could be valuable in the formulation of the problem. One of them is the Fisher-Neyman factorization theorem [163, 358].

Theorem 6.1 (Fisher-Neyman Factorization Theorem) *A statistic $\xi(X_1, X_2, \dots, X_n)$ is said to be sufficient for θ if and only if the joint probability distribution function of $X_1, X_2, \dots, X_n$, denoted f_X , can be written in the following form:*

$$f_X(x_1, x_2, \dots, x_n | \theta) = s(x_1, x_2, \dots, x_n) g(\xi(x_1, x_2, \dots, x_n), \theta),$$

where s and g are non-negative functions.

Problem 6.1 (Sufficiency) Let $X_1, \dots, X_n$ be n independent random variables following a normal distribution $N(\mu, \sigma^2)$, where σ^2 is known. Find a sufficient statistic for the unknown parameter μ .

The probability density function is given by

$$f(x | \mu) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

The corresponding joint probability density function is given by

$$f_X(x_1, x_2, \dots, x_n | \mu) = \prod_{i=1}^n f(x_i | \mu) \quad (6.23)$$

$$= \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \quad (6.24)$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma} \right)^n e^{-\frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2} \quad (6.25)$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma} \right)^n e^{-\frac{1}{2\sigma^2} (\sum_{i=1}^n (x_i - \bar{x})^2) + n(\bar{x} - \mu)^2} \quad (6.26)$$

$$= \left[e^{-\frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \bar{x})^2} \right] \left[\left(\frac{1}{\sqrt{2\pi}\sigma} \right)^n e^{-\frac{1}{2\sigma^2} n(\bar{x} - \mu)^2} \right] \quad (6.27)$$

$$= [s(x_1, x_2, \dots, x_n)] [g(\bar{x}, \mu)], \quad (6.28)$$

where $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Therefore, from the Fisher-Neyman factorization theorem, $\bar{x}$ is a sufficient statistic for μ .

Problem 6.2 (Counter-Example of Sufficiency) Let $X_1, \dots, X_n$ be n independent random variables following a Cauchy distribution with parameter θ , i.e.,

$$f(x|\theta) = \frac{1}{\pi[1 + (x - \theta)^2]}, \quad -\infty < x < +\infty.$$

The corresponding joint probability density function is given by

$$f_X(x_1, x_2, \dots, x_n|\theta) = \prod_{i=1}^n f(x_i|\theta) \quad (6.29)$$

$$= \prod_{i=1}^n \frac{1}{\pi[1 + (x_i - \theta)^2]}. \quad (6.30)$$

Clearly, the joint probability density function (6.29) cannot be factorized in terms of $s(x_1, x_2, \dots, x_n)g(\xi(x_1, x_2, \dots, x_n), \theta)$. Therefore, a sufficient statistic for the parameter θ of a Cauchy distribution doesn't exist.

A typical class of probability distributions, satisfying the Fisher-Neyman factorization theorem, is the *regular exponential class (REC)*.

Definition 6.11 A probability distribution function, $f(x|\theta)$, belongs to the REC if

$$f(x|\theta) = e^{a(\theta)b(x)+c(\theta)+d(x)}. \quad (6.31)$$

Some well-known distributions, which belong to the REC, include:

- The binomial distribution $\mathcal{B}(n, \theta)$, where n is known.
Since

$$f(x|\theta) = \binom{n}{x} \theta^x (1 - \theta)^{n-x}, \quad x = 0, \dots, n,$$

then we have $a(\theta) = \log\left(\frac{\theta}{1-\theta}\right)$, $b(x) = x$, $c(\theta) = n\log(1 - \theta)$, $d(x) = \log\left(\binom{n}{x}\right)$.

- The Poisson distribution: Poisson (θ) given by

$$f(x|\theta) = \frac{\theta^x e^{-\theta}}{x!}, \quad x = 0, 1, \dots,$$

where $a(\theta) = \log\theta$, $b(x) = x$, $c(\theta) = -\theta$, $d(x) = -\log(x!)$.

- The normal distribution $\mathcal{N}(\mu, \sigma^2)$, where either μ or σ^2 is known.
- The negative binomial distribution with parameters (r, θ) , where r is known.
- The gamma distribution with parameters (α, β) , where the shape parameter α is known.
- The geometric distribution.
- The negative exponential distribution.

The aforementioned important properties of the point estimator (bias, mean square error, sufficiency) provide “out-of-the-box” tools to assess the quality or the qualitative behavior of the estimator.

6.3 Bayesian Inference

The principal idea of Bayesian inference is based on utilizing Bayes’ theorem. We formulate Bayes’ theorem here for continuous random variables:

$$p(\theta|x) = \frac{f(x|\theta)p(\theta)}{p(x)} \quad (6.32)$$

with

$$p(x) = \int f(x|\theta)p(\theta) \quad (6.33)$$

Here, θ is a parameter of a model, $f(x|\theta)$, for which we would like to make an estimate based on the observed data x .

In Bayesian inference, it is common to denote the above terms in the following way:

$$\underbrace{p(\theta|x)}_{\text{posterior}} = \underbrace{f(x|\theta)}_{\text{likelihood}} \underbrace{p(\theta)}_{\text{prior}} / \underbrace{p(x)}_{\text{evidence}} \quad (6.34)$$

whereas the evidence, $(p(x))$, is given by Eq. 6.33. The prior distribution $p(\theta)$ summarizes our knowledge about the parameter θ before we observe the data x . That means the prior is data independent. The likelihood $f(x|\theta)$ defines a statistical model from which the data could have been generated. Finally, the posterior distribution $p(\theta|x)$ gives us an updated distribution of the parameter θ . Here, “updated” means with respect to our knowledge before the data, given by the prior.

To make a Bayesian inference about a model parameter, θ , we need in addition to the likelihood $f(x|\theta)$, which defines the statistical model, a prior distribution over θ , given by $p(\theta)$. This distribution needs to be determined independent of the observed data x . That means the knowledge about x must not be used to specify

the distribution $p(\theta)$. This explains the name “prior distribution,” because this distribution should be specified before the data are observed.

Finally, for making an inference about a model parameter θ , one needs to specify a rule that is used to extract this information from the posterior distribution $p(\theta|x)$. There are several measures of location that can be used, and we want to mention three of them.

$$\text{posterior mean } \bar{\theta} = \int \theta p(\theta|x) d\theta \quad (6.35)$$

$$\text{posterior median } \bar{\theta} = \theta_m : \int^{\theta_m} p(\theta|x) d\theta = 0.5 \quad (6.36)$$

$$\text{posterior mode } \bar{\theta} = \operatorname{argmax}_{\theta'} p(\theta'|x) \quad (6.37)$$

The first is the posterior mean, which corresponds to the expected value of θ . The second is the posterior median, which has the advantage of being more robust against outliers, compared to the posterior mean. Finally, the posterior mode is the maximum of the posterior distribution. For this reason, it is also called the *maximum a posteriori (MAP)*. An advantage of the MAP estimator over the posterior mean and the posterior median is that if $p(\theta|x)$ is graphically visualized, the MAP can be easily spotted by eye. For this reason, in the following, we use the MAP estimator.

Bayesian inference can be used for three different purposes that will be discussed in more detail in the following sections:

1. Parameter estimation,
2. Prediction,
3. Model selection.

Before we continue, we would like to note that the first two points above, i.e., parameter estimation and prediction, are similar since a parameter estimation is a prediction, namely, a prediction of the (true) parameter value. Alternatively, one can see prediction as parameter estimation, where the value of the error is the parameter to be estimated.

Importantly, the difference is that, usually, parameter estimation is a one-step process, whereas prediction is a two-step process. Specifically, one can see parameter estimation as the following mapping:

$$D \rightarrow \text{parameter estimation.} \quad (6.38)$$

Here, the data, D , are used for estimating the value of a parameter. In contrast, for prediction, we have the following two mappings:

$$D \rightarrow \text{parameter estimation (of } \alpha), \quad (6.39)$$

$$M(\alpha, D') \rightarrow \text{prediction (of the error).} \quad (6.40)$$

The first mapping estimates a parameter, let's call it α , and the second mapping uses this parameter (in addition with another data set D') to make an estimation of the error of a model M , and this mapping is called the prediction. Hence, prediction requires two mappings, two data sets (D and D'), and the coupling of both mappings by using the estimated parameter α as the parameter of the model M .

Leaving philosophical considerations aside, the understanding of the principal idea behind Bayesian inference is quite simple, as one can see from the above formulation. However, practically, the realization of a Bayesian inference can be intricate. In the following, we will not present a comprehensive discussion but limit our focus to a special case of Bayesian inference, which is based on the so-called conjugate priors.

6.3.1 Conjugate Priors

In Table 6.2, we present an overview of known conjugate priors for likelihood functions. For all of these configurations, the calculation of the posterior distribution is straightforward because a closed-form expression for the posterior is known due to the particular form of the (conjugate) prior, and it is given in the first column of the table.

This information enables to simplify the estimation considerably because for all the cases shown in Table 6.2, a Bayesian inference can be conducted analytically without any numerical approximations.

6.3.2 Continuous Parameter Estimation

In this section, we present an application of conjugate priors, which illustrates the working mechanism of Bayesian inference.

Problem 6.3 Suppose that there is an infinitely large urn, containing a fraction θ of **red balls** and a fraction $1 - \theta$ of **blue balls**. We draw n balls from this urn and observe x red balls. What is θ ?

This problem is solved via a five-step procedure:

Table 6.2 Conjugate priors and their corresponding likelihood functions

Posterior	Likelihood	Prior
Normal: $\mathcal{N}(\frac{\mu_P \cdot \sigma_L^2 + \mu_L \cdot \sigma_P^2}{\sigma_P^2 + \sigma_L^2}, \frac{\sigma_P^2 \cdot \sigma_L^2}{\sigma_P^2 + \sigma_L^2})$	Normal: $\mathcal{N}(\mu_L, \sigma_L^2)$	Normal: $\mathcal{N}(\mu_P, \sigma_P^2)$
Beta: $\mathcal{Be}(x + \alpha, n - x + \beta)$	Binomial: $\mathcal{B}(n, \theta)$	Beta: $\mathcal{Be}(\alpha, \beta)$
Gamma: $\mathcal{G}(\alpha + \gamma, \beta + x)$	Gamma: $\mathcal{G}(\gamma, \delta)$	Gamma: $\mathcal{G}(\alpha, \beta)$
Gamma: $\mathcal{G}(\alpha + 0.5, \beta + (\mu - x)^2/2)$	Normal: $\mathcal{N}(\mu_L, \sigma_L^2)$	Gamma: $\mathcal{G}(\alpha, \beta)$

- (1) Define the likelihood.
- (2) Define the prior.
- (3) Calculate the denominator $p(x)$.
- (4) Calculate the posterior distribution $p(\theta|x)$.
- (5) Obtain an estimate of the parameter from the posterior distribution.

Step 1 The likelihood to draw n balls of which x are red, from an infinitely large urn, is a binomial distribution given by

$$f(x|\theta) = \binom{n}{x} \theta^x (1 - \theta)^{n-x}. \quad (6.41)$$

We would like to note that the infinite size of the urn is important here as it guarantees that the fraction of red balls, given by θ , does not change when removing some balls from the urn. If θ would not be constant, we would need to account for this.

Alternatively, instead of assuming an infinitely large urn, we could also draw the n balls one after another, and each time, after we observed the color of a ball, we place it back in the urn. This way, the number of balls in the urn would be constant, and this would also guarantee a constant probability θ of drawing a red ball.

Step 2 From Table 6.2, we see that the conjugate prior of a binomial distribution is a beta distribution. Hence, the prior is given by the following beta distribution:

$$p(\theta) = \mathcal{Be}(\theta|\alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1} (1 - \theta)^{\beta-1}. \quad (6.42)$$

The value of $\mathcal{Be}(\theta|\alpha, \beta)$ is defined for any θ between 0 and 1. That means for any values of the parameters α and β , there will be always a value of $\theta \in [0, 1]$, which could be used as an estimate of the proportion of red balls.

Step 3 The next step is to calculate the denominator in Eq. 6.32 using Eq. 6.33, as follows.

$$\begin{aligned} p(x) &= \int_0^1 f(x|\theta) p(\theta) d\theta \\ &= \int_0^1 \binom{n}{x} \theta^x (1 - \theta)^{n-x} \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1} (1 - \theta)^{\beta-1} d\theta \\ &= \binom{n}{x} \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \int_0^1 \theta^{x+\alpha-1} (1 - \theta)^{n-x+\beta-1} d\theta \\ &= \binom{n}{x} \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \int_0^1 \theta^{\alpha'-1} (1 - \theta)^{\beta'-1} d\theta \end{aligned}$$

$$\begin{aligned}
&= \binom{n}{x} \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(\alpha')\Gamma(\beta')}{\Gamma(\alpha' + \beta')} \\
&= \binom{n}{x} \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(x + \alpha)\Gamma(n - x + \beta)}{\Gamma(\alpha + n + \beta)} \quad (6.43)
\end{aligned}$$

Here, we defined $\alpha' = x + \alpha$ and $\beta' = n - x + \beta$ to simplify the notation.

Step 4 Finally, we calculate the posterior probability from the above components. After this step, we will also have a full appreciation of using a beta distribution as a prior distribution and also understand what we mean by a conjugate prior.

$$\begin{aligned}
p(\theta|x) &= \frac{f(x|\theta)p(\theta)}{p(x)} \\
&= \frac{\Gamma(\alpha + n + \beta)}{\Gamma(x + \alpha)\Gamma(n - x + \beta)} \theta^x (1 - \theta)^{n-x} \theta^{\alpha-1} (1 - \theta)^{\beta-1} \\
&= \frac{\Gamma(\alpha + n + \beta)}{\Gamma(x + \alpha)\Gamma(n - x + \beta)} \theta^{x+\alpha-1} (1 - \theta)^{n-x+\beta-1} \quad (6.44)
\end{aligned}$$

As one can see, the resulting posterior distribution $p(\theta|x)$ is again a beta distribution, as is the prior, however, with different parameters. In general, whenever the prior and the posterior are probability distributions from the same family, we call the prior *conjugated*. The reason why this is the case for the above example is that the beta prior distribution and the binomial likelihood “fit together” or conjugate each other. Hence, the combination of a prior and a likelihood is key for this analytical result.

In Fig. 6.3, we show nine numerical examples for different values of the sample size n and the proportion of red balls in the urn θ . In the first row, we assume that $\theta = 1.0$, meaning that there are only red balls in the urn. For this reason, we observe always as many red balls as the number of samples we draw, i.e., $x = n$. In each figure, we show the prior distribution in red and the estimated posterior distribution in blue. We include the prior distribution to remind that our initial guess about θ is uninformative, by considering a uniform prior distribution. Drawing samples, we can see that by increasing the sample size (from left to right), our estimates of θ are getting closer to the true value of $\theta = 1.0$, which is shown as dotted vertical lines.

In the second row of Fig. 6.3, we assume a fixed proportion of red ball given by $\theta = 0.2$ (dotted vertical lines). Furthermore, we also assume that we observe always $x = \lfloor \theta \cdot n \rfloor$ red balls. Here, the symbol $\lfloor y \rfloor$ indicates the floor function, which down rounds the real value y to the nearest integer value. This rounding is necessary because x needs to be an integer value. The assumption $x = \lfloor \theta \cdot n \rfloor$ is quite strict and artificial, but it will help us in making a point, as we will see below.

From the second row in Fig. 6.3, we can see that our estimates of θ are always correct, for any value of the sample size n , because the maximum of the posterior distribution is always exactly at $\theta = 0.2$. However, the difference between the three

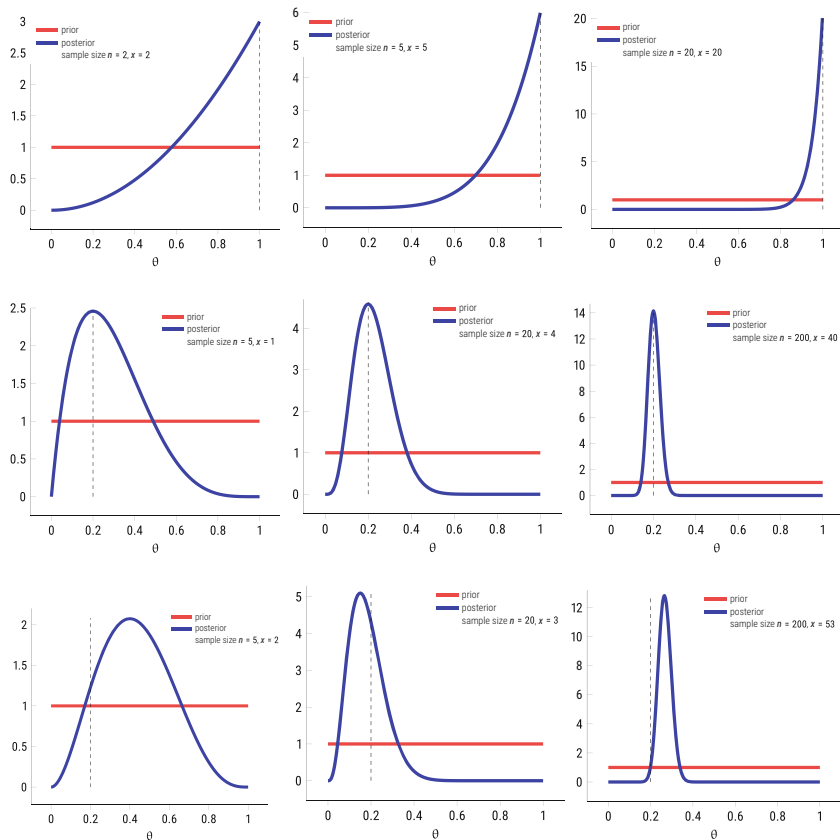


Fig. 6.3 Examples of a Bayesian inference for a continuous parameter p of a binomial likelihood function

examples is that our uncertainty about this value decreases with the increase of the sample size from left to right. Here, by uncertainty, we mean the variance of the posterior distribution. This is plausible because the more information we have (larger sample sizes), the more false estimates can be potentially eliminated.

In row 3 of Fig. 6.3, we assume again a fixed proportion of red ball given by $\theta = 0.2$ (dotted vertical lines). However, this time, we do not fix the number of observed red balls by assuming $x = \lfloor \theta \cdot n \rfloor$. Instead, we draw n samples from a binomial distribution with probability $\theta = 0.2$, as specified in Eq. 6.41. Hence, we are actually conducting experiments to generate the data x , corresponding to the number of observed red balls. In contrast with the second row in Fig. 6.3, this time, all modes of the posterior distribution do not exactly correspond to $\theta = 0.2$, but are located around this value. The reason for this is that in none of the experiments, the number of observed red balls corresponds exactly to $x = \lfloor \theta \cdot n \rfloor$ (see the figure legend for observed x values in the three cases). This leads to an error, which has

some consequences. Interestingly, despite the different locations of the mode of the posterior distribution, the variance is reduced with an increased sample size, as in the second row in Fig. 6.3.

6.3.2.1 Example: Continuous Bayesian Inference Using R

Although the results in the previous section can be obtained analytically, writing a computer program can be helpful. For this reason, in Listing 6.2, we provide the R script to reproduce the results in Fig. 6.3. When executing this program repeatedly, one will realize that each time, a slightly different result is obtained. This aspect is studied in Exercise 6.

Listing 6.2: Example of continuous Bayesian inference using R, see Fig. 6.3

```
p.red <- 0.2
n <- 200 # sample size
y <- rbinom(1, n, p.red) # red balls

a <- 1; b <- 1
p <- seq(0, 1, 0.001)
prior <- dbeta(p, a, b)
a2 <- y+a
b2 <- n-y+b
post <- dbeta(p, a2, b2)

ym <- max(c(prior, post))
plot(p, prior, col = "red", type="l", lwd = 7,
ylim = c(0, ym), cex.lab = 1.8, cex.axis = 1.6, ylab = "",
font.lab=2, xlab=expression(theta))
lines(p, post, col = "blue", lwd = 7)
legend("topright", legend = c("prior", "posterior",
paste("sample size =", n, ", x =", y)),
fill = c("red", "blue", "white"),
border=c("red", "blue", "white"))
abline(v=0.2, lty=2, lwd=2)
```

6.3.3 Discrete Parameter Estimation

In this section, we show how to conduct a Bayesian inference for a discrete parameter. That means, in contrast with Sect. 6.3.2, where we assumed θ is a continuous parameter, in this section, we assume that θ is discrete, i.e., it can take values $\{\theta_1, \dots, \theta_k\}$. In this case, we make a Bayesian inference from

$$p(\theta_i|x) = \frac{f(x|\theta_i)p(\theta_i)}{\sum_j f(x|\theta_j)p(\theta_j)}, \quad (6.45)$$

which is the discrete version of Eq. 6.32.

In Sect. 6.3.2, we provided an analytical solution to Problem 6.3 when a binomial likelihood is combined with a prior corresponding to a beta distribution. The rationale behind this is that the posterior is a beta distribution and its parameters are given by the parameters of the likelihood and the prior. However, you may wonder, what if there is no known analytical solution to a problem? Or what if there is such a solution but you don't know it?

In the following, we study again Problem 6.3; however, this time we do not make use of the elegant analytical approach based on a conjugate prior for the continuous parameter θ as in Sect. 6.3.2, but we will study a discrete version using numerical means.

Let's assume that we limit the values of θ_i to d equally spaced, discrete values between 0 and 1 and that we do not have any information that would favor any of these values. For this reason, we assume a non-informative prior distribution, which assigns to each value of θ_i the same probability. In Listing 6.3, we present the R script, which implements Eq. 6.45 to produce the results shown in Fig. 6.4.

For Fig. 6.4, we used three different values of d to show the influence, of the number of points used to evaluate the parameter θ , on the study. Specifically, for row 1, we use $d = 5$; for row 2, $d = 20$; and for row 3, $d = 50$. Moreover, for each column, we use a different sample size: column 1 $n = 5$ and column 2 $n = 20$.

Clearly, for higher values of d , the estimated posterior distributions become more and more smooth and very similar to the results from the analytical solution (see Fig. 6.3). This is an important observation because it shows that the numerical solution is as good as the analytical solution for large values of d . Interestingly, d does not need to be extremely large in our example in order to provide sensible results, and even values as low as $d = 5$ give a meaningful approximation.

The observation from this specific example is actually true in general, which means we can always approximate an analytical solution numerically, and the difference between both solutions decreases with increased values of d . However, practically, the disadvantage of a numerical solution is that it is usually more time-consuming because the likelihood and the prior need to be evaluated at many different parameter points. For our one-dimensional problem considered here, this is not a problem; however, for high-dimensional problems say with ten variable, this can be computationally expensive, because there are d^{10} evaluations for the *ten* variables, each with d discrete values. Fortunately, there are advanced Monte Carlo approaches that have been developed for such cases.

In addition to the asymptotical equivalence between analytical and numerical solutions, another advantage of using a discrete Bayesian inference is that there is more flexibility in the choice of the priors compared to the continuous approach. Specifically, using a beta distribution as a prior does not actually allow to assume any arbitrary functional shape, but it is limited to the forms that can be modeled by different values of the parameters α and β . For instance, the distribution shown in Fig. 6.5 cannot be realized with a beta distribution or any other analytical distribution. In this respect, a discrete numerical approach is more flexible than an analytical one.

Listing 6.3: Example of discrete Bayesian inference using R, see Fig. 6.4

```

Bayes.I <- function(LH, prior){
  evi <- LH %**% prior
  post <- (LH * prior)/evi

  return(post)
}

d <- 5 # number of discrete values
p.red <- 0.2 # true value
n <- 5 # sample size
y <- 3 #rbinom(1, n, p.red) # red balls

z <- 0:n
theta <- seq(0, 1, length.out=d)
prior <- vector(mode="numeric", length=d) + 1/d
LH <- vector(mode="numeric", length=d)
LH <- dbinom(y, n, theta)
LH <- LH/sum(LH)

post <- Bayes.I(LH, prior)

ym <- max(c(prior, post))
plot(theta, prior, col = "blue", type="l", lwd = 1, ylim = c(0,
  ym), cex.lab = 1.8, cex.axis = 1.6, ylab = "", font.lab=2,
  xlab=expression(theta))
points(theta, prior, col = "blue", lwd=7)
lines(theta, post, col = "purple", lwd = 6)
points(theta, post, col = "purple", lwd = 7)
legend("topright", legend = c("prior", "posterior", paste("sample
  size =", n, ", ", x = " ", y)), fill = c("blue", "purple",
  "white"), border=c("blue", "purple", "white"))
abline(v=0.2, lty=2, lwd=2)

```

6.3.4 Bayesian Credible Intervals

In Sects. 6.3.2 and 6.3.3, we discussed Bayesian inference for point estimation. However, so far, we have not explicitly addressed the variability of such estimates. Instead, we only discussed measures of location like the posterior mean or maximum a posteriori (see Eqs. 6.35 and 6.37), for summarizing the information provided by the posterior distribution. Now, we turn our focus to the quantification of the variability of Bayesian point estimates by means of so-called **credible intervals**.

Suppose that for a given posterior distribution, $p(\theta|x)$, we would like to know what is the probability to observe a value of the model parameter θ within the interval (θ_l, θ_u) . Formally, this probability can be estimated by

$$Pr(\theta_l < \theta < \theta_u) = \int_{\theta_l}^{\theta_u} p(\theta|x)d\theta. \quad (6.46)$$

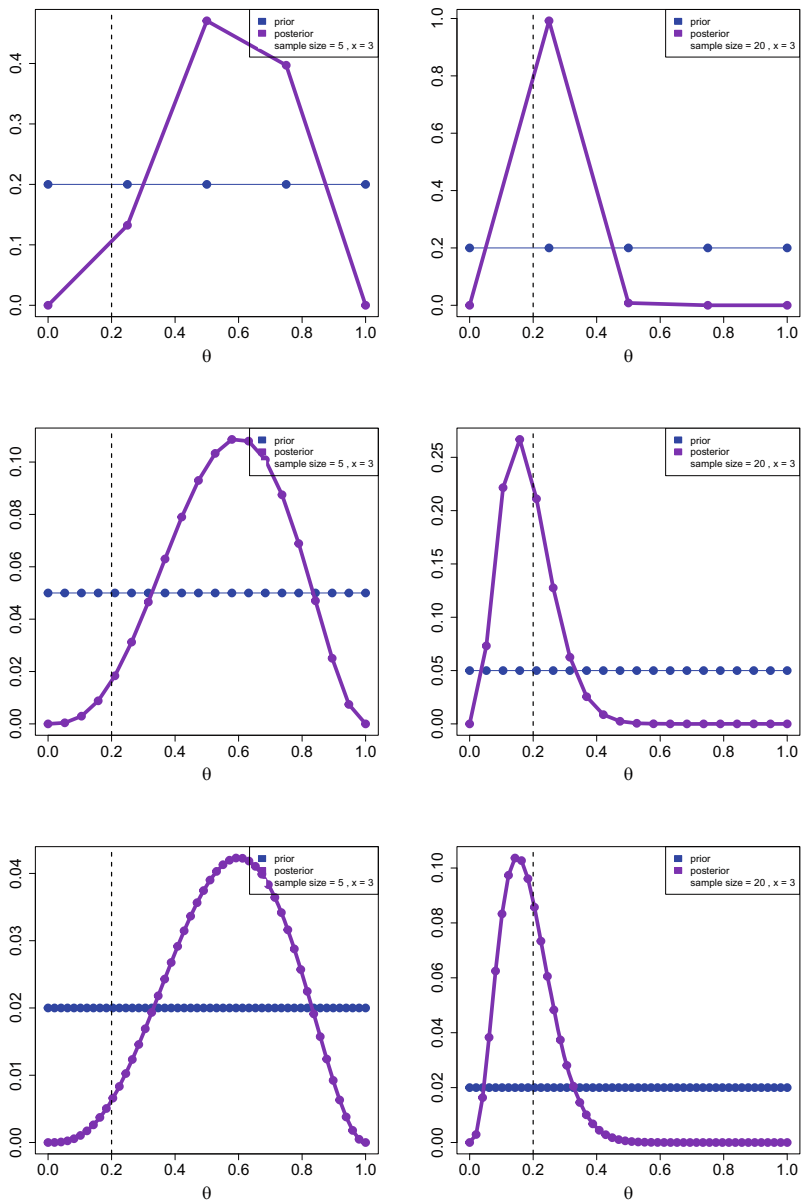


Fig. 6.4 Examples for a Bayesian inference for a discrete parameter θ of a discretized binomial distribution. The connecting lines between the points are only included to enhance the visual impression of the discrete distributions

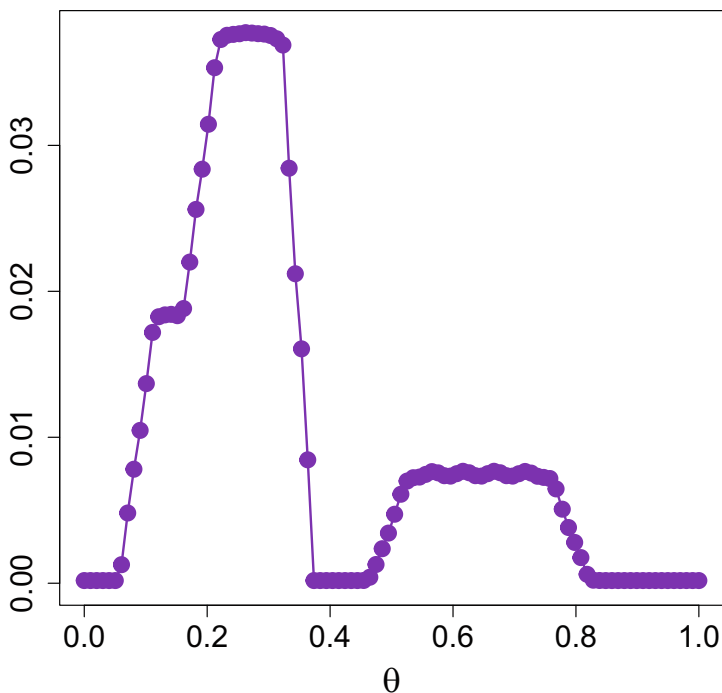


Fig. 6.5 Discrete prior distribution that cannot be realized with a beta distribution

If we write

$$1 - \alpha = \Pr(\theta_l < \theta < \theta_u), \quad (6.47)$$

for $\alpha \in [0, 1]$, then (θ_l, θ_u) is called the $(1 - \alpha) \times 100\%$ credible interval for θ .

The interpretation of the credible interval (θ_l, θ_u) is straightforward from Eq. 6.46, meaning that the probability to observe any value of θ within this interval is $1 - \alpha$.

Interestingly, there are actually many intervals within the posterior distribution that result in a probability of $1 - \alpha$. The two most common ones are shown in Fig. 6.6. The left figure shows

$$1 - \alpha = \int_{\theta_l}^{\theta_u} p(\theta|x) d\theta \quad \text{with } p(\theta_l|x) = p(\theta_u|x), \quad (6.48)$$

whereas the right figure depicts

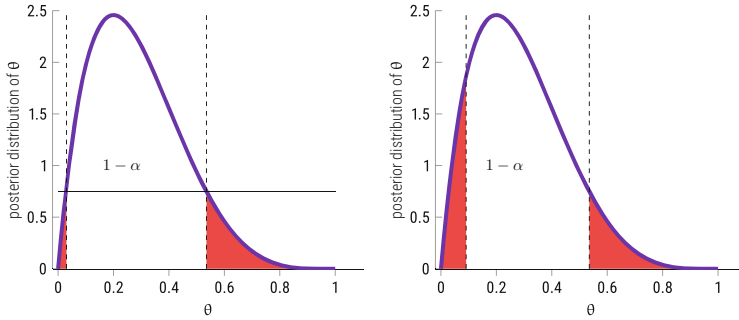


Fig. 6.6 Two examples for defining Bayesian credible intervals for the same underlying posterior distribution $p(\theta|x)$

$$1 - \alpha = \int_{\theta_l}^{\theta_u} p(\theta|x) d\theta \quad \text{with} \quad \frac{\alpha}{2} = \int_0^{\theta_l} p(\theta|x) d\theta = \int_{\theta_u}^{\infty} p(\theta|x) d\theta. \quad (6.49)$$

We would like to emphasize that both definitions use the same underlying posterior distribution $p(\theta|x)$. However, the values of θ_l and θ_u are set up differently.

The left credible interval has equal probability for the values of the posterior distribution at the boundaries of the interval, i.e., at θ_l and θ_u , whereas the right credible interval has an equal tail probability of $\alpha/2$ in the lower and upper tail of $p(\theta|x)$.

6.3.5 Prediction

There are two ways to make a systematic prediction about a distribution of values. The first is based on a prior and the second on the posterior distribution from a Bayesian inference.

$$\text{prior predictive density} \quad f(y') = \int f(y'|\theta) p(\theta) d\theta; \quad (6.50)$$

$$\text{posterior predictive density} \quad f(y') = \int f(y'|\theta) p(\theta|y) d\theta. \quad (6.51)$$

Here, $p(\theta)$ is the prior and $p(\theta|y)$ is the posterior distribution of the data denoted y . The distribution $f(y')$ corresponds to the predicted distribution of a new observation y' .

6.3.6 Model Selection

The last problem we discuss, and for which Bayesian inference can be used, is model selection. Suppose that we have two parametric models M_1 and M_2 and we would like to know which of these models is a better description of the observed data x . This problem is called *model selection*. In the following, we want to limit our discussion to two models M_1 and M_2 , but the extension to a larger number of models is possible.

If we have two models, we select model M_1 with probability $p(M_1)$ and model M_2 with probability $p(M_2)$. Because there are only two models, it holds that $p(M_1) + p(M_2) = 1$. A Bayesian approach to model selection is provided by using a so-called **Bayes factor (BF)**. A Bayes factor is defined as the posterior odds of model M_1 divided by the prior odds of model M_1

$$\text{BF} = \frac{\text{odds}_{\text{post}}(M_1)}{\text{odds}_{\text{prior}}(M_1)} \quad (6.52)$$

$$= \frac{p(M_1|x)/p(M_2|x)}{p(M_1)/p(M_2)} \quad (6.53)$$

The posterior distributions $p(M_i|x)$ are obtained from

$$p(M_i|x) = \frac{p(x|M_i)p(M_i)}{p(x)} \quad \text{for } i \in \{1, 2\}, \quad (6.54)$$

where $p(x|M_i)$ is called the *marginal likelihood* given by

$$p(x|M_i) = \int f(x|\theta_i, M_i)p(\theta_i)d\theta_i \quad \text{for } i \in \{1, 2\}. \quad (6.55)$$

This probability is called “marginal” because the likelihood of model M_i is integrated over its parameters θ_i .

If we substitute Eq. 6.54 into 6.53, then the evidence $p(x)$ cancels out, and a Bayes factor is given by the fraction of marginal likelihoods of the models, i.e.,

$$\text{BF} = \frac{p(x|M_1)}{p(x|M_2)}. \quad (6.56)$$

Once a BF has been calculated, its interpretation is simple. If $\text{BF} > 1$, then model M_1 fits better the data. On the other hand, if $\text{BF} < 1$, then model M_2 fits better the data. However, what is not so simple is to define a vector $\Theta = (\theta_1, \theta_2)$ so that a BF is either larger or smaller, and we are sufficiently confident to consider the findings robust.

6.4 Maximum Likelihood Estimation

Now, we turn our attention to another principal way for making a statistical inference about a model parameter. This is called *maximum likelihood estimation (MLE)*.

Definition 6.12 Let $f(x_i|\theta)$ be a probability function from which the observed data $x = \{x_1, \dots, x_n\}$ are sampled. Then $L(\theta|x) = f(x|\theta)$ is called the likelihood function because the distribution $f(x|\theta)$ is considered as a function of its parameter θ .

Definition 6.13 Based on the previous definition, a maximum likelihood estimation of a parameter can be defined as follows:

$$\bar{\theta} = \operatorname{argmax}_{\theta} L(\theta|x). \quad (6.57)$$

Hence, the maximum likelihood estimation is another way to obtain a point estimator for the parameter of a model.

Problem 6.4 Suppose that we have a coin and we perform n independent random experiments resulting in $x = \{x_1, \dots, x_n\}$ observations, where each x_i is either head (1) or tail (0). What is the probability of the coin to give a head?

For each conducted experiment, we can assume that the random variable x_i is drawn from a Bernoulli distribution. This results in the following likelihood function:

$$L(\theta|x) = \prod_{i=1}^n \theta^{x_i} (1 - \theta)^{1-x_i}. \quad (6.58)$$

For practical reasons, it is usually beneficial to take the logarithm of Eq. 6.58 and maximize $\log L(\theta|x)$ instead,

$$\log L(\theta|x) = \sum_{i=1}^n \left(x_i \log \theta + (1 - x_i) \log(1 - \theta) \right), \quad (6.59)$$

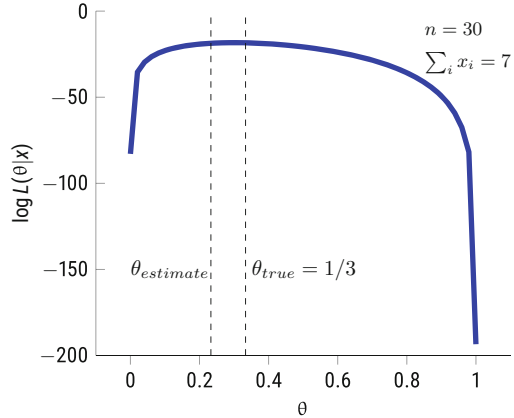
$$= \left(\sum_{i=1}^n x_i \right) \log \theta + \left(n - \sum_{i=1}^n x_i \right) \log(1 - \theta). \quad (6.60)$$

In order to find the maximum of $\log L(\theta|x)$, we calculate the first derivative and solve the following:

$$\frac{d \log L(\theta|x)}{d\theta} \stackrel{!}{=} 0 \quad \Rightarrow \quad (6.61)$$

$$\frac{\sum_{i=1}^n x_i}{\theta} - \frac{n - \sum_{i=1}^n x_i}{1 - \theta} = 0 \quad \Rightarrow \quad (6.62)$$

Fig. 6.7 Logarithmic likelihood function for $n = 30$ independent Bernoulli experiments where *seven* heads have been observed



$$\bar{\theta} = \frac{1}{n} \sum_{i=1}^n x_i. \quad (6.63)$$

To verify that this is indeed a maximum, and not a minimum, we could calculate the second derivative of $\log L(\theta|x)$ to confirm it is positive for $\bar{\theta}$,

$$\frac{d^2 \log L(\theta|x)}{d\theta^2} = -\frac{\sum_{i=1}^n x_i}{\theta^2} + \frac{n - \sum_{i=1}^n x_i}{(1 - \theta)^2}. \quad (6.64)$$

Alternatively, we show in Fig. 6.7 a graphical visualization of $\log L(\theta|x)$, given by Eq. 6.60, for an example, where $n = 30$ and *seven* heads have been observed. We can see that this function has its maximum at $\bar{\theta} = \frac{1}{n} \sum_{i=1}^n x_i = 0.23$, whereas the true value the parameter θ is $1/3$.

6.4.1 Asymptotic Confidence Intervals for MLE

The purpose of an MLE is to obtain a point estimator for a random sample that has been drawn from a parametric distribution. Again, this does not tell us anything about the variability of this estimate. To obtain some insights about this variability, we can utilize the result from the following theorem, which provides an estimate for the asymptotic sampling distribution of the MLE.

Theorem 6.2 *Let $\hat{\theta}$ be an MLE that maximizes $L(\theta|x)$, with $x = \{x_1, \dots, x_n\}$, and assume also that the second and third derivatives of the likelihood function exist. Then, the asymptotic distribution of $\hat{\theta}$ for $n \rightarrow \infty$ is given by*

$$\hat{\theta} \sim \mathcal{N}(\theta_0, (I(\theta_0))^{-1}). \quad (6.65)$$

Here, θ_0 is the true value of θ , and $I(\theta_0)$ is the Fisher information.

The results from this theorem mean that $\hat{\theta}$ is normally distributed with mean θ_0 and variance $I(\theta_0)$.

The Fisher information is defined as follows.

Definition 6.14 (Fisher Information) Let $x = \{x_1, \dots, x_n\}$ be a random variable with likelihood function $L(\theta|x)$ for a statistical model with parameter θ and $\lambda(\theta|x) = \log L(\theta|x)$ is the logarithm of this likelihood function. Assume that $\lambda(\theta|x)$ is twice differentiable with respect to θ . Then, the Fisher information $I(\theta)$ is defined by

$$I(\theta) = \mathbb{E} \left[\left(\lambda'(\theta|x) \right)^2 \right] \quad (6.66)$$

$$= \int \left(\lambda'(\theta|x) \right)^2 f(x|\theta) dx \quad (6.67)$$

For applications, an alternative formulation of the Fisher information proves useful, which we state below without proof.

Theorem 6.3 (Alternative Form of Fisher Information)

$$I(\theta) = -\mathbb{E} \left[\lambda''(\theta|x) \right] \quad (6.68)$$

$$= - \int \lambda''(\theta|x) f(x|\theta) dx. \quad (6.69)$$

To show how to calculate the Fisher information practically, let's consider the following example.

Problem 6.5 Suppose that we have a coin with a probability for head (1) of θ and a probability for tail (0) of $1 - \theta$. Hence, this model generates random variables x , from $\{0, 1\}$, with a Bernoulli distribution $f(x|\theta) = \theta^x \cdot (1 - \theta)^{1-x}$. What is the Fisher information?

If we observe just one sample, then

$$L(\theta|x) = \theta^x (1 - \theta)^{1-x} \quad (6.70)$$

is the likelihood function and

$$\lambda(\theta|x) = x \log \theta + (1 - x) \log(1 - \theta) \quad (6.71)$$

is the logarithmic likelihood. Calculating the first and second derivatives gives

$$\lambda'(\theta|x) = \frac{x}{\theta} - \frac{1-x}{1-\theta}; \quad (6.72)$$

$$\lambda''(\theta|x) = -\frac{x}{\theta^2} - \frac{1-x}{(1-\theta)^2}. \quad (6.73)$$

From this, the Fisher information writes as follows:

$$I(\theta) = -\mathbb{E}[\lambda''(\theta|x)] \quad (6.74)$$

$$= \frac{1}{\theta^2} \mathbb{E}[x] + \frac{1}{(1-\theta)^2} \mathbb{E}[1-x] \quad (6.75)$$

$$= \frac{1}{\theta^2} \mathbb{E}[x] + \frac{1}{(1-\theta)^2} (1 - \mathbb{E}[x]) \quad (6.76)$$

$$= \frac{1}{\theta} + \frac{1}{1-\theta}, \quad (6.77)$$

since $\mathbb{E}[x] = \theta$ for a Bernoulli distribution.

In principle, we can now calculate the Fisher information for any sample of arbitrary size, i.e., $x = \{x_1, \dots, x_n\}$, with a known likelihood function $L(\theta|x)$. In this case, we denote the Fisher information $I_n(\theta)$ to indicate that it is based on a sample of size n . However, for independent and identically distributed (iid) samples, this can be even simplified because the following relation holds between $I_n(\theta)$ and $I(\theta)$ (with $I(\theta) = I_{n=1}(\theta)$):

Theorem 6.4 (Additivity Property [172])

$$I_n(\theta) = nI(\theta). \quad (6.78)$$

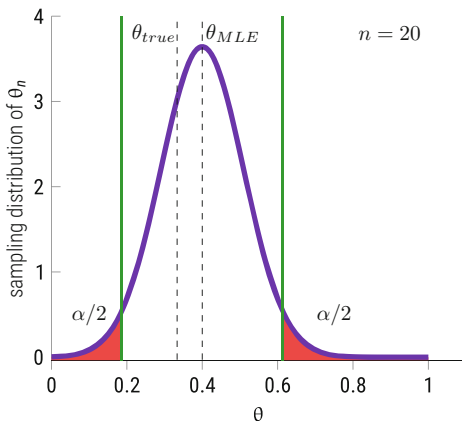
Problem 6.6 Suppose that we have a coin with a probability for head (1) of θ and a probability for tail (0) that is $1 - \theta$. The coin is thrown n times, generating $x = \{x_1, \dots, x_n\}$ with $x_i \in \{0, 1\}$. Hence, each sample x_i is drawn from a Bernoulli distribution $f(x_i|\theta) = \theta^{x_i}(1-\theta)^{1-x_i}$. What is the Fisher information?

Using the result from Theorem 6.4, the Fisher information for Problem 6.6 can be easily obtained for n draws of a coin, as follows:

$$I_n(\theta) = n \left(\frac{1}{\theta} + \frac{1}{1-\theta} \right). \quad (6.79)$$

In Fig. 6.8, we show an example of an asymptotic sampling distribution of θ_n for a Bernoulli sample with $n = 20$. The interval marked by the two vertical green lines corresponds to the $(1 - \alpha)$ confidence interval. Here, we use $\alpha = 0.05$, which means that the shown interval is the 95% confidence interval. The red areas under the distribution on the left and right sides correspond each to $\alpha/2$.

Fig. 6.8 Asymptotic sampling distribution of θ_n given by $\mathcal{N}(\theta, (I(\theta_0))^{-1})$



6.4.2 Bootstrap Confidence Intervals for MLE

In the previous section, we showed how to obtain confidence intervals for the MLE of a model parameter θ . These intervals are based on asymptotic results, i.e., they assume a large sample size n . However, this does not guarantee that for small n , the resulting confidence intervals are good approximations for the asymptotic results. For this reason, in practice and for small n , it is more appropriate to estimate confidence intervals numerically, using a bootstrap approach instead of results based on asymptotic considerations.

To estimate confidence intervals for MLE, we can apply the following procedure.

1. Estimate the MLE, $\hat{\theta}$, for $x = \{x_1, \dots, x_n\}$ with $x_i \sim f(x|\theta_0)$.
2. Generate new parametric bootstrap samples $x^b = \{x_1^b, \dots, x_n^b\}$ with $x_i^b \sim f(x|\hat{\theta})$.
3. Estimate new MLE, $\hat{\theta}^b$, for the new parametric bootstrap samples $x^b = \{x_1^b, \dots, x_n^b\}$.

The idea of this procedure is to use the MLE estimate $\hat{\theta}$ of the sample $x = \{x_1, \dots, x_n\}$ as a parameter for $f(x|\hat{\theta})$ to generate n new bootstrap samples. The way the functional form of $f(x|\theta)$ is used in combination with the MLE estimate $\hat{\theta}$, to generate new samples, is called *parametric bootstrap*. Then, for $b \in \{1, \dots, B\}$ bootstrap samples, new MLE $\hat{\theta}^b$ are calculated. From this, the sampling distribution of θ_n can be estimated and then used to derive the desired confidence interval.

Problem 6.7 Suppose that we have a coin with a probability for head (1) of θ and a probability for tail (0) that is $1 - \theta$. Hence, this model generates random variables x , from $\{0, 1\}$, with a Bernoulli distribution $f(x|\theta) = \theta^x(1 - \theta)^{1-x}$. What is the bootstrap 95% confidence interval of $\hat{\theta}$?

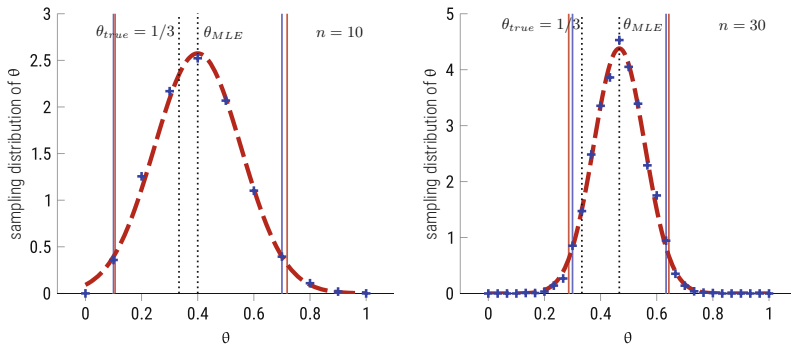


Fig. 6.9 Sampling distribution of θ_n . The blue crosses correspond to the estimates from the bootstrap approach, and the red dashed curves give the asymptotic distribution of the MLE. The vertical lines in red and blue correspond to the 95% confidence intervals for the bootstrap and the asymptotic distribution, respectively

In Fig. 6.9, we show estimates of the sampling distribution of θ_n . The blue crosses correspond to the results from bootstrap estimates using the above procedure, and the red dashed curve is the asymptotic distribution given by $\mathcal{N}\left(\hat{\theta}, \left(n\theta(1-\theta)\right)^{-1}\right)$ using the Fisher information in Eq. 6.79. We can see that for $n = 10$ and $n = 30$, the approximate bootstrap estimates and the corresponding 95% confidence intervals are quite similar to the results from the asymptotic distribution, although these sample sizes are only moderately large.

6.4.3 Meaning of Confidence Intervals

It is worth discussing the meaning of confidence intervals for MLE in details as these are frequently misinterpreted. First of all, a $(1-\alpha)$ confidence interval does *not* mean that an MLE for θ lies within a percentage of $100 \times (1-\alpha)$ in this interval. If one would like to have such an interpretation, one needs to use a Bayesian approach and credible intervals, as discussed in Sect. 6.3.4. Instead, a $(1-\alpha)$ confidence interval for an MLE for θ means that, if we repeat the same experiment many times, then, on average, in $100 \times (1-\alpha)$ percent of the cases, a $(1-\alpha)$ confidence interval will include the MLE for θ .

The rationale behind this interpretation is as follows. To know the probability for an MLE of θ to lie within a certain confidence interval, we need to consider this MLE as a random variable, and we can only make probabilistic statements about random variables. Since the MLE is a random variable, then we need to define its prior probability, which brings us directly into a Bayesian framework. The problem is that using a maximum likelihood framework, we assume that there is an unknown

but fixed value of θ (the parameter of the distribution from which the data are sampled) instead of a probability distribution.

6.5 Expectation-Maximization Algorithm

The expectation-maximization (EM) algorithm is an iterative method to estimate a parameter, or a vector of parameters θ , of a parametric probability distribution while attempting to maximize the associated likelihood function.

Although the EM has been widely popularized by Dempster, Laird, and Rubin [99], similar approaches can be traced back to the work of Newcomb [356] in 1886 and McKendrick [327] in 1926.

The problem of interest can be formulated as follows. Let Y denote a n -dimensional random real variable (i.e., $Y \in \mathbb{R}^n$) with a probability density function $f_Y(y|\theta)$, where θ is a parameter or a vector of parameters to be estimated. For a given realization y of Y , we would like to estimate the value of θ , denoted θ^* , which maximizes the likelihood function of θ , given by

$$L(\theta) = f_Y(y|\theta). \quad (6.80)$$

This yields the following optimization problem:

$$\theta^* = \arg \max_{\theta \in \Theta} L(\theta), \quad (6.81)$$

where Θ is the set of all potential values of θ and L is defined in Eq. 6.80.

In theory, various techniques can be used to solve the above optimization problem. However, it is often difficult to define the topology of the set of potential parameter values, Θ . Therefore, it is customary to resort to some heuristic methods, and the EM algorithm is one of the most commonly used to estimate solutions to the problem in Eq. (6.81).

The EM assumes that there exists another random real variable $X \in \mathbb{R}^m$ with a probability density function $f_X(x|\theta)$, such that for a given realization x of X , it is computationally easier to find the value of θ , which maximizes the likelihood function of θ given by

$$L_x(\theta) = f_X(x|\theta). \quad (6.82)$$

In this case, the optimization problem of interest is therefore

$$\theta^* = \arg \max_{\theta \in \Theta} f_X(x|\theta). \quad (6.83)$$

Since $\log()$ is a monotonically increasing function, then a value of θ , which maximizes $L_x(\theta)$, also maximizes $\log L_x(\theta) = \log f_X(x|\theta)$. Often, it is easier to find the parameter θ that maximizes the log-likelihood $\log f_X(x|\theta)$.

In the EM approach, the vector y , which represents the observed data or the data at hand, is referred to as the *incomplete* data, whereas the vector x , which represents the “ideal” data (which will facilitate the estimation of the parameter θ) we wish we had, is called the *complete* data.

For a given estimate θ^{k-1} of θ and the data y , the EM algorithm estimates $\log L_x(\theta)$ as the conditional expected value of the random function $\log f_X(X|\theta)$, conditioned on y and θ^{k-1} as follows:

$$\mathbb{E}_{X|y, \theta^{k-1}}(\log f_X(X|\theta)) = \int_{\Omega_y} f_{X|Y}(x|y, \theta^{k-1}) \log f_X(x|\theta) dx, \quad (6.84)$$

where $f_{X|Y}(x|y, \theta^{k-1})$ is the conditional probability density function for the complete data x , given θ^{k-1} and y , whereas Ω_y is the closure of the set

$$\{x : f_{X|Y}(x|y, \theta^{k-1}) > 0\}.$$

This step of the EM algorithm is referred to as the *Expectation step*.

The only unknown in the right-hand side of Eq. 6.84 is θ . Thus, it can be defined as a function of θ , for a fixed given value θ^{k-1} , denoted $Q(\theta|\theta^{k-1})$, i.e.,

$$Q(\theta|\theta^{k-1}) = \int_{\Omega_y} f_{X|Y}(x|y, \theta^{k-1}) \log f_X(x|\theta) dx. \quad (6.85)$$

The *Maximization step* of the EM algorithm consists of finding the value of θ , which maximizes $Q(\theta|\theta^{k-1})$. The corresponding value of θ , denoted θ^k , is the “optimal” value of θ for the iteration k .

Therefore, the EM algorithm generates a sequence $\{\theta^k\}$ of estimates of θ , and during such process, it is expected that:

- (i) The sequence $\{L(\theta^k)\}$ is increasing, that is, $\forall k, L(\theta^{k+1}) \geq L(\theta^k)$.
- (ii) The sequence $\{L(\theta^k)\}$ converges to $L(\theta^*)$.
- (iii) The sequence $\{\theta^k\}$ converges to θ^* .

In the discrete case, we need to replace the probability density function with the probability mass function and replace the integral with the summation.

The EM algorithm can be summarized as follows:

Step 0: Initialization Let θ^0 denote an initial estimate of the parameter θ ; θ^0 is generally generated randomly.

Step 1: Expectation Step Given the observed (or “incomplete”) data y and θ^{k-1} the current estimate of the parameter θ , formulate the conditional probability density function $f_{X|Y}(x|y, \theta^{k-1})$ for the complete data X . Then, use the formulated

conditional probability to compute the conditional expected log-likelihood, as a function of θ , as follows:

$$Q(\theta|\theta^{k-1}) = \int_{\Omega_Y} f_{X|Y}(x|y, \theta^{k-1}) \log f_X(x|\theta) dx. \quad (6.86)$$

Step 2: Maximization Step Maximize the function $Q(\theta|\theta^{k-1})$ over $\theta \in \Theta$ to find

$$\theta^k = \arg \max_{\theta \in \Theta} (Q(\theta|\theta^{k-1})). \quad (6.87)$$

Step 3: Exit Condition If $\|\theta^k - \theta^{k-1}\| < \varepsilon$ or $\|L(\theta^k) - L(\theta^{k-1})\| < \varepsilon$, where $L(\theta)$ is defined in (6.80), for some $\varepsilon > 0$, then stop; otherwise, go to **Step 1**.

For a compiling discussion on the EM algorithm and its various extensions, we refer the reader to the textbook of McLachlan and Krishnan [328].

6.5.1 Example: EM Algorithm

In this section, we provide an illustrative example of the EM algorithm, introduced in [328] and further elaborated by Byrne [63]. Let W denote the non-negative random variable representing the time to failure of an item, which is assumed to be exponentially distributed, i.e.,

$$f(w|\theta) = \frac{1}{\theta} e^{-\frac{w}{\theta}}, \quad (6.88)$$

where the parameter $\theta > 0$ denotes the expected time to failure.

The associated cumulative probability distribution is given by

$$F(w|\theta) = 1 - e^{-\frac{w}{\theta}}. \quad (6.89)$$

Suppose that we observe a random sample of n items and let w_i , $i = 1, \dots, n$ denote their corresponding failure times. We would like to use these data to estimate the mean time to failure θ , which, in this case, is a scalar parameter. In practice, the study generally terminates before we observe the random sample $w_1, w_2, \dots, w_n$. This means that we can only record the time to failure for the first r items ($r < n$), which failed. For the $n - r$ items whose times to failure were not observed at the end of the study, we assume that their time to failure is T , the duration of the study. Such a data set is referred to as *censored*, and let denote it $y = (y_1, \dots, y_n)$, with

$$\begin{cases} y_i = w_i, & \text{for } i = 1, \dots, r; \\ y_i = T, & \text{for } i = r + 1, \dots, n; \end{cases}$$

The censored data y can be viewed as the incomplete data, whereas the completed data will be those obtained if the study terminated only when all the n items failed.

The probability of an item to survive for the duration of the study, T , is

$$1 - F(T|\theta) = 1 - 1 + e^{-\frac{T}{\theta}} \quad (6.90)$$

$$= e^{-\frac{T}{\theta}}. \quad (6.91)$$

Therefore, the likelihood or the probability density function of the vector of incomplete data, y , is

$$f_Y(y|\theta) = \left(\prod_{i=1}^r \frac{1}{\theta} e^{-\frac{y_i}{\theta}} \right) \left(\prod_{i=r+1}^n e^{-\frac{T}{\theta}} \right) \quad (6.92)$$

$$= \left(\prod_{i=1}^r \frac{1}{\theta} e^{-\frac{y_i}{\theta}} \right) e^{-(n-r)\frac{T}{\theta}}. \quad (6.93)$$

The log-likelihood function of y is

$$\log f_Y(y|\theta) = \log \left[\left(\prod_{i=1}^r \frac{1}{\theta} e^{-\frac{y_i}{\theta}} \right) e^{-(n-r)\frac{T}{\theta}} \right] \quad (6.94)$$

$$= -r \log \theta - \frac{1}{\theta} \left[\sum_{i=1}^r y_i + (n-r)T \right]. \quad (6.95)$$

In this particular example, the value of θ , which maximizes $\log f_Y(y|\theta)$, can be obtained analytically by solving

$$\frac{\partial \log f_Y(y|\theta)}{\partial \theta} = 0,$$

and the corresponding optimal value of θ , denoted $\hat{\theta}$, is given by

$$\hat{\theta} = \frac{1}{r} \left[\sum_{i=1}^r y_i + (n-r)T \right].$$

However, this is not generally the case when dealing with incomplete data.

Now, assume that the actual times to failure of the items, which did not fail during the study, are in fact known, and let's denote the corresponding (completed) data by

$x = (w_1, \dots, w_n)$. Then, the likelihood or the probability density function of the vector of complete data, x , is

$$f_X(x|\theta) = \prod_{i=1}^n \frac{1}{\theta} e^{-\frac{w_i}{\theta}}. \quad (6.96)$$

The log-likelihood function of x is

$$\log f_X(x|\theta) = -n \log \theta - \frac{1}{\theta} \sum_{i=1}^n w_i. \quad (6.97)$$

Again in this example, the value of θ , which maximizes $\log f_X(x|\theta)$, can be easily obtained analytically, solving

$$\frac{\partial \log f_X(x|\theta)}{\partial \theta} = 0,$$

and the corresponding optimal value of θ , denoted $\hat{\theta}$, is given by

$$\hat{\theta} = \frac{1}{n} \sum_{i=1}^n w_i.$$

In the subsequent section, we are going to apply the EM algorithm to this example. Assume that we are given the incomplete data, y , and we have the current estimate of the parameter θ , θ^{k-1} . We will illustrate below the Expectation and the Maximization steps of the EM algorithm.

Expectation Step

Note that $\log f_X(x|\theta)$ is linear in the unobserved data w_i , $i = r + 1, \dots, n$. Then, to calculate $\mathbb{E}_{X|y, \theta^{k-1}}(\log f_X(X|\theta))$, we just need to replace the unobserved values with their conditional expected values, given y and θ^k .

$$\mathbb{E}_{X|y, \theta^{k-1}}(\log f_X(X|\theta)) = -n \log \theta - \frac{1}{\theta} \left[\left(\sum_{i=1}^n y_i \right) + (n-r) \theta^{k-1} \right]. \quad (6.98)$$

Thus,

$$Q(\theta|\theta^{k-1}) = -n \log \theta - \frac{1}{\theta} \left[\left(\sum_{i=1}^n y_i \right) + (n-r) \theta^{k-1} \right]. \quad (6.99)$$

Maximization Step

The value of θ , which maximizes $Q(\theta|\theta^{k-1})$, is obtained by solving

$$\frac{\partial Q(\theta|\theta^{k-1})}{\partial \theta} = 0,$$

which yields

$$\theta^k = \frac{1}{n} \left[\left(\sum_{i=1}^n y_i \right) + (n - k)\theta^{k-1} \right]. \quad (6.100)$$

Exit Condition

If $\|\theta^k - \theta^{k-1}\| < \varepsilon$, where ε is a specified threshold, then the algorithm stops; otherwise, we iterate the *Expectation step* using the calculated value θ^k .

6.6 Summary

In this chapter, we discussed the basics of statistical inference. We started by discussing exploratory data analysis (EDA), which comprises descriptive statistics and visualization methods for quickly summarizing information contained in data. We have seen that descriptive statistics has a close relationship with sample estimators, and for this reason, we discussed the associated important properties.

The classical task of statistics is parameter estimation, for which two different conceptual frameworks exist. The first is Bayesian inference and the second is maximum likelihood estimation. We discussed both methodologies and emphasized the differences. Finally, we discussed the expectation-maximization (EM) algorithm as a practical way for finding the (local) maximum of a likelihood function or the maximum posteriori (maximum of the posterior distribution) of a statistical model using an iterative approach.

Learning Outcome 6: Statistical Inference

Statistical inference refers to the process of learning from a (small) data sample and making predictions about an underlying population from which data have been drawn. Due to the variety of ways to formulate this problem, there are many approaches which can be used to investigate the different aspects of the problem.

We would like to emphasize that the above description is essentially true for all fields dealing with the analysis of data, including machine learning, artificial intelligence, or pattern recognition. However, statistics was the first field that articulated and formulated this aim explicitly. Hence, the term statistical inference is usually connected to the field of statistics, although all data science-related fields utilize the same conceptual framework.

6.7 Exercises

1. Let $X_1, X_2, \dots, X_n$ be some independent random variables following the normal distribution $\mathcal{N}(\mu, \sigma^2)$. Show that the estimator of the variance, σ , given by

$$S^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2, \quad \text{where the random variable } \bar{X} = \frac{1}{n} \sum_{i=1}^n X_i,$$

is an unbiased estimator.

2. Suppose H_1 is a biased estimator of θ , with a small bias, but its MSE is considerably lower compared to the MSE of H_2 , which is an unbiased estimator of θ . Which estimator is preferred?
3. Let H be an unbiased estimator of θ . Show that the estimator $\eta(H)$ is generally not an unbiased estimator of $\eta(\theta)$, unless the function η is linear.
4. Show that the following probability distributions belong to the regular expression class (REC), and find the functions a, b, c , and d as well as a sufficient statistic for the parameter, based on a random sample of size n :
 - a. The binomial distribution $\mathcal{B}(n, \theta)$, where n is known;
 - b. The Poisson distribution with parameter θ ;
 - c. The geometric distribution with parameter θ , given by

$$f(x|\theta) = \theta(1-\theta)^x, \quad x = 0, 1, \dots$$

- d. The negative exponential distribution with parameter θ , given by

$$f(x|\theta) = \theta e^{-\theta x}, \quad x \geq 0, \quad \theta > 0$$

- e. the normal distribution $\mathcal{N}(\mu, \sigma^2)$, where σ^2 is known.
5. Let $X_1, X_2, \dots, X_n$ be some independent random variables with the probability distribution function, $f(x|\theta)$, from the regular exponential class (REC).
 - a. Using the Fisher-Neyman factorization theorem, show that the statistic $\sum_i b(X_i)$ is a sufficient statistic for any distribution in the REC.

- b. Suppose that the random variables $X_1, X_2, \dots, X_n$ are from the binomial distribution $\mathcal{B}(m, \theta)$, where m is known. Using the result from the previous question, determine a sufficient statistic for the parameter θ .
6. Repeat the analysis shown in Listing 6.2 and record the estimates for the maximum a posteriori (MAP). Why is the MAP a random variable? What is the cause of this randomness?

Chapter 7

Clustering



7.1 Introduction

The task of grouping data points or instances into clusters is quite fundamental for data science [214, 260]. In general, clustering methods belong to the area of *unsupervised learning* [220], because the data upon which such methods are based on are unlabeled, i.e., no information is available about the true cluster to which a data point belongs. The aim of clustering methods is to group a set of data points, which can correspond to a wide variety of different objects, e.g., texts, vectors, or networks, into groups that we call clusters.

There are many different approaches underlying clustering methods. However, in this chapter, we focus on clustering methods based on similarity and distance measures [104, 411]. Such methods provide criteria for thresholding the similarity (or distance) of data points for their assignment to groups. In this chapter, we discuss two main classes of clustering methods:

- Non-hierarchical clustering methods
- Hierarchical clustering methods

Furthermore, we discuss the difficult problem of cluster validation. Due to the fact that the data used for clustering are unlabeled, this is a challenging task, requiring further assumptions.

We start this chapter by motivating the task of clustering data. Then, we outline existing similarity and distance measures that are frequently used in data science. Afterward, we discuss important clustering methods [214, 260] and techniques for evaluating clustering solutions.

7.2 What Is Clustering?

Cluster analysis, also called clustering, relies only on data of the type $X = \{x_i\}_1^n$ without additional information, e.g., about class labels that would allow to associate individual data points x_i with specific classes or, in general, specific categories. Depending on the context, the data points x_i are sometimes also called feature vectors, profile vectors, or instances. In the following, we denote the number of available instances by n , i.e., $i \in \{1, \dots, n\}$. This corresponds also to the sample size. In contrast, we denote the length of the data points x_i by p .

Suppose there are $n = 300$ patients suffering from a tumor. From a biopsy of each tumor, a molecular profile for each of these tumors is generated by measuring the gene expression levels of $p = 10,000$ genes using, e.g., DNA microarrays. Similar data sets $X = \{x_i\}_1^n$ have been used to identify **clusters of tumors** according to their similarity. In this case, x_i corresponds to patient i , or her/his tumor, and each component of x_i corresponds to the expression level of a gene. There are different chip types available for DNA microarrays that allow to measure a different number of genes. For instance, there are chips that allow to measure the gene expression levels of $p = 20,000$ genes, for the same tumor biopsies. As one can see, in this case, the number of patients ($n = 300$) would not change of course, but the number of measurements for the genes. The above example allows to clearly see the asymmetry between p and n . For any analysis, it is important to be very clear about the meaning of p and n in a given context, which is problem-specific.

In order to complete our discussion, we need one further layer of argument. This relates to the question of identifying the **features** and the **instances** for a given data set. In fact, this is a definition that needs to be made by the analyst. For instance, using our example from above, if we want to **cluster tumors**, the tumors become the instances (n) and the genes the features (p). However, if we want to **cluster genes**, then the genes become the instances (n) and the tumors the features (p). Both are valid perspectives using the same data X [371]. Hence, the role of, e.g., a patient is not always the same, but it depends on the question the analyst wants to study.

Figure 7.1 summarizes the overall situation. It shows characteristics of the data type, principles of major clustering methods and lists some known clustering approaches.

7.3 Comparison of Data Points

For clustering methods, similarity and distance measures are crucial to group data points. Before we start outlining the various similarity and distance measures used in data science, we define such measures in general [46, 145].

Definition 7.1 Let X be a set and s a mapping $s : X \times X \longrightarrow [0, 1]$. $s(x, y)$ with $x, y \in X$ is called **similarity measure** if

Data type:

$X = \{(x_i)\}_1^n$ with $x_i \in \mathbb{R}^p$

x_i is called a data point, instance or feature vector

p : number of variables

n : number of samples

Question addressed:

Is there a 'structure' between variables?

Principles of major clustering approaches:

Partitioning-based clustering $\implies$ partitions X into K non-overlapping clusters based on a **distance measure, d , for data points** resulting in clusters $C_k = \{x_{1,k}, \dots, x_{N_k,k}\}$ such that $X = \cup_{k=1}^K C_k$, with $k \in \{1, \dots, K\}$ and N_k is the number of data points in cluster k .

Hierarchical clustering $\implies$ hierarchical partitioning of X resulting in nested clusters as function of a **distance measure, d , for clusters** called linkage function.

Fig. 7.1 Overview of cluster analysis with respect to the data type used, the question addressed, and the principles of major approaches

$$s(x, y) > 0 \quad (\text{positivity}). \quad (7.1)$$

$$s(x, y) = s(y, x) \quad (\text{symmetry}). \quad (7.2)$$

$$s(x, y) = 1 \quad \text{iff} \quad x = y \quad (\text{identity}). \quad (7.3)$$

A distance measure can be defined similarity.

Definition 7.2 Let X be a set and d a mapping $d : X \times X \longrightarrow \mathbb{R}_+$. $d(x, y)$ with $x, y \in X$ is called **distance measure** if

$$d(x, y) \geq 0 \quad (\text{positivity}). \quad (7.4)$$

$$d(x, y) = d(y, x) \quad (\text{symmetry}). \quad (7.5)$$

$$d(x, y) = 0 \quad \text{iff} \quad x = y \quad (\text{identity}). \quad (7.6)$$

If in addition the following holds:

$$d(x, z) \leq d(x, y) + d(y, z), \quad x, y, z \in X, \quad (7.7)$$

then we call $d(x, y)$ a **distance metric** [104].

The inequality in Eq. 7.7 is called the *triangular inequality* [104]. We would like to emphasize that the properties of distance measures are sufficient for constructing clustering methods. However, the triangular inequality and, hence, distance metrics

are required if one wants to define an embedded distance measure in a metrical space.

Let's study two examples that show how to correctly classify quantitative measures for real numbers.

Example 7.1 Let's define a quantitative similarity measure s between $x, y \in \mathbb{R}$ by

$$s(x, y) := \frac{1}{1 + |x - y|}. \quad (7.8)$$

Now, we show that all properties of Definition 7.1 hold.

Positivity: As $|x - y| \geq 0$, we see that $s(x, y) > 0$ and the positivity property is fulfilled.

Symmetry: As $|x - y| = |y - x|$, the symmetry property is also satisfied.

Identity: To see the identity property, we first show that

$$s(x, y) := \frac{1}{1 + |x - y|} = 1 \implies x = y.$$

Thus, we have to distinguish two cases when considering $|x - y|$.

The first case $x - y \geq 0$ leads to $\frac{1}{1+x-y} = 1$ and, finally, to $x = y$.

The second case $x - y < 0$ leads to $\frac{1}{1-(x-y)} = 1$. Again, this yields to $x = y$.

Now, it remains to show that if $x = y \implies s(x, y) := \frac{1}{1+|x-y|} = 1$. But if $x = y$, then $s(x, x) = 1$. Altogether, we see that the measure given by Eq. 7.8 is a similarity measure according to Definition 7.1.

Example 7.2 We define a quantitative distance measure between $x, y \in \mathbb{R}$ by

$$d(x, y) := |x - y|. \quad (7.9)$$

Let's show that all the properties in Definition 7.2 are fulfilled.

Positivity: The positivity property is fulfilled by definition of the modulus function.

Symmetry: The symmetry property follows directly because $|x - y| = |y - x|$.

Identity: To show that $d(x, y) := |x - y| = 0 \iff x = y$, we start with $d(x, y) := |x - y| = 0 \implies x = y$. Considering the two cases of $|x - y|$, we always obtain $x = y$ as this follows from $x - y = 0$ and $-(x - y) = 0$. The second direction follows immediately; $x = y$ leads to $d(x, y) = |x - x| = 0$.

Let's study a numerical example for the application of the measure from Example 7.1. Suppose that $x = 1$ and $y = 1.1$. We would expect that the similarity measure given by Eq. 7.8 yields a high similarity score close to one since the values of x and y are quite close to each other. Numerically, we obtain $s(1, 1.1) = 0.90909$. If we utilize the known distance-similarity relation [46]

$$d(x, y) = 1 - s(x, y), \quad s(x, y) \leq 1, \quad (7.10)$$

we get $d(1, 1.1) = 1 - s(1, 1.1) = 0.090910$. That means a high similarity value between x and y corresponds to a small distance between x and y .

7.3.1 Distance Measures

In this section, we discuss some specific distance measures [46, 214], which are widely used in data science. For all the discussed distance measures, we assume that the properties given in Definition 7.2 are satisfied.

Examples of commonly used distance measures for pairs of data points are:

$$d_E(x, y) = \sqrt{\sum_{i=1}^p (x_i - y_i)^2} \quad (\text{Euclidian distance}). \quad (7.11)$$

$$d_{\min}(x, y) = \left(\sum_{i=1}^p |x_i - y_i|^p \right)^{1/\alpha} \quad (\text{Minkowski distance}). \quad (7.12)$$

$$d_{\text{man}}(x, y) = \sum_{i=1}^p |x_i - y_i| \quad (\text{Manhattan distance}). \quad (7.13)$$

$$d_{\max}(x, y) = \max_i |x_i - y_i| \quad (\text{maximum distance}). \quad (7.14)$$

$$d_{\min}(x, y) = \min_i |x_i - y_i| \quad (\text{minimum distance}). \quad (7.15)$$

$$d_\rho(x, y) = \frac{1 - \rho_{x_i, y_j}}{2} \quad (\text{correlation distance}). \quad (7.16)$$

For the Minkowski distance, α is a positive integer value, which gives for $\alpha = 2$ the Euclidean distance.

We would like to point out that each of the above distance measures provides the minimum possible distance for $d(x, x)$, which is called the self-distance. In this case, $d(x, x) = 0$ for all possible data points x .

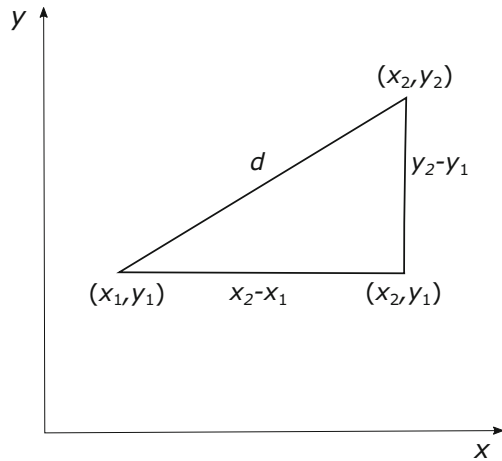
Let's illustrate the calculation of the Euclidian distance via an example. Figure 7.2 depicts two vectors $x = (x_1, x_2) \in \mathbb{R}^2$ and $y = (y_1, y_2) \in \mathbb{R}^2$. If we set $a = x_2 - x_1$ and $b = y_2 - y_1$, then using Pythagoras' theorem, we obtain

$$d^2 = a^2 + b^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2, \quad (7.17)$$

and

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}. \quad (7.18)$$

Fig. 7.2 Calculating the Euclidian distance between two-dimensional vectors, using Pythagoras' theorem



Listing 7.1 shows an application of the command `dist()` for calculating the Euclidian distance of two vectors using $\mathbb{R}^5$. By setting different options for “method,” one can obtain alternative distances for “maximum,” “Manhattan,” “Canberra,” or “Minkowski.”

Listing 7.1: Euclidian distance

```
x <- c(1, 1, 4, 1, 0)
y <- c(2, 7, 4, 6, 0)
length(x)
[1] 5
length(y)
[1] 5
dist(rbind(x, y), method="euclidean")
[1] 7.874008
```

7.3.2 Similarity Measures

In this section, we discuss some known similarity measures [46, 214] which are used for clustering methods. For all the discussed similarity measures, we assume that the properties given in Definition 7.1 are satisfied.

Examples of commonly used similarity measures for pairs of data points and sets are [214]:

$$s_D(X, Y) = \frac{2(|X \cap Y|)}{|X| + |Y|} \quad (\text{Dice's coefficient}). \quad (7.19)$$

$$s_J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} \quad (\text{Jaccard's coefficient}). \quad (7.20)$$

$$s_\rho(x, y) = \rho_{x_i, y_j} \quad (\text{correlation coefficient}). \quad (7.21)$$

$$s_{\cos}(x, y) = \frac{\sum_{i=1}^p x_i y_i}{\sqrt{\sum_{i=1}^p x_i^2} \sqrt{\sum_{i=1}^p y_i^2}} \quad (\text{cosine similarity}). \quad (7.22)$$

In Eqs. 7.19 and 7.20, X and Y are finite sets. The cosine similarity has been applied extensively in text mining and information retrieval; see [18]. We illustrate the calculation of Jaccard's coefficient using an example. Let X and Y be two text fragments given by the two sets

$$X = \{\text{Data, Science, is, challenging}\}, \quad (7.23)$$

$$Y = \{\text{Information, Science, is, modern}\}. \quad (7.24)$$

We obtain $X \cap Y = \{\text{Science, is}\}$ and, hence, $|X \cap Y| = 2$. Furthermore, $X \cup Y = \{\text{Data, Information, Science, is, challenging, modern}\}$ and $|X \cup Y| = 6$. Hence, the similarity between the two text fragments X and Y , measured by Jaccard's coefficient, equals $s_J(X, Y) = \frac{2}{6} = \frac{1}{3}$.

7.4 Basic Principle of Clustering Algorithms

There are two basic principles/properties of any clustering method, which have been proposed, in the literature, from a theoretical point of view; see [17, 46, 260]. Suppose that we start with a set of objects to be grouped using a clustering algorithm. First, the objects in a given generated cluster should be (very) similar among each other (*homogeneity*) [46, 260], with respect to a chosen similarity measure; see Sect. 7.3. Second, the objects that belong to different generated clusters should be (very) different from each other (*heterogeneity*) [46, 260]. We emphasize that Everitt et al. [154] call clusters, which fulfill these two properties, *natural clusters*. Various quantitative measures have been proposed to quantify homogeneity and heterogeneity; see [46].

One possible definition of homogeneity can be stated by the well-known homogeneity measure [46]

$$\text{Hom}(X) := \frac{2}{p(p-1)} \sum_{j=1}^p \sum_{i=1}^p d(x_i, x_j). \quad (7.25)$$

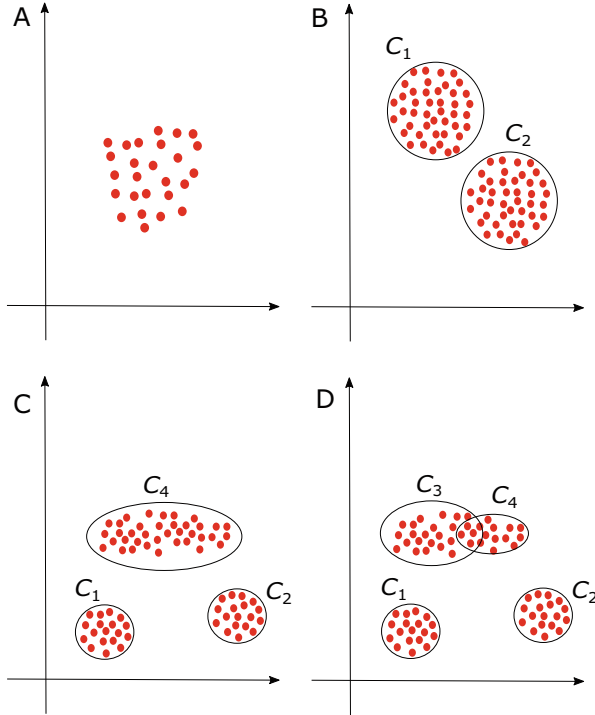


Fig. 7.3 Homogeneity vs. heterogeneity of a concrete clustering solution

For sets $X = \{x_1, x_2, \dots, x_p\}$ and $Y = \{y_1, y_2, \dots, y_p\}$ we call X more homogenous than Y if $\text{Hom}(X) < \text{Hom}(Y)$. In general, the smaller the value $\text{Hom}(X)$, the more homogenous the cluster according to Eq. 7.25.

We briefly illustrate this measure using an example. Let's assume that $X = \{1, 2, 3\}$, $Y = \{5, 7, 10\}$, and $d(x_i, x_j) = |x_i - x_j|$. First, we note that X and Y are non-overlapping and therefore appear heterogeneous. Equation 7.25 yields $\text{Hom}(X) = \frac{4}{3}$ and $\text{Hom}(Y) = \frac{10}{3}$, showing that $\text{Hom}(X) < \text{Hom}(Y)$. Based on our intuition, we also find that X is more homogenous than Y as the distances between the data points in X are smaller compared to Y .

Figure 7.3 explains the concept of homogeneity and heterogeneity visually; see [17]. In Fig. 7.3a, there is no structure in the given data sets, and, hence, no proper clusters can be generated. Two homogenous clusters C_1 and C_2 can be seen in Fig. 7.3b. Also, there is heterogeneity between C_1 and C_2 . In C, we see three clusters. Again, C_1 and C_2 are homogenous. However, C_4 forms a large cluster, and the homogeneity of this cluster is rather low. But there is heterogeneity between C_1 , C_2 , and C_4 . The last situation, in Fig. 7.3d, shows four clusters. C_1 and C_2 are homogenous, and the large cluster from Fig. 7.3c is now split into C_3 and C_4 . We now see in Fig. 7.3d that by generating C_3 and C_4 , the homogeneity property is fulfilled more properly compared to the old cluster C_4 in Fig. 7.3c. However, we

observe that C_3 and C_4 are not disjoint, and, therefore, we end up with overlapping clusters. Hence, the heterogeneity between C_3 and C_4 in Fig. 7.3d is not fulfilled.

Note that the two above-discussed properties, namely, homogeneity and heterogeneity, belong to the so-called *hard clustering* paradigm; see [260]. In this paradigm, data points are only allowed to be a member of one cluster.

The counterpart of hard clustering is *soft* or *fuzzy clustering* [260]. When using soft clustering, an object belongs to a cluster to a certain degree; this property is also referred to as *fuzzy membership*. Therefore, an object may belong to several clusters with a degree greater than zero.

7.5 Non-hierarchical Clustering Methods

After the above discussion on the preliminaries, we are now in a position to address clustering methods. The first important class of algorithms we discuss are non-hierarchical clustering methods, which are sometimes also called partition-based methods [46, 260].

7.5.1 *K-Means Clustering*

The *K*-means clustering method [320] is an iterative algorithm that requires as input the number of clusters K . The algorithm is initialized by randomly setting the cluster centers $\{m_k\}_{k=1}^K$. Then one assigns each data point, x_i , to exactly one cluster resulting in K sets of data points,

$$C(k) = \{x_i | x_i \text{ is in cluster } k\}, \quad (7.26)$$

that contain the data points for each cluster. Mathematically, this is accomplished by calculating the Euclidean distance between x_i and all centroids m_k and selecting the cluster with minimal Euclidean distance using

$$j = \underset{k}{\operatorname{argmin}} \{d_E(x_i, m_k)\}. \quad (7.27)$$

That means x_i will be assigned to cluster $C(j)$.

Then, the data points of the clusters, i.e., $C(k)$, are used to calculate updated centroids of the clusters, given by

$$m_k = \frac{1}{N_k} \sum_{j: x_j \in C(k)} x_j. \quad (7.28)$$

Here, N_k is the number of data points in $C(k)$, i.e., $N_k = |C(k)|$. The centroids are just the mean value of all samples and can be seen as a *representative* of a cluster. This completes the first iteration step. Then, all above steps are repeated leading to updated $C(k)$ and centroids.

In order to terminate the algorithm, one can either set a fixed number of iterations, I , or assess the progress made during the iterations. The latter implies that one needs a quality measure to assess the progress quantitatively. For this reason, the squared Euclidean distance

$$\text{ESS} = \sum_{k=1}^K \sum_{x_i \in C_k} (x_i - \bar{x}^k)^t (x_i - \bar{x}^k) \quad (7.29)$$

is used. The closer the samples are around the centroids of their respective cluster, the smaller the ESS. For example, by using a small $\varepsilon > 0$, one can terminate the iteration process if

$$\text{ESS}(i) - \text{ESS}(i + 1) > \varepsilon \quad (7.30)$$

does no longer hold. The major steps in the implementation of the K -means are given by Algorithm 1.

Algorithm 1: K -means clustering

```

1: input:  $n$  data points  $\{x_i\}_{i=1}^n$  with  $x_i \in \mathbb{R}^P$ 
2: input:  $K$  - number of clusters
3: initialize the cluster centroids  $\{m_k\}_{k=1}^K$  with  $m_k \in \mathbb{R}^P$  randomly
4: repeat
5:   for  $i \leftarrow 1$  to  $n$  do
6:      $j = \text{argmin}_k \{d_E(x_i, m_k)\}$ 
7:      $x_i \rightarrow C(j)$  - assign to closest cluster
8:   end for
9:   for  $k \leftarrow 1$  to  $K$  do
10:    define  $N_k = |C(k)|$ 
11:     $m_k = \frac{1}{N_k} \sum_{j: x_j \in C(k)} x_j$ 
12:   end for
13: until termination criteria
14: output: return clusters  $\{C(k)\}_{k=1}^K$ 

```

An obvious disadvantage of k -means is that K must be given to start the algorithm. However, K is generally not known and can also be in the eye of a beholder requiring special domain knowledge. Also, K -means is sensitive to outliers that could disrupt a natural clustering structure. Another drawback relates to the initial choice of the centroids which have a strong impact on the expected clustering solution. Therefore, a global minimum of the objective function cannot be guaranteed with only one starting configuration of the initial centroids. To overcome

this problem, one can run K -means multiple times by using different choices of initial centroids.

Algorithm 2: K-medoids clustering

```

1: input:  $n$  data points  $\{x_i\}_{i=1}^n$  with  $x_i \in \mathbb{R}^P$ 
2: input:  $K$  - number of clusters
3: initialize the cluster medoids  $\{m_i\}_{i=1}^K$  by randomly selecting  $K$  data points from  $\{x_i\}_{i=1}^n$ 
4: repeat
5:   for  $i \leftarrow 1$  to  $n$  do
6:      $j = \operatorname{argmin}_k \{d(x_i, m_k)\}$ 
7:      $x_i \rightarrow C(j)$  - assign to closest cluster
8:   end for
9:   for  $k \leftarrow 1$  to  $K$  do
10:    for  $i : x_i \in C(k)$  do
11:       $D(i) = \sum_{j: x_j \in C(k)} d(x_i, x_j)$ 
12:    end for
13:     $l = \operatorname{argmin}_{i: x_i \in C(k)} \{D(i)\}$ 
14:     $m_k = x_l$ 
15:  end for
16: until termination criteria
17: output: return clusters  $\{C(k)\}_{k=1}^K$ 

```

7.5.2 K -Medoids Clustering

We can generalize the K -means algorithm by making two modifications [260]. First, instead of using centroids to represent clusters, one can use medoids [275]. Second, instead of the Euclidean distance, one can use any other distance measure defined in Sect. 7.3.2.

In contrast to a centroid, which is the mean of all data points that belong to a cluster, a medoid corresponds to one of the data points within a cluster itself. That means a medoid does not need to be estimated by any measure, e.g., the mean, but just needs to be selected among all data points in a cluster.

In order to select such a medoid, a criteria is used that is based on the distances between data points within the cluster. Specifically, a medoid for cluster k is defined as the data point x_i , with $x_i \in C(k)$, which has the minimal distance to all other data points in $C(k)$,

$$D(i) = \sum_{j: x_j \in C(k)} d(x_i, x_j). \quad (7.31)$$

In Algorithm 2, we highlight this part of the algorithm in orange.

The generalization from centroids to medoids has its price because the identification of the K medoids is far more computationally demanding than the estimation of the K centroids.

7.5.3 Partitioning Around Medoids (PAM)

There is a further variation of the K -means clustering algorithm called partitioning around medoids (PAM); see [260]. The basic steps of PAM are shown in Algorithm 3. PAM assigns in its first step data points x_i to their closest medoids (red part in Algorithm 3). Then, these clusters are quantified by measuring the distance between all data points and their corresponding medoids. In Algorithm 3, this measure is denoted by Q . Then, for all medoid-data point pair combinations, a swapping of x_i with m_k is assessed by calculating the resulting quality measure Q'_{ki} . Finally, the medoid-data point pair that leads to the maximal reduction in $Q - Q'_{ki}$, corresponding to the largest reduction in the distances between all data points and medoids, is selected.

Algorithm 3: Partitioning around medoids (PAM) clustering

```

1: input:  $n$  data points  $\{x_i\}_{i=1}^n$  with  $x_i \in \mathbb{R}^P$ 
2: input:  $K$  - number of clusters
3: initialize the cluster medoids  $\{m_k\}_{k=1}^K$  by randomly selecting  $K$  data points from  $\{x_i\}_{i=1}^n$ 
4: repeat
5:   for  $i \leftarrow 1$  to  $n$  do
6:      $j = \operatorname{argmin}_k \{d(x_i, m_k)\}$ 
7:      $x_i \rightarrow C(j)$  - assign to closest cluster
8:   end for
9:    $Q = \sum_{k=1}^K \sum_{x_i \in C(k)} d(x_i, m_k)$ 
10:  for  $k \leftarrow 1$  to  $K$  do
11:    for  $i : x_i \in C(k)$  do
12:      swap  $x_i$  with  $m_k$  ( $x_i \leftrightarrow m_k$ )
13:       $Q'_{ki} = \sum_{k=1}^K \sum_{x_i \in C(k)} d(x_i, m_k)$ 
14:    end for
15:  end for
16:   $[k', i'] = \operatorname{argmax}_{k,i} \{Q - Q'_{ki}\}$ 
17:   $m'_{k'} = x'_{i'}$ 
18:   $x'_{i'} = m'_{k'}$ 
19: until termination criteria
20: output: return clusters  $\{C(k)\}_{k=1}^K$ 

```

7.6 Hierarchical Clustering

The second class of clustering methods we discuss are called hierarchical clustering. Hierarchical clustering algorithms are among the most popular clustering approaches [46, 260]. There is a large variety of different procedures that can be distinguished by the distance measure they are using. Furthermore, all of these methods perform either (1) an agglomerative (bottom-up) or (2) a divisive (top-down) clustering.

Suppose that we have n data points we want to cluster. *Agglomerative algorithms* start with n clusters where each cluster consists of exactly one data point. Then, the distances between all n clusters are evaluated, and the two “closest” clusters are merged, resulting in $n - 1$ clusters. This successive merging of two clusters is iteratively repeated until all clusters are merged into a single cluster. In Algorithm 4, we summarize the principal algorithmic steps. The distance function D can be any of the particular linkage functions, defined in Sect. 7.6.3.

In contrast, *divisive algorithms* start with just *one* cluster containing all n data points. Then this large cluster is successively split down into more clusters until one ends up with n clusters, each containing just one data point.

To perform the corresponding merging or splitting steps for the agglomerative and divisive algorithms, appropriate distance measures need to be used to decide what “close” clusters means. We discuss these distance measures in Sect. 7.6.3. In the following, we focus on agglomerative algorithms, because they are usually computationally more efficient and less restrictive.

Algorithm 4: Agglomerative Hierarchical Clustering

```

1: input:  $n$  data points  $\{x_i\}_{i=1}^n$  with  $x_i \in \mathbb{R}^p$ 
2: initialize  $n$  clusters,  $\{C(k)\}_{k=1}^n$ , by placing exactly one data point  $\{x_i\}_{i=1}^n$  in each randomly
3: for  $i \leftarrow 1$  to  $n - 1$  do
4:   for  $k \leftarrow 1$  to  $n - 1$  do
5:     for  $k' \leftarrow i + 1$  to  $n$  do
6:        $L_{k,k'} = D(C(k), C(k'))$ 
7:     end for
8:   end for
9:   merge cluster  $C(k)$  and  $C(k')$  with  $[k, k'] = \operatorname{argmin}\{L_{k,k'}\}$ 
10:   $h_{C_i} = [k, k', L_{k,k'}]$  store
11: end for
12: output: return hierarchical clusters  $h_C$ 

```

7.6.1 Dendrograms

Graphically, the result of an agglomerative algorithm (but also of a divisive algorithm) can be represented as a *dendrogram*. In Fig. 7.4, we show two examples of dendrograms. In general, a dendrogram is similar to a tree containing branches

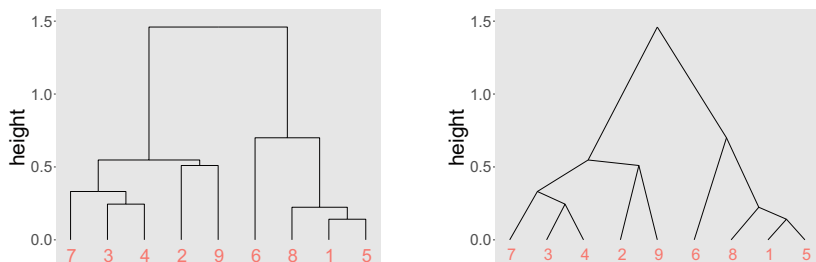


Fig. 7.4 Two dendrograms corresponding to the same result of an agglomerative clustering algorithm

that correspond to the clusters. However, it is important to note that in contrast to an ordinary tree, a dendrogram contains a *scale*, namely, its height. For this reason, on the left-hand side of both dendrograms, an axis that corresponds to the height of the branches is shown. Below we will see that these heights are related to the distances between the clusters.

There are various ways to visualize a dendrogram. One is a “rectangular” displaying of the branches (left), and the other is “triangular” (right). Despite the different visual representations, both dendrograms contain exactly the same amount of information about the clustering of the *nine* data points in Fig. 7.4. For this reason, it is merely a matter of personal taste which representation form to choose.

7.6.2 Two Types of Dissimilarity Measures

One needs to distinguish between two different types of distance measures. The first distance measure assesses directly the distance between two data points, whereas the second measure evaluates the distance between clusters, containing data points. That means one needs to distinguish between distance measures for:

- Pairs of data points (see Sect. 7.3),
- Pairs of clusters.

This implies that the former measures can be written as $d(x, y)$, where x and y are two data points of length p ; see Sect. 7.3. The latter can be expressed as $d(C(m), C(n))$, where $C(m)$ and $C(n)$ are two clusters that contain the two data points, i.e., $x \in C(m)$ and $y \in C(n)$. Distance measures between clusters will be defined in the next section, because they are required by hierarchical clustering methods.

7.6.3 Linkage Functions for Agglomerative Clustering

It is often unclear how to determine the distance between clusters. Cluster distances are needed to generate a dendrogram and, finally, to generate a clustering solution by cutting the dendrogram horizontally. Agglomerative clustering algorithms are distinguished from each other depending on the cluster distance measure they are using. However, most of these cluster distance measures, with the exception of the Ward distance [492], are based on the data point distances given by $d(x_i, x_j)$.

In general, a cluster distance measure, D , is called a *linkage function*. In the following, we give examples for five widely used linkage functions:

$$D_{\text{single}}(C_i, C_j) = \min d(x, y) \text{ with } x \in C_i \text{ and } y \in C_j \text{ (single linkage).} \quad (7.32)$$

$$D_{\text{comp}}(C_i, C_j) = \max d(x, y) \text{ with } x \in C_i \text{ and } y \in C_j \text{ (complete linkage).} \quad (7.33)$$

$$D_{\text{ave}}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i \text{ and } y \in C_j} d(x, y) \text{ (average linkage).} \quad (7.34)$$

$$D_{\text{ward}}(C_i, C_j) = \frac{|C_i||C_j|}{|C_i| + |C_j|} \|\mu_i - \mu_j\|^2 \text{ (Ward method).} \quad (7.35)$$

In $D_{\text{ward}}(C_i, C_j)$, μ_i and μ_j are the centers of the cluster C_i and C_j , respectively. Using the above measures, one can calculate the distance between clusters. These are crucial to generate the dendrogram and to generate plausible clustering solutions. The final clustering and the quality of clusters (e.g., measured by homogeneity) strongly depend on the linkage function that needs to be chosen in advance.

7.6.4 Example

In Fig. 7.5, we show four examples of hierarchical clustering using the four distance measures discussed in the previous section. The clustering analysis in Listing 7.2 is performed for the *Iris* data set that provides measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of *Iris*. The species are *Iris setosa*, *Iris versicolor*, and *Iris virginica*. For our analysis, we used only a subset of data consisting of *five* flowers from each species.

From Fig. 7.5, it is interesting to see that each of the four linkage functions gives different results. Only single linkage gives the “correct” clusters as known from the three *Iris* species.

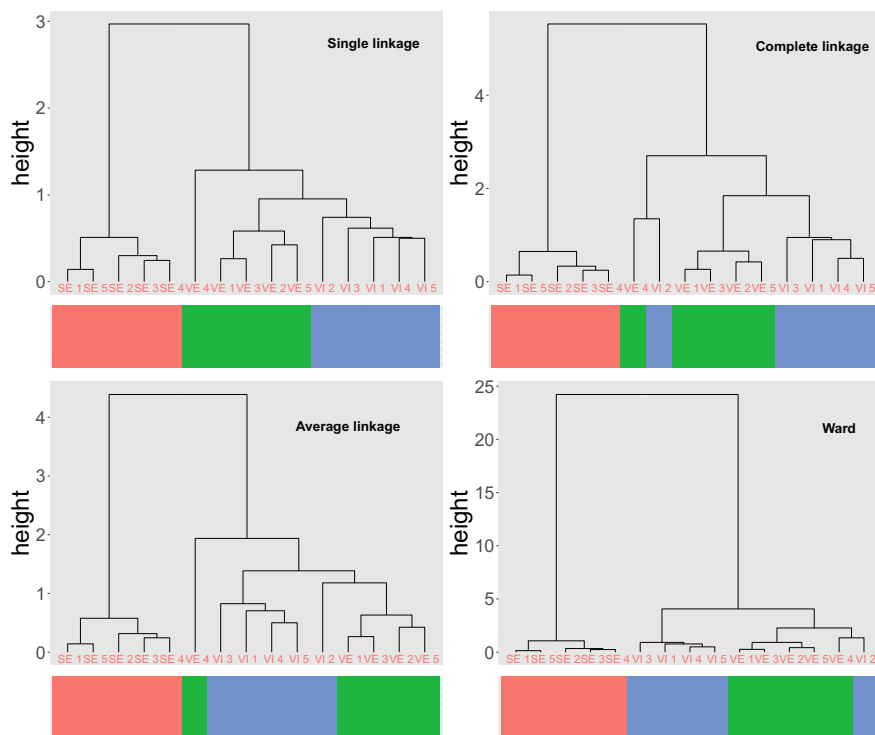


Fig. 7.5 Analysis of the *Iris* data set using the Euclidian distance measure. The following cluster distance measures were used. First row: single linkage (left) and complete linkage (right). Second row: average linkage (left) and Ward (right)

Listing 7.2: Hierarchical clustering; see Figure 7.5

```
data(iris)
n1 <- 5
n2 <- 5
n3 <- 5
n <- n1 + n2 + n3
dat <- matrix(0, nrow=n, ncol=4)
dat[1:n1,] <- as.matrix(iris[1:n1,1:4])
dat[(n1+1):(n1+n2),] <- as.matrix(iris[(50+1):(50+n2),1:4])
dat[(n1+n2+1):(n1+n2+n3),] <-
  as.matrix(iris[(100+1):(100+n3),1:4])
rownames(dat) <- c(paste("SE", 1:n1), paste("VE", 1:n2),
  paste("VI", 1:n3))

d <- dist(dat, method="euclidean")
hc <- hclust(d, method="single")
```

7.7 Defining Feature Vectors for General Objects

So far, we described partition-based clustering and hierarchical clustering algorithms for data sets of the type $X = \{x_i\}_1^n$ with $x_i \in \mathbb{R}^p$. That means feature vectors are given as an input for the clustering method. Furthermore, we mentioned briefly set-based measures, e.g., Jaccard’s and Dice’s coefficients, to show that clustering is more flexible with respect to the required representation of data points.

Now, we go one step further by generalizing these requirements. That means it is not only possible to cluster objects represented by vectors or sets but general objects. In Fig. 7.6, we show a visualization of this. On the left-hand side, two examples are shown for different types of objects: graph and document. Regardless of the nature of such objects, it is always possible to map these to feature vectors using domain-specific quantifications. For documents, this could correspond to features like TF-IDF (term frequency-inverse document frequency), POS (part of speech), or WE (word embeddings); see Chap. 5. Similarly, such a mapping is possible for graphs, and we discuss below some quantitative features corresponding to topological indices and graph entropy measures. Hence, by mapping an (abstract)

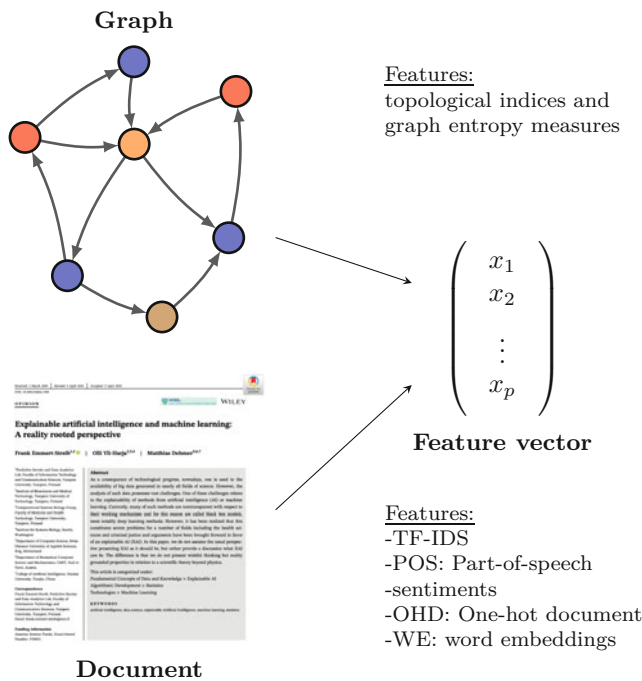


Fig. 7.6 Mapping of general objects to feature vectors. The left-hand side shows two examples for different types of objects: graph and document. These objects are then mapped by domain-specific quantifications leading to features

object to a feature vector, one can obtain an approximation of the properties of the object itself.

Now, we describe how a clustering can be performed for graphs or complex networks [135, 215]. One reason why we choose networks is the fact that graphs/networks are currently ubiquitous in data science and related disciplines. For instance, they have been applied for classification and modeling tasks extensively; see, e.g., [84, 94, 255, 357, 465].

In order to cluster networks, we need to transform a network $G = (V, E)$ into a vector $\mathcal{V} = (I_1, I_2, \dots, I_n)$. In the simplest case, $I_j : \mathcal{G} \rightarrow \mathbb{R}_+$, $1 \leq j \leq n$ is a topological index capturing structural information of a network. $\mathcal{F}$ is a class of graphs. For applications, many of these indices have been used so far [96, 103, 135], where a so-called graph invariant is required. A graph invariant is a graph measure (or index) that is invariant under isomorphism [94]. In the following, we give some *topological indices*, which have been applied to characterize graphs in a wide range of applications [94, 103, 109]:

$$W := \frac{1}{2} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} d(v_i, v_j) \quad (\text{Wiener index}), \quad (7.36)$$

where $d(v_i, v_j)$ denotes the shortest distance between v_i and v_j .

$$Z_1 := \sum_{i=1}^{|V|} \delta(v_i) \quad (\text{first Zagreb index}), \quad (7.37)$$

where $\delta(v_i)$ is the degree of the vertex v_i .

$$R := \sum_{(v_i, v_j) \in E} [\delta(v_i) \delta(v_j)]^{-\frac{1}{2}} \quad (\text{Randić index}). \quad (7.38)$$

$$B := \frac{|E|}{\mu + 1} \sum_{(v_i, v_j) \in E} [DS_i DS_j]^{-\frac{1}{2}} \quad (\text{Balaban index}), \quad (7.39)$$

where DS_i denotes the distance sum (row sum) of v_i and μ is the cyclomatic number (i.e., the number of rings in the graph).

We end this section by introducing measures for *graph entropy*, which turned out to be very meaningful for the quantitative characterization of graphs [47, 92, 96, 345]:

$$I_a := - \sum_{i=1}^k \frac{|N_i|}{|V|} \log \left(\frac{|N_i|}{|V|} \right) \quad (\text{topological information content}), \quad (7.40)$$

where $|N_i|$ stands for the number of topologically equivalent vertices in the i -th vertex orbit of G , with k is the number of different orbits.

$$I_D := -\frac{1}{|V|} \log \left(\frac{1}{|V|} \right) - \sum_{i=1}^{\rho(G)} \frac{2k_i}{|V|^2} \log \left(\frac{2k_i}{|V|^2} \right) \quad (\text{magnitude-based information index}), \quad (7.41)$$

where the distance of a value i in the distance matrix D appears $2k_i$ times. $\rho(G)$ stands for the diameter of a graph G .

Finally, the graph entropy, based on vertex functionals, was defined as follows [92, 96]:

$$I_f := -\sum_{i=1}^{|V|} \frac{f(v_i)}{\sum_{j=1}^{|V|} f(v_j)} \log \left(\frac{f(v_i)}{\sum_{j=1}^{|V|} f(v_j)} \right), \quad (7.42)$$

where $f : V \rightarrow \mathbb{R}_+$ and the vertex probabilities are

$$p(v_i) := \frac{f(v_i)}{\sum_{j=1}^{|V|} f(v_j)}. \quad (7.43)$$

Applications of the graph entropy can be found in bioinformatics, systems biology, and computer science, in general, for tackling classification, clustering, and modeling tasks; see, e.g., [133, 291, 347].

Above, we mentioned that a graph (or network) $G = (V, E)$ can be characterized by a vector $\mathcal{V}$. For instance, we could define feature vectors by $\mathcal{V}_1 = (W, R, B)$ or $\mathcal{V}_2 = (I_D, I_a, I_f, W, Z_1)$ or any other combination of graph measures for performing clustering. Altogether, this allows us to define numerical vectors, which can be used to apply the clustering techniques discussed in the previous sections. Lastly, we would like to remark that the above measures and many more have been implemented in the R package QuACN [348, 349].

7.8 Cluster Validation

The validation of clusters is a challenging task because often one does not have any prior or domain knowledge for judging the results of a clustering [260]. This is also directly visible from the unlabeled data upon which the clustering is based on. Therefore, various measures have been developed to quantify and assess the validity of clusters. In this section, we distinguish two major categories of such measures [260]: external criteria and internal criteria.

7.8.1 External Criteria

In this case, the result of a clustering is assessed using additional, *external* information that has not been used for the cluster analysis itself. This information consists in labels for the data points defining the gold standard of comparison. Depending on the origin of these labels, this can be the correct solution for the problem or just the best assignment available, e.g., provided by human experts. An example for the latter could be a biomedical data set containing measurements for tumor samples of patients for which the labels are provided by a pathologist performing a histological analysis of the morphology of the tumor tissues. Overall, this means labeled data are available; however, the labels have not been used for learning the clustering algorithm. This allows to assess clustering in the same way as a (multiclass) classification problem.

Let's assume that a cluster analysis results in a partitioning of n data points given by $C = \{C_1, \dots, C_K\}$ where K is the total number of clusters. That means, for each cluster, C_m is a set consisting of the data points that belong to this cluster, i.e., $C_m = \{x_i\}$. Furthermore, let's denote the reference information by $R = \{R_1, \dots, R_L\}$ with a possibly different number of clusters, i.e., $K \neq L$.

Utilizing R , it is now possible to decide if, e.g., the *two* data points, x_i and x_j , are correctly or incorrectly placed in the same cluster. Specifically, if $x_i, x_j \in R_m$ and $x_i, x_j \in C_n$, we call this pair a true positive (TP). Similarly, we define the following:

- If $x_i \in R_m, x_j \in R_{m'}$ and $x_i \in C_n, x_j \in C_{n'}$, we call this pair true negative (TN).
- If $x_i \in R_m, x_j \in R_{m'}$ and $x_i, x_j \in C_n$, we call this pair false positive (FP).
- If $x_i, x_j \in R_m$ and $x_i \in C_n, x_j \in C_{n'}$, we call this pair false negative (FN).

Performing such a pairwise data point comparison enables us to identify the total number of true positive, false positive, true negative, and false negative pairs. As a consequence, any statistical measure, based on these four errors (discussed in Chap. 3), can be used. However, in the context of cluster analysis, the following indices are frequently used [213, 260]:

Rand Index It is defined by

$$R = \frac{TP + TN}{TP + TN + FP + FN}. \quad (7.44)$$

Jaccard Index It is defined by

$$R = \frac{TP}{TP + FP + FN}. \quad (7.45)$$

F-Score It is defined by

$$F = \frac{TP}{TP + FP + FN} = \frac{PR}{P + R}, \quad (7.46)$$

where P is the precision and R the recall (sensitivity) given by

$$P = \frac{TP}{TP + FP}, \quad (7.47)$$

$$R = \frac{TP}{TP + FN}. \quad (7.48)$$

Fowlkes-Mallows (FM) Index It is defined by

$$FM = \sqrt{\frac{TP}{TP + FP}} \sqrt{\frac{TP}{TP + FN}} = \sqrt{P} \sqrt{R}. \quad (7.49)$$

The FM index is the geometric mean of the precision and recall P and R , while the F-score is their harmonic mean.

Normalized Mutual Information The NMI is defined by

$$NMI = \frac{I(C, R)}{\max\{H(C), H(R)\}}, \quad (7.50)$$

where $I(C, R)$ is the mutual information between the predicted set (C) and the reference set (R) and $H(R)$ and $H(C)$ are the entropies of these sets.

To evaluate the information theoretic entities, we need to estimate marginal and joint probability distributions based on the comparison of R with C . Table 7.1 shows a contingency table obtained from such a comparison. Here, the ordering of the sets C_i and R_j is not crucial because every pair is assessed. For instance, n_{11} gives the number of data points common in R_1 and C_1 , and a_1 is the number of all data points in R_1 , i.e., $a_1 = \sum_j n_{1j}$.

Table 7.1 Contingency table to evaluate the normalized mutual information

Reference	Predicted				
	C_1	C_2	$\dots$	C_K	Sums
R_1	n_{11}	n_{12}	$\dots$	$n_{1,K}$	a_1
R_2	n_{21}	n_{22}	$\dots$	$n_{2,K}$	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
R_L	$n_{L,1}$	$n_{L,2}$	$\dots$	$n_{L,K}$	a_L
Sums	b_1	b_2	$\dots$	b_K	$\sum_{ij} n_{ij} = N$

From the contingency table, we can estimate the following necessary probability distributions:

$$p(C_i) = \frac{b_i}{N}, \quad (7.51)$$

$$p(R_j) = \frac{a_j}{N}, \quad (7.52)$$

$$p(C_i, R_j) = \frac{n_{ij}}{N}, \quad (7.53)$$

which are then used to estimate the mutual information and the entropies as follows:

$$H(C) = - \sum_{i=1}^K p(C_i) \log p(C_i), \quad (7.54)$$

$$H(R) = - \sum_{j=1}^L p(R_j) \log p(R_j), \quad (7.55)$$

$$I(C, R) = \sum_{i=1}^K \sum_{j=1}^L p(C_i, R_j) \log \left(\frac{p(C_i, R_j)}{p(C_i)p(R_j)} \right). \quad (7.56)$$

7.8.2 Assessing the Numerical Values of Indices

After obtaining a numerical value of any of the above indices, the next question is what does it mean? Specifically, is the result from our cluster analysis “good” enough compared to the reference set R or not? This question is actually not easy to answer as it requires further considerations.

One way to show that the obtained clustering C is meaningful compared to R is via a hypothesis test. We would like to note that we can use any of the above indices as *test statistic* for a hypothesis test. However, since we usually do not know the analytical form of the sampling distribution that belong to a selected test statistic, we need to obtain the sampling distribution for the null hypothesis numerically.

7.8.3 Internal Criteria

When no external information about the partitioning of the data points within a data set is available in the form of a reference set R , then we need to assess the obtained clustering, C , using other criteria. In this case, the assessment of the quality of a cluster analysis is a more challenging task. Well-known criteria to

evaluate the quality of a clustering include the Dunn index, the Davies-Bouldin index, and the silhouette index. Most of these indices are based on concepts such as the homogeneity and the variance of the clustering. For instance, we say a cluster is homogenous if there are small distance and variance between the points in this cluster. We provide below the definition of the aforementioned indices:

Dunn Index Assume that the distance between two cluster C_m and C_n is given by

$$d(C_m, C_n) = \min_{x \in C_m, y \in C_n} d(x, y), \quad (7.57)$$

for any distance measure defined in Sect. 7.8.1.

The diameter of a cluster C_k is defined by

$$\text{diam}(C_k) = \max_{x, y \in C_k} d(x, y). \quad (7.58)$$

The diameter is just the maximum distance of any two data points in a cluster C_k .

For a fixed number K of clusters, the Dunn index is

$$D_K = \min_{m \in \{1, \dots, K\}} \min_{n \in \{m+1, \dots, K\}} \left(\frac{d(C_m, C_n)}{\max_{k \in \{1, \dots, K\}} \text{diam}(C_k)} \right). \quad (7.59)$$

If the number of clusters K is not known, one can estimate D_K for different values, e.g., by performing K -means clustering choosing different numbers of clusters. Then the number of clusters that maximize the Dunn index can be selected as the best number of clusters. In general, larger values of D_K indicate a better clustering since the aim is to maximize the distance between clusters.

Davies-Bouldin Index The Davies-Bouldin index, DB , is defined by

$$DB = \frac{1}{K} \sum_{k=1}^K R_k, \quad (7.60)$$

where

$$R_k = \max_{n \in \{1, \dots, K\}, n \neq k} R_{kn}, \quad (7.61)$$

$$R_{mn} = \frac{s_m + s_n}{d(C(m), C(n))}, \quad (7.62)$$

with

$$s_k = \left(\frac{1}{b_k} \sum_{x_i \in C(k)} |x_i - m_k|^r \right)^{1/r}, \quad (7.63)$$

$$d(C(m), C(n)) = \left(\sum_{i=1}^n |m_{m,i} - m_{n,i}|^q \right)^{1/q}. \quad (7.64)$$

Note that, here, b_k is the number of data points in cluster $C(k)$ and m_k is its centroid.

Silhouette Coefficient The silhouette coefficient [406], denoted s_i , is defined, for each data point x_i , with $x_i \in C(k)$, by

$$s_i(k) = \frac{b_i - a_i(k)}{\max_i (a_i(k), b_i)}, \quad (7.65)$$

$$a_i(k) = \frac{1}{N_k} \sum_{j: x_j \in C(k)} d(x_j, x_i), \quad (7.66)$$

$$b_i = \min_{m \neq k} \{a_i(m)\}. \quad (7.67)$$

where a_i denotes the average distance between the data point x_i and all the other data points in its cluster and b_i denotes the minimum average distance between x_i and the data points in other clusters. By definition, the silhouette coefficient, s_i , is normalized, i.e.,

$$-1 \leq s_i \leq +1, \quad \text{for all } i. \quad (7.68)$$

Average Silhouette Coefficient For the n data points $\{x_i\}_{i=1}^n$, the average silhouette coefficient is just the average of all silhouette values

$$s_K = \frac{1}{K} \sum_{i=1}^K s_i. \quad (7.69)$$

To evaluate s_K quantitatively, the following characteristic values were suggested [406]:

- 0.71–1.00: strong clusters
- 0.51–0.70: reasonable good clusters
- 0.26–0.50: weak clusters
- < 0.25: no substantial clusters

7.9 Summary

Clustering methods are a type of unsupervised learning methods. In this chapter, we have seen that clustering algorithms can discover structures in data without using information about labels. For this reason, clustering is a powerful approach to perform an exploratory data analysis and discover new groups or classes in a data set. This is usually the first step in a data science project because it allows a largely unbiased interrogation of the data.

The application of clustering algorithms is not always straightforward since the true number of clusters hidden in the data is usually unknown. For this reason, although there exist various measures to evaluate the goodness/quality of clusters, a clustering solution is in the eye of a beholder. Therefore, it's not surprising that the evaluation of clusters is the most intricate part of a clustering analysis, in practice.

Learning Outcome 7: Clustering

Clustering methods are based on unlabeled data. For this reason, the evaluation of the found clusters/groups is challenging since no reference information is available that could be directly used for their evaluation.

In this chapter, we have seen that instances to be cluster cannot only correspond to profile vectors but networks. Even more generally, one can cluster images, documents, or human behavior. Since this can correspond to very different data types (as discussed in Chap. 5), the underlying similarity measures can assume in such cases domain-specific forms. Nevertheless, the basic idea of those clustering methods is very similar to those discussed in this chapter.

7.10 Exercises

1. Show that $s_1 : \mathbb{R} \times \mathbb{R} \rightarrow [0, 1]$ defined by $s_1(x, y) := e^{-(x-y)^2}$ is a similarity measure according to Definition 7.1.
2. Show that $d_1 : \mathbb{R} \times \mathbb{R} \rightarrow [0, 1]$ defined by $d_1(x, y) := 1 - e^{-(x-y)^2}$ is a distance measure according to Definition 7.2.
3. Given a similarity measure $s(x, y)$, where $s(x, y) \leq 1$, i.e., $s(x, y)$ fulfills the properties given by Definition 7.1, show that $d(x, y) = 1 - s(x, y)$ is a distance measure.
4. Find some examples in terms of applications for non-symmetric similarity or distance measures.
5. Calculate the Euclidian distance between $v_1 := (1, 2, 0)$ and $v_2 := (-2, 1, 7)$.
6. Use R to compute the Euclidian distance between v_1 and v_2 .

7. Given the vectors

$$a := (0, 0, 1, 2),$$

$$b := (2, 1, 1, 2),$$

$$c := (8, 0, 2, 2),$$

$$d := (0, -1, 1, 2),$$

$$e := (20, 10, 1, 4),$$

$$f := (0, 0, 0, 23),$$

$$g := (1, 1, 1, 1),$$

use R to pairwise compute the Euclidian distances between these vectors and generate the corresponding distance matrix.

8. Use R to run K -means with $K = 2$ and Euclidian distance. Use the function `scale()` to standardize the data. To perform the clustering, choose two initial centroids arbitrarily. Compare the clustering results using different initial centroids.
9. Use R to run agglomerative clustering with Euclidian distance and average linkage. Use the function `scale()` to standardize the data. Plot a dendrogram. Find a method to find meaningful clusters from this dendrogram.

Chapter 8

Dimension Reduction



8.1 Introduction

When speaking about big data, one refers generally to the (sample) size of the data, which is also called volume. However, there is another entity, which can make data “big” in some sense, and this is the dimensionality of a data point. Specifically, for data represented by a $n \times p$ matrix, where n corresponds to the number of samples (observations) and p corresponds to the number of features, a data point is represented as a p -dimensional vector where its components correspond to the so-called features.

In data science, we are frequently confronted with data sets having a large number of features. However, many of these features are often highly redundant or non-informative, and this generally hinders the ability of most machine learning algorithms to perform efficiently. A common approach to address these issues is to check whether we can detect a low-dimensional structure within these high-dimensional data. If the answer is yes, then we can identify the most meaningful basis in a lower dimension, which can be used to re-represent the data. This results in a new data matrix of the form $n \times k$ with $k < p$ and possibly $k \ll p$.

The procedures used to devise such a compact representation of the data without a significant loss of information are referred to as *dimension reduction* (or *dimensionality reduction*) techniques. According to their working mechanisms, most dimension reduction techniques can be divided into two categories:

1. Feature extraction techniques: These methods generate a small set of new features containing most of the information from the original data set via some linear/nonlinear weighted combination of the original features.
2. Feature selection techniques: These methods identify and select the most relevant features from the original data set.

In this chapter, we introduce both techniques, and we present also some examples for such methods. Specifically, for feature extraction, we discuss PCA (principal

component analysis) and NNMF (non-negative matrix factorization), whereas for feature selection, we present maximum relevance and MRMR (minimum redundancy and maximum relevance) techniques. Most of the methods discussed in this chapter are based on unlabeled data; hence, they belong to the paradigm of unsupervised learning methods.

8.2 Feature Extraction

A variety of approaches for feature extraction have been proposed, including principal component analysis [250, 380], Isomap [458], diffusion maps [294], local tangent space analysis [518], and multilayer autoencoders [98, 234]. However, in this chapter, we will restrict ourselves to the most popular of these methods, namely, the principal component analysis (PCA).

8.2.1 An Overview of PCA

PCA is a feature extraction process by definition, and therefore it aims to find a subset of linear combinations of the original features, which encompasses the majority of the variation within the data. The elements of the sought-after subset are referred to as *principal components*. They are mutually uncorrelated, and they are extracted such that the first few encompass most of the variation within the original features or variables. The principal components are extracted in a decreasing order of importance, with the first accounting for the maximum variation within the original data set. The second principal component represents the variance-maximizing direction orthogonal to the first principal component. Subsequently, it follows that the k th principal component is the direction which maximizes variance among all directions orthogonal to the previous $k - 1$ components.

In other words, PCA seeks the most accurate data representation in a lower-dimensional space through a linear transformation of an original set of features or variables into a substantially smaller set of uncorrelated variables, which represent most of the information in the original set of features or variables [266].

As such, PCA enables to highlight any trends, patterns, and outliers in the data, which may have been unclear from the original data set. Due to its simplicity of extracting important information from complex data sets, PCA is used abundantly in many forms of analysis, in particular for any initial analysis of large data sets to obtain insights about the dimensionality of the data and the distribution of the variance within these dimensions.

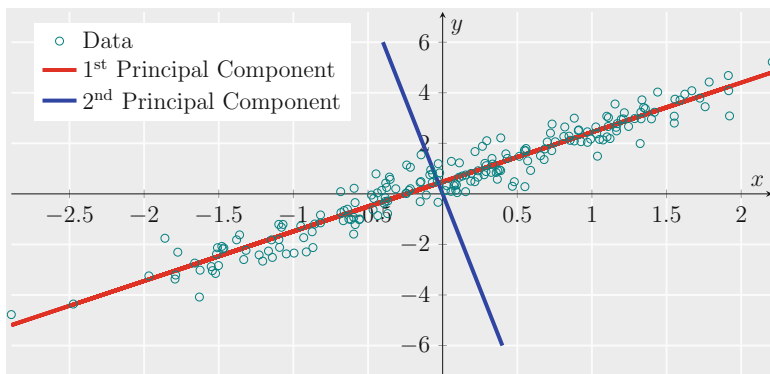


Fig. 8.1 Example of PCA for a two-dimensional data set

8.2.2 Geometrical Interpretation of PCA

Geometrically, PCA is analogous to fitting a hyperplane to noisy data, and this can be achieved via a series of rotations and/or projections with orthogonality constraints on the data points from a high-dimensional space onto a lower-dimensional space, as illustrated in Fig. 8.1.

In Fig. 8.1, it is clear that the first principal component lies along the direction of the highest variation within the data, whereas the second principal component is orthogonal to the first one. The two principal components form the new axes for the data, and they can be viewed as a result of the rotation of the original x - y axes.

On the other hand, in Fig. 8.2, the difference in the degree of variation within the data along the first and second principal components is not obvious. However, the two principal components are orthogonal to each other.

8.2.3 PCA Procedure

Let $X \in \mathbb{R}^{n \times p}$ denote the data matrix, where the columns and rows represent the features or variables and the observations, respectively. The PCA procedure on the matrix X can be summarized by the following key steps:

1. Center the variables in X so that the mean of each column is equal to 0. When the variables are measured in different units, then the centered matrix needs to be standardized, i.e., by dividing each columns by its norm. After a such transformation, each variable has a unit norm. Let $\hat{X}$ denote the corresponding transformed matrix.
2. Calculate the matrix

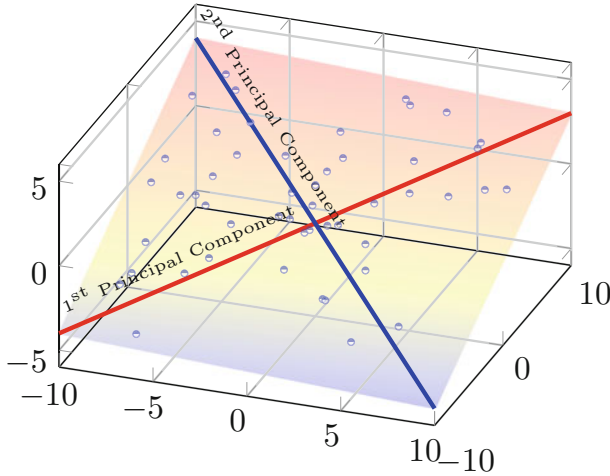


Fig. 8.2 Example of PCA for a three-dimensional data set. Only the plane defined by the first two principal components is represented

$$S = \frac{1}{n-1} \hat{X}^T \hat{X}. \quad (8.1)$$

If the matrix $\hat{X}$ consists of the centered variables in X , then S is called the *covariance matrix*, and in this case, the subsequent analysis is referred to as a covariance PCA. On the other hand, if $\hat{X}$ is the centered and standardized form of X , then S is called the *correlation matrix*, and the subsequent analysis is referred to as a correlation PCA.

3. Find the p eigenvalues of S , denoted λ_i , $i = 1, \dots, p$, and their corresponding right eigenvectors u_i such that

$$S u_i = \lambda_i u_i, \quad i = 1, \dots, p, \quad (8.2)$$

$$\|u_i\| = 1, \quad i = 1, \dots, p, \quad (8.3)$$

$$u_i^T u_j = 0, \quad i \neq j. \quad (8.4)$$

Let Λ denote the following diagonal matrix:

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_p \end{bmatrix}, \quad \text{with } \lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p,$$

and $U = (u_1, \dots, u_p) \in \mathbb{R}^{p \times p}$ is the matrix of the eigenvectors. Then, the matrix S can be obtained as follows:

$$S = U \Lambda U^T. \quad (8.5)$$

The eigenvector u_i represents one of the directions of the principal components, whereas the corresponding eigenvalue λ_i represents the variance of the data along the direction u_i . The projection of a vector x along the direction u_i is given by $u_i^T x$.

8.2.4 Underlying Mathematical Problems in PCA

Suppose that we have a data set represented by a matrix $X \in \mathbb{R}^{n \times p}$, i.e., the data consist of n observations and p variables. Then, the PCA of the data matrix X consists of a series of best-fit subspaces of dimensions $r = 1, 2, p - 1$ along the directions, which maximize the variation within the data. In other words, at each step, we are seeking a vector such that the variance of the projections of the data points in X onto the corresponding one-dimensional subspace is maximized. Since the sample covariance/correlation matrix of the data points in the matrix X is given by

$$S = \frac{1}{n-1} \hat{X}^T \hat{X}, \quad (8.6)$$

where $\hat{X}$ denotes the transformed version of X , then we can find the first best-fit subspace by solving the following optimization problem:

$$\max_u u^T \hat{X}^T \hat{X} u \quad (8.7)$$

$$u^T u = 1 \quad (8.8)$$

The optimal solution of the optimization problem given by Eq. 8.7–8.8, denoted u_1 , is the eigenvector of the matrix S associated with the largest eigenvalue.

The second best-fit subspace is obtained by solving the following optimization problem:

$$\max_u u^T \hat{X}^T \hat{X} u \quad (8.9)$$

$$u^T u = 1 \quad (8.10)$$

$$u_1^T u = 0 \quad (8.11)$$

The additional constraint given by Eq. 8.11, compared to the optimization problem given in Eq. 8.7–8.8, enforces the orthogonality between the first best-fit

subspace obtained previously (which is characterized by u_1) and the sought-after second best-fit subspace. The optimal solution of the optimization problem given in Eq. 8.9–8.11, denoted u_2 , is the eigenvector of the matrix S associated with the second largest eigenvalue.

Similarly, the k th best-fit subspace is given by the solution of the following optimization problem:

$$\max_u u^T \hat{X}^T \hat{X} u \quad (8.12)$$

$$u^T u = 1 \quad (8.13)$$

$$u_i^T u = 0, \quad i = 1, \dots, k-1. \quad (8.14)$$

The vectors u_i , $i = 1, \dots, k < p$, also referred to as the *loadings of the principal components*, form the first k columns of the PCA rotation matrix.

The above steps can be generalized to find the best-fit k -dimensional subspace, with $k < p$, solving the following optimization problem:

$$\max_U U^T \hat{X}^T \hat{X} U \quad (8.15)$$

$$U^T U = I, \quad (8.16)$$

where U denotes the basis of the optimal k -dimensional subspace onto which to project the data set $\hat{X}$ and I denotes the $k \times k$ identity matrix.

The solution of the quadratic problem 8.15–8.16 provides us with the orthonormal eigenvectors u_i , i.e., satisfying the constraints 8.3 and 8.4 in Step 3 of the PCA procedure, along which the variance of the data is maximum. Substituting the eigenvectors u_i in the constraint in Eq. 8.2, we can solve the corresponding linear problem to obtain the eigenvalues λ_i . However, this multi-step solution to obtain the eigenvalues λ_i and their associated eigenvectors u_i can be replaced by a single-step process using a computationally efficient technique, on the matrix $\hat{X}$, called the singular value decomposition.

8.2.5 PCA Using Singular Value Decomposition

Singular value decomposition (SVD) is a generalized approach for matrix factorization. Let $X \in \mathbb{R}^{n \times p}$ denote the original data matrix, where the columns represent the features or variables in the data, and let $\hat{X}$ denote the corresponding centered and standardized matrix. Then, the SVD of $\hat{X}$ is given by the following factorization:

$$\hat{X} = U \Lambda V^T, \quad (8.17)$$

where $U \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{p \times p}$ are unitary or orthogonal matrices, i.e., $U^T U = I_n$ and $V^T V = I_p$, and $\Lambda \in \mathbb{R}^{n \times p}$ is a matrix with real, non-negative entries on the diagonal and zero off-diagonal elements.

The matrix Λ has at most $r = \min(n, p)$ non-zero diagonal entries, λ_i for $i = 1, 2, \dots, r$, which are ordered in descending order

$$\lambda_1 \geq \lambda_2 \geq \dots \geq 0. \quad (8.18)$$

The matrices U and V are referred to as the left and right singular vectors, respectively, whereas Λ is the diagonal matrix of singular values.

The projection of the observations onto the principal component space, also referred to as the *scores' matrix*, denoted Y , is given by

$$Y = U \Lambda. \quad (8.19)$$

The matrix V , referred to as the *loadings' matrix*, can also be used to obtain the projection of the observations onto the principal component space, Y . This can be shown using Eqs. 8.17 and 8.19 as follows:

$$Y = U \Lambda \quad (8.20)$$

$$= U \Lambda V^T V, \text{ since } V^T V = I_p \quad (8.21)$$

$$= \hat{X} V. \quad (8.22)$$

Therefore, given the scores' matrix Y and the loadings' matrix V , the original data matrix $\hat{X}$ can be recovered as follows:

$$\hat{X} = Y V^T \quad (8.23)$$

For a new given observation vector $x_{\text{new}} \in \mathbb{R}^{1 \times p}$, not included in the PCA process, its projection onto the principal component space or its scores, denoted y_{new} , can be obtained as follows:

$$y_{\text{new}} = \hat{x}_{\text{new}} V, \quad (8.24)$$

where $\hat{x}_{\text{new}}$ is the centered and normalized version of x_{new} obtained using the same values of the mean and standard deviation utilized to derive matrix $\hat{X}$.

8.2.6 Assessing PCA Results

The results of a PCA can be assessed using various metrics, including:

- The importance of each principal component, reflected by the magnitude of its corresponding eigenvalue, i.e., the fraction of variation within the original data captured by the principal component.
- The correlation between a principal component and a variable.
- The contribution of a given observation i in the construction of a given principal component j , denoted $\text{contrib}_{i,j}$, given by

$$\text{contrib}_{i,j} = \frac{y_{i,j}^2}{\lambda_j}, \quad (8.25)$$

where $y_{i,j}$ is the score, i.e., the projection of the observation i onto the principal component j , and λ_j is the eigenvalue of the principal component j .

- The contribution of a principal component j for the representation of a given observation i in the principal component space, denoted $\cos_{i,j}^2$, given by

$$\cos_{i,j}^2 = \frac{y_{i,j}^2}{\sum_j y_{i,j}^2}, \quad (8.26)$$

where $y_{i,j}$ is the score, i.e., the projection, of the observation i onto the principal component j ; the quantity $\cos_{i,j}^2$ is called the *squared cosine* between the observation i and the principal component j .

A commonly used approach, to identify how many principal components should be retained, is to plot the eigenvalues as a function of their indices, which denote the order of importance of the principal components, and an elbow in the plot provides a cut-off point. Such a plot is known as a *scree plot*. Another approach is to consider a principal component as relevant when its associated eigenvalue is larger than the mean of all the eigenvalues.

8.2.7 Illustration of PCA Using R

In R, PCA can be performed readily using the linear algebra tools and following the steps defined in Sects. 8.2.3 or 8.2.5. However, many packages, which enable to perform PCA directly, are available, including *h2o*, *FactoMineR*, *ade4*, *amap*, and *stats*. We will use the latter package to illustrate the application of PCA on the data set *PimaIndiansDiabetes*, available in the package *mlbench*. The analysis and the visualization of the results are performed using Listing 8.1.

Listing 8.1: Implementation of PCA using R, see Fig. 8.3

```

#Loading required libraries
library("devtools")
library("FactoMineR")
library("factoextra")
library("corrplot")
library(mlbench)

#Loading the data
data(PimaIndiansDiabetes)
rawData = PimaIndiansDiabetes

#An overview of the data
head(rawData)

#PCA on the quantitative variables
pca_rawData<- prcomp(rawData[,c(1:8)], scale = TRUE)

#Summary of the PCA results
summary(pca_rawData)

#Plotting the screeplot
fviz_eig(pca_rawData, addlabels = TRUE, ylim = c(0, 40),
         choice = c("variance", "eigenvalue"),
         geom = c("bar", "line"), barfill = "#D6604D",
         barcolor = "#D6604D", linecolor = "#2166AC")

#Plot of the correlation between the variables and the first two
principal components
fviz_pca_var(pca_rawData, col.var = "cos2",
             gradient.cols = c("#B2182B", "#FDDBC7", "#2166AC"),
             repel = TRUE)

#Plot of the contribution of the variables to the principal
components
corrplot(results_var$contrib, method = "square", col =
         terrain.colors(100), is.corr=FALSE)

#Plot of the projection of the observations into the space of the
first two principal components
fviz_pca_ind(pca_rawData,
             geom.ind = "point",
             col.ind = rawData$diabetes, # color by variable
             diabetes
             palette = c("#F4A582", "#4393C3", "#FDDBC7"),
             legend.title = "Diabetes outcome"
)

```

The scree plot, of the corresponding analysis, is depicted in Fig. 8.3, and it can be used to identify the appropriate number of principal components to be retained. For this example, there appears to be an elbow at the third principal component. Therefore, the elbow approach suggests to retain the first three principal components, which account for 60.7% of the overall variance. The alternative approach also suggests the retention of the first three principal components, since only the values of the first three eigenvalues are greater than the mean of all the eigenvalues.

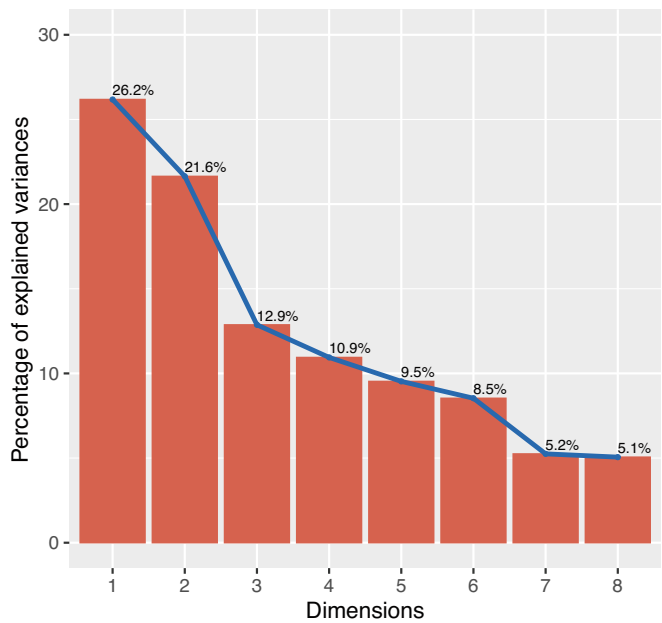


Fig. 8.3 Scree plot for the PCA in Listing 8.1

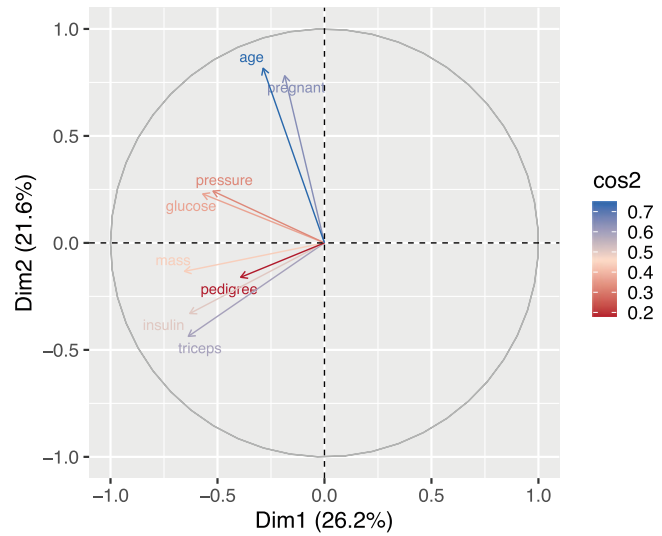


Fig. 8.4 Correlation between the first two principal components and the variables

The quality of the PCA can be assessed using the correlation between the retained principal components and the variables. Figure 8.4 shows the representation of the correlations between the variables and the first two principal components. The sign

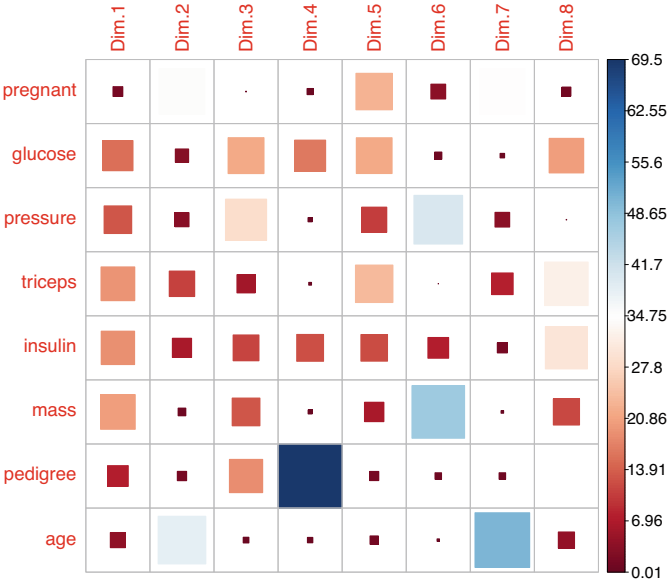


Fig. 8.5 Contribution of the variables in the construction of the principal components: the larger the box in the cell (i, j), the higher the contribution of the variable in row i into the construction of the principal component in column j

of the dimension axis indicates the sign of the correlation, whereas the magnitude of the corresponding vector represents the degree of correlation. The variables *age* and *pregnant* have a high positive correlation and a low negative correlation with the first and second principal components, respectively. The variables *glucose* and *pressure* have a modest positive correlation and a moderate negative correlation with the first and second principal components, respectively. The variables *insulin* and *triceps* have a moderate negative correlation with both the first and second principal components. The variable *mass* has a moderate negative correlation and low negative correlation with the first and second principal components, respectively. The variable *pedigree* has a modest negative correlation with both the first and second principal components.

Figure 8.5 depicts the contribution of the variables in the construction of the principal components. The variables are represented by the rows, whereas the principal components are represented by the columns. Most of the variables have some moderate to low contribution in the construction of most principal components, whereas:

- The variable *pregnant* has only a moderate contribution to the construction of the fifth principal component (denoted Dim.5) and a very low or no contribution at all to the construction of the other principal components.
- The variable *pedigree* has a high contribution to the construction of the fourth principal component (denoted Dim.4), a moderate contribution to the construc-

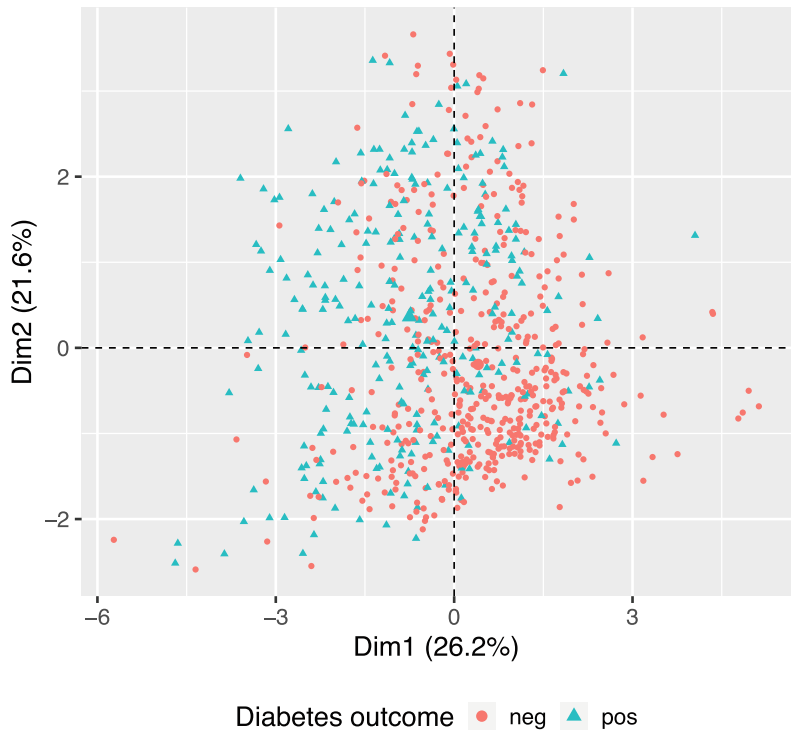


Fig. 8.6 Projection of the observations into the space of the first two principal components

- tion of the third principal component (denoted Dim.3), and a very low or no contribution at all to the construction of the other principal components.
- The variable *age* has a significant contribution to the construction of the second and seventh principal components (denoted Dim.2 and Dim.7), while its contribution to the construction of the other principal components is very low.
 - The variables *mass* and *pressure* are the main contributors to the construction of the sixth principal component (denoted Dim.6).

We can further assess the quality of the results from the PCA by analyzing the representation of the observations in the space of the retained principal components. For our particular example, we can assess whether the representation of the observations in the space of the first two principal components could facilitate the classification of observations in terms of their diabetes test outcome. Figure 8.6 provides some insights on the complexity of the task to be performed by a classification model, for predicting diabetes outcome, if only the first two principal components are to be retained. In fact, there is no separation between the observations with a negative diabetes outcome, represented by circles, and those with a positive diabetes outcome, represented by triangles. This suggests that we would need to consider additional principal components or use a kernel PCA approach, which will be introduced in the next section.

Listing 8.2 provides an illustration of PCA using SVD via the package *MASS*. This package enables to carry out the SVD of a given matrix $X \in \mathbb{R}^{m \times n}$ and outputs the matrices U , D , and V . In this listing, X is an 8×8 matrix, and the output matrices U , D , and V were used to estimate the approximation of the matrix X , denoted $\hat{X}$, using the following scenarios: (a) all the principal components, (b) only the principal components corresponding to the four largest eigenvalues, and (c) only the principal components corresponding to the two largest eigenvalues. The visualization of the matrices, X , U , D , V , and $\hat{X} = UV$, for each of the three cases, was depicted in Fig. 8.7a, b, and c, respectively.

From the visualization results, it is clear that using all the principal components enables to reconstruct perfectly the matrix X . On the other hand, using only the principal components corresponding to the four largest eigenvalues provides a good reconstruction of the matrix X , whereas using only the principal components corresponding to the two largest eigenvalues results in an average reconstruction of the matrix X .

Listing 8.2: Illustration of PCA using SVD in R

```
library(MASS)
library(mlbench)

data(PimaIndiansDiabetes)
rawData = PimaIndiansDiabetes
X = rawData[,c(1:8), c(1:8)]

#PCA of X using SVD
X.svd<- svd(X)
D <- diag(X.svd$d)
U <- X.svd$u
V <- X.svd$v

#Reconstruction of X using all the Principal Components
Xhat <- U%%D%%t(V)

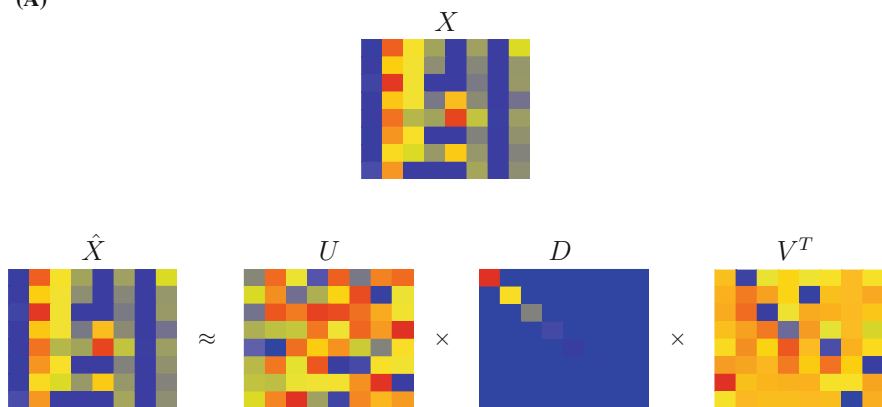
#Reconstruction of X using the first four Principal Components
Xhat4PCs <- Usvd[,1:4]%%Dsvd[1:4,1:4]%%t(Vsvd[,1:4])

#Reconstruction of X using the first two Principal Components
Xhat2PCs <- Usvd[,1:2]%%Dsvd[1:2,1:2]%%t(Vsvd[,1:2])
```

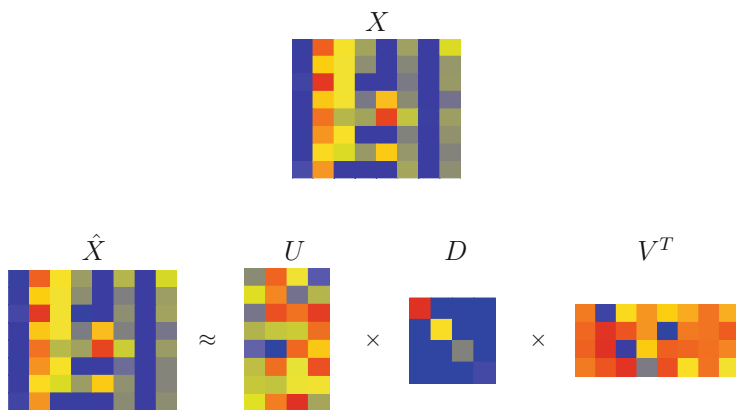
8.2.8 Kernel PCA

Although PCA enables to reduce the dimension of the data matrix, this doesn't always improve the performance of a model, as shown in the example illustrated in Fig. 8.6. For this reason, an extension to PCA, called kernel PCA (KPCA), has been introduced.

(A)



(B)



(C)

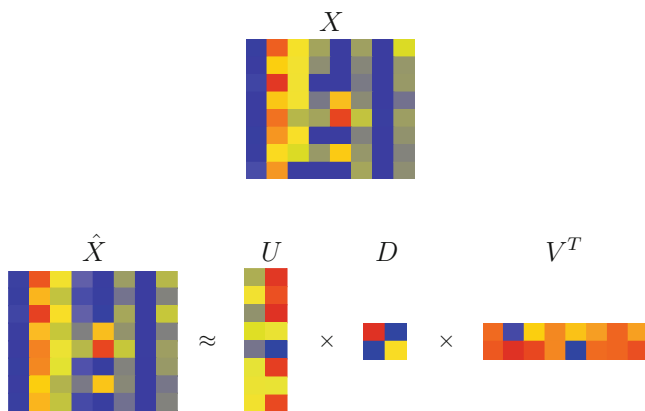


Fig. 8.7 Visualization of the reconstruction of matrix X , in Listing 8.2: (a) Using all principal components. (b) Using only the principal components corresponding to the four largest eigenvalues. (c) Using only the principal components corresponding to the two largest eigenvalues

KPCA is the nonlinear form of PCA, which leverages some complex spatial transformations of the features. These transformations may sometimes require moving into a high-dimensional feature space because such a transformation could enable a simpler model to perform better on data with a complex structure.

In order to illustrate the concept of KPCA, let us consider a binary classification problem, where the two classes are highlighted in red and blue in the plots in Fig. 8.8. The graphs on the left represent the original data in a two-dimensional space, which obviously require nonlinear classifiers to model the problem. The graphs on the right-hand side represent the transformed data in a three-dimensional space. Clearly, linear classifiers would perform well in differentiating the two classes, in the space of the transformed data.

The mapping process, used in KPCA to transform the data, is called a kernel. Many kernels have been suggested in the literature, including:

- Linear kernel:

$$k(x, x') = x^T x';$$

- Polynomial kernel of degree d :

$$k(x, x') = (x^T x' + 1)^d;$$

- Gaussian kernel or radial basis function with bandwidth σ :

$$k(x, x') = \exp(-\|x - x'\|^2 / 2\sigma^2),$$

where x denote a vector of observations in the data set on interest.

8.2.9 Discussion

Although the origin of PCA can be traced back to Pearson [380] (1901), it remains one of the most popular techniques used in multivariate analysis. PCA is a valuable tool for removing correlated features, which can reduce the performance of machine learning algorithms. Furthermore, PCA helps to reduce the problem of overfitting, which can result from the high dimensionality of the data.

By reducing the dimensionality of the data, PCA offers also the opportunity to visualize the data. Visualization of the data is an important step in data science and explorative data analysis as discussed in Chap. 6. Note that the PCA presented in the previous sections can only be used on quantitative variables, i.e., numerical but not categorical features. However, some generalizations of PCA, e.g., correspondence analysis (CA) and multiple factor analysis (MFA), allow to address the cases of qualitative variables and mixed variables (quantitative and qualitative), respectively.

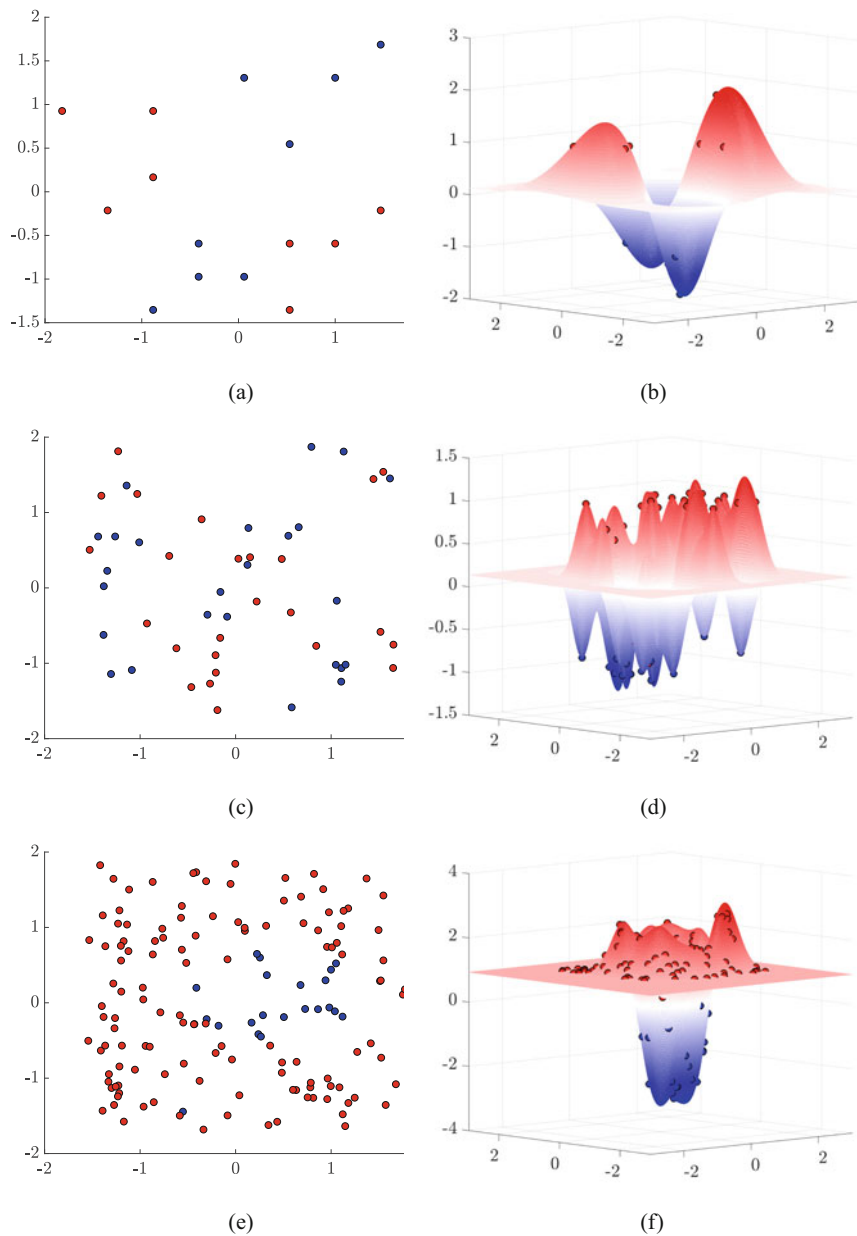


Fig. 8.8 Illustrative examples of the relevance of KPCA. (a) Example 1: original data in 2D. (b) Example 1: transformed data in 3D. (c) Example 2: original data in 2D. (d) Example 2: transformed data in 3D. (e) Example 3: original data in 2D. (f) Example 3: transformed data in 3D

8.2.10 Non-negative Matrix Factorization

Non-negative matrix factorization (NNMF) is one of the most widely used tools to analyze high-dimensional data. It enables to automatically extract meaningful features from a non-negative data set. Unsupervised learning techniques, such as PCA, can be viewed as constrained matrix factorization problems. Let $X \in \mathbb{R}^{n \times p}$ denote an $n \times p$ matrix with non-negative elements, representing n samples of p -dimensional data. Then, the non-negative matrix factorization of X consists of finding two matrices $U \in \mathbb{R}^{n \times m}$ and $V \in \mathbb{R}^{m \times p}$, both with non-negative elements, such that $m < \min(n, p)$ and

$$X = UV. \quad (8.27)$$

Since the matrices U and V are smaller compared to X , then such a mapping can be viewed as a compression of the data within X .

Mathematically, the NNMF problem can be formulated as follows:

$$\begin{aligned} &\text{Find } (U, V) \in (\mathbb{R}^{n \times m}, \mathbb{R}^{m \times p}) \\ &\text{subject to } X = UV, \\ &\quad [U]_{il} \geq 0, i = 1, \dots, n, l = 1, \dots, m \\ &\quad [V]_{lj} \geq 0, l = 1, \dots, m, j = 1, \dots, p, \end{aligned} \quad (8.28)$$

where $[U]_{il}$ and $[V]_{lj}$ denote the elements of the matrices U and V , respectively.

To solve the problem 8.28, the following approximated form is generally used:

$$\begin{aligned} &\min_{U \in \mathbb{R}^{n \times m}, V \in \mathbb{R}^{m \times p}} F(X, UV) \\ &\text{subject to} \\ &\quad [U]_{il} \geq 0, i = 1, \dots, n, l = 1, \dots, m \\ &\quad [V]_{lj} \geq 0, l = 1, \dots, m, j = 1, \dots, p, \end{aligned} \quad (8.29)$$

where the objective function F is a suitable scalar measure of the matrix X and the product of the sought-after matrices, U and V . Examples of such scalar measure include:

1. The Frobenius norm, where

$$F(X, UV) = \|X - UV\|_F^2 \quad (8.30)$$

$$= \sum_{i=1}^n \sum_{j=1}^p |[X]_{ij} - [UV]_{ij}|^2, \quad (8.31)$$

with $[X]_{ij}$ and $[UV]_{ij}$ denoting the elements of the matrices X and UV , respectively.

2. The *generalized* Kullback-Leibler divergence or the relative entropy, where

$$F(X, UV) = \sum_{i=1}^n \sum_{j=1}^p \left([X]_{ij} \log \frac{[X]_{ij}}{[UV]_{ij}} - [X]_{ij} + [UV]_{ij} \right), \quad (8.32)$$

with $[X]_{ij}$ and $[UV]_{ij}$ denoting the elements of the matrices X and UV , respectively.

Note that the functions 8.30 and 8.32 are not convex in U and V . Therefore, the NMF is a non-convex optimization problem, and available numerical optimization techniques can only guarantee locally optimal solutions to the problem. On the other hand, since matrix multiplication is bilinear, the function $F(X, UV)$, such as 8.30 and 8.32, is convex in its argument U and also convex in its argument V , respectively.

The most commonly used approach to find a local optimum of the problem 8.29 is a variant of the gradient descent method, known as the block-coordinate descent. Let U_0 and V_0 denote some given initial values of the matrices U and V . Then, the method alternates between optimizing U and optimizing V , respectively, as follows:

At a given iteration k , the updated matrices U_k and V_k are given by

$$U_k = \arg \min_U F(X, UV_{k-1}) \quad (8.33)$$

$$= U_{k-1} - \eta_U \odot \nabla_U F(X, U_{k-1} V_{k-1}), \quad (8.34)$$

$$V_k = \arg \min_V F(X, U_k V_{k-1}) \quad (8.35)$$

$$= V_{k-1} - \eta_V \odot \nabla_V F(X, U_k V_{k-1}), \quad (8.36)$$

where η_U and η_V are the learning rate matrices used to update the matrices U and V , respectively, the symbol $\odot$ denotes the Hadamard product or the element-wise product, and partial gradients $\nabla_U F$ and $\nabla_V F$ are given by

$$\nabla_U F = \begin{bmatrix} \frac{\partial F}{\partial u_{11}} & \frac{\partial F}{\partial u_{12}} & \cdots & \frac{\partial F}{\partial u_{1p}} \\ \frac{\partial F}{\partial u_{21}} & \frac{\partial F}{\partial u_{22}} & \cdots & \frac{\partial F}{\partial u_{2p}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F}{\partial u_{m1}} & \frac{\partial F}{\partial u_{m2}} & \cdots & \frac{\partial F}{\partial u_{mp}} \end{bmatrix} \quad \nabla_V F = \begin{bmatrix} \frac{\partial F}{\partial v_{11}} & \frac{\partial F}{\partial v_{12}} & \cdots & \frac{\partial F}{\partial v_{1n}} \\ \frac{\partial F}{\partial v_{21}} & \frac{\partial F}{\partial v_{22}} & \cdots & \frac{\partial F}{\partial v_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F}{\partial v_{p1}} & \frac{\partial F}{\partial v_{p2}} & \cdots & \frac{\partial F}{\partial v_{pn}} \end{bmatrix} \quad (8.37)$$

8.2.10.1 NMF Using the Frobenius Norm as Objective Function

Using the Frobenius norm, the objective function of the NMF problem 8.29 writes

$$\begin{aligned}
 F(X, UV) &= \|X - UV\|_F^2 \\
 &= \text{tr}[(X - UV)^T(X - UV)] \\
 &= \text{tr}[X^T X - X^T(UV) - (UV)^T X + (UV)^T(UV)] \\
 &= \text{tr}[X^T X] - \text{tr}[X^T(UV)] - \text{tr}[(UV)^T X] + \text{tr}[(UV)^T(UV)] \\
 &= \text{tr}[X^T X] - \text{tr}[X^T UV] - \text{tr}[V^T U^T X] + \text{tr}[V^T U^T UV]
 \end{aligned}$$

where $\text{tr}[Z]$ denote the trace of the matrix Z .

Then, the partial gradient of $F(X, UV)$ with respect to U is given by

$$\begin{aligned}
 \nabla_U F(X, UV) &= \nabla_U \text{tr}[X^T X] - \nabla_U \text{tr}[X^T UV] - \nabla_U \text{tr}[V^T U^T X] + \nabla_U \text{tr}[V^T U^T UV] \\
 &= \nabla_U \text{tr}[X^T X] - \nabla_U \text{tr}[V X^T U] - \nabla_U \text{tr}[X V^T U^T] + \nabla_U \text{tr}[U V V^T U^T] \\
 &= 0 - (V X^T)^T - X V^T + U \left[(V V^T)^T + V V^T \right] \\
 &= 0 - X V^T - X V^T + 2U V V^T \\
 &= -2(X V^T - U V V^T).
 \end{aligned}$$

Similarly, the partial gradient of $F(X, UV)$ with respect to V is given by

$$\begin{aligned}
 \nabla_V F(X, UV) &= \nabla_V \text{tr}[X^T X] - \nabla_V \text{tr}[X^T UV] - \nabla_V \text{tr}[V^T U^T X] + \nabla_V \text{tr}[V^T U^T UV] \\
 &= 0 - (X^T U)^T - U^T X + [U^T U + (U^T U)^T] V \\
 &= 0 - U^T X - U^T X + 2U^T U V \\
 &= -2(U^T X - U^T U V).
 \end{aligned}$$

Thus, the scheme 8.34–8.36 can be written as

$$U_k = U_{k-1} + \tilde{\eta}_U \odot (X V_{k-1}^T - U_{k-1} V_{k-1} V_{k-1}^T), \quad (8.38)$$

$$V_k = U_{k-1} + \tilde{\eta}_V \odot (U_k^T X - U_k^T U_k V_{k-1}), \quad (8.39)$$

where $\tilde{\eta}_U = 2\eta_U$ and $\tilde{\eta}_V = 2\eta_V$.

The updating scheme given by Eqs. 8.38–8.39 is referred to as the *additive update rules*. Since these rules are just the conventional gradient descent method,

then all the values in the learning rates need to be set to some sufficiently small positive numbers to ensure the convergence of the scheme.

Now, let's set the learning rate in Eq. 8.38 to

$$\tilde{\eta}_U = \tilde{\eta}_{U_{k-1}} = \frac{U_{k-1}}{U_{k-1}^T V_{k-1} V_{k-1}^T}, \quad (8.40)$$

where the fraction symbol denotes the element-wise division. Then, the additive update rule in Eq. 8.38 writes

$$U_k = U_{k-1} + \frac{U_{k-1}}{U_{k-1} V_{k-1} V_{k-1}^T} \odot (X V_{k-1}^T - U_{k-1} V_{k-1} V_{k-1}^T) \quad (8.41)$$

$$= U_{k-1} + U_{k-1} \odot \frac{X V_{k-1}^T}{U_{k-1} V_{k-1} V_{k-1}^T} - U_{k-1} \odot \frac{U_{k-1} V_{k-1} V_{k-1}^T}{U_{k-1} V_{k-1} V_{k-1}^T} \quad (8.42)$$

$$= U_{k-1} \odot \frac{X V_{k-1}^T}{U_{k-1} V_{k-1} V_{k-1}^T}. \quad (8.43)$$

Let set the learning rate in Eq. 8.39 to

$$\tilde{\eta}_V = \tilde{\eta}_{V_{k-1}} = \frac{V_{k-1}}{U_k^T U_k V_{k-1}}, \quad (8.44)$$

where the fraction symbol denotes the element-wise division and U_k is given by 8.43. Then, the additive update rule 8.39 writes

$$V_k = V_{k-1} + \frac{V_{k-1}}{U_k^T U_k V_{k-1}} \odot (U_k^T X - U_k^T U_k V_{k-1}) \quad (8.45)$$

$$= V_{k-1} + V_{k-1} \odot \frac{U_k^T X}{U_k^T U_k V_{k-1}} - V_{k-1} \odot \frac{U_k^T U_k V_{k-1}}{U_k^T U_k V_{k-1}} \quad (8.46)$$

$$= V_{k-1} \odot \frac{U_k^T X}{U_k^T U_k V_{k-1}}. \quad (8.47)$$

The resulting update rules 8.43–8.47, given in an element-wise form in Eqs. 8.48–8.49, are referred to as the multiplicative update rules [299].

$$[U_k]_{i,l} = [U_{k-1}]_{i,l} \frac{[X V_{k-1}^T]_{i,l}}{[U_{k-1} V_{k-1} V_{k-1}^T]_{i,l}}, \quad (8.48)$$

$$[V_k]_{l,j} = [V_{k-1}]_{l,j} \frac{[U_k^T X]_{l,j}}{[U_k^T U_k V_{k-1}]_{l,j}}, \quad (8.49)$$

with $i = 1, \dots, n$, $l = 1, \dots, m$, $j = 1, \dots, p$, and where $[A]_{rs}$ denotes the element in row r and column s of the matrix A .

The data-adaptive feature of the learning rates 8.40 and 8.44 enables the multiplicative update rules 8.48–8.49 to intrinsically satisfy the non-negativity constraint in the NNMF problem 8.29. Therefore, such learning rates improve the running time of the scheme compared to the additive update rules 8.38–8.39.

8.2.10.2 NNMF Using the Generalized Kullback-Leibler Divergence as Objective Function

Using the generalized Kullback-Leibler divergence in Eq. 8.32 as the objective function of the NNMF problem 8.29 yields the following additive update rules at iteration k [299]:

$$[U_k]_{i,l} = [U_{k-1}]_{i,l} + [\eta_U]_{i,l} \left(\sum_s \frac{[X]_{i,s}}{[U_{k-1}]_{i,s} [V_{k-1}]_{l,s}} [V_{k-1}]_{l,s} - \sum_s [V_{k-1}]_{l,s} \right) \quad (8.50)$$

$$[V_k]_{l,j} = [V_{k-1}]_{l,j} + [\eta_V]_{l,j} \left(\sum_r [U_k]_{r,l} \frac{[X]_{r,j}}{[U_k]_{r,l} [V_{k-1}]_{l,j}} - \sum_r [U_k]_{r,l} \right) \quad (8.51)$$

Setting the learning rates in Eq. 8.50 to

$$[\eta_U]_{i,l} = [\eta_{U_{k-1}}]_{i,l} = \frac{[U_{k-1}]_{i,l}}{\sum_s [V_{k-1}]_{l,s}} \quad (8.52)$$

yields

$$[U_k]_{i,l} = [U_{k-1}]_{i,l} \frac{\sum_s [V_{k-1}]_{l,s} \frac{[X]_{i,s}}{[U_{k-1}]_{i,s} [V_{k-1}]_{l,s}}}{\sum_s [V_{k-1}]_{l,s}}. \quad (8.53)$$

Now, setting the learning rates in 8.51 to

$$[\eta_V]_{l,j} = [\eta_{V_{k-1}}]_{l,j} = \frac{[V_{k-1}]_{l,j}}{\sum_r [U_{k-1}]_{r,l}} \quad (8.54)$$

yields

$$[V_k]_{l,j} = [V_{k-1}]_{l,j} \frac{\sum_r [U_{k-1}]_{r,l} \frac{[X]_{r,j}}{[U_{k-1}]_{r,l} [V_{k-1}]_{l,j}}}{\sum_s [U_{k-1}]_{s,l}} \quad (8.55)$$

The update rules 8.53–8.55 provide the multiplicative update scheme for the NMF using the generalized Kullback-Leibler divergence [299].

8.2.10.3 Example of NMF Using R

Listing 8.3 provides an illustration of NMF using the package *NMF* in R. This package allows to carry out NMF of a given matrix $X \in \mathbb{R}^{n \times p}$, for a given rank $m \leq \min(n, p)$. In this listing, X is an 8×8 matrix, and three possible values of m were considered: (a) $m = 8$, (b) $m = 4$, and (c) $m = 2$. For these three cases, the approximation of the matrix X , denoted $\hat{X}$, is estimated. The visualization of the resulting matrices, X , U , V , and $\hat{X} = UV$, for each value of m , are shown in Fig. 8.9a, b, and c, respectively. Similar to PCA, using all the features enables to reconstruct perfectly the matrix X . On the other hand, using only the four best features provides a good reconstruction of the matrix X , whereas using only the two best features results in an average reconstruction of the matrix X .

Listing 8.3: Implementation of NMF using R, see Figure 8.9

```
library(NMF)
library(mlbench)

data(PimaIndiansDiabetes)
rawData = PimaIndiansDiabetes
X = rawData[c(1:8), c(1:8)]

#Reconstruction of X using all features
X.nmf <- nmf(X, 8)
U <- basis(X.nmf)
V <- coef(X.nmf)
Xhat <- U%*%V

#Reconstruction of X using 4 features
X.nmf4f <- nmf(X, 4)
U4f <- basis(X.nmf4f)
V4f <- coef(X.nmf4f)
Xhat4f <- U4f%*%V4f

#Reconstruction of X using 2 features
X.nmf2f <- nmf(X, 2)
U2f <- basis(X.nmf2f)
V2f <- coef(X.nmf2f)
Xhat2f <- U2f%*%V2f
```

8.3 Feature Selection

In contrast with feature extraction, where a small subset of relevant features is derived through some (nonlinear) combination of the origination features, the relevant features obtained with feature selection consist of a subset of the original

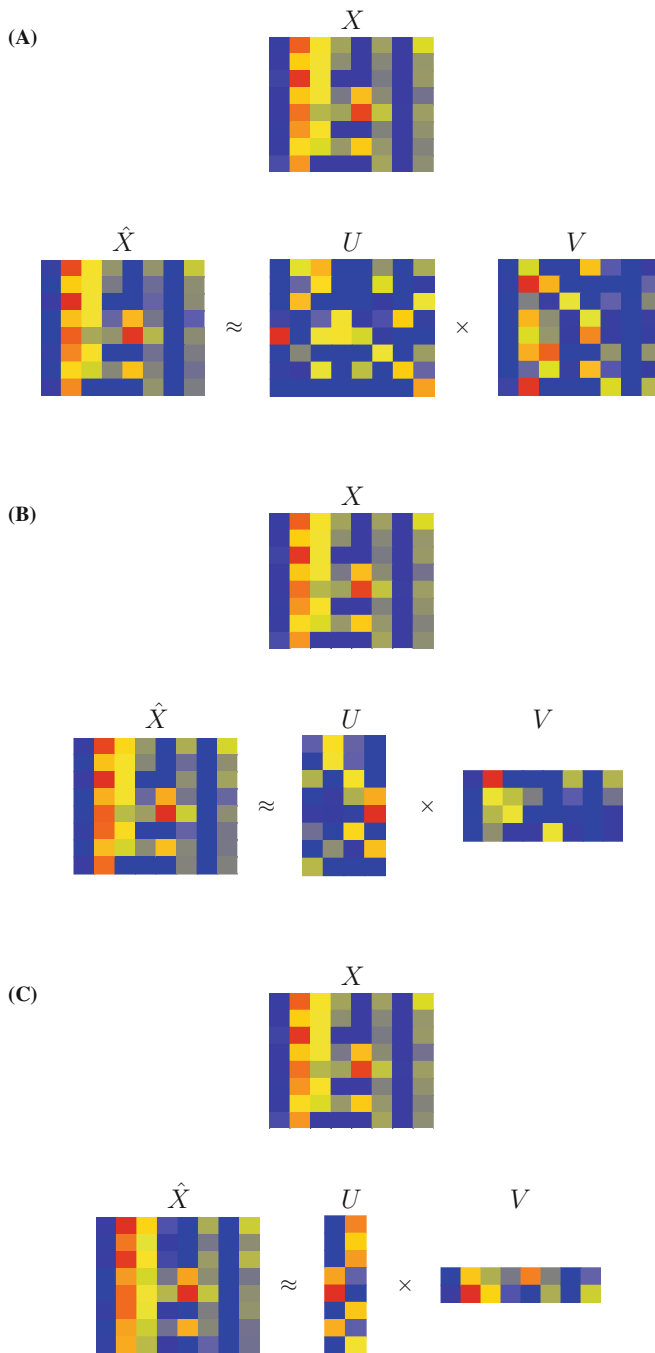


Fig. 8.9 Visualization of the reconstruction of matrix X , in Listing 8.3: (a) Using all features available. (b) Using only the four best features selected through NNMF. (c) Using only the two best features selected through NNMF

features. Feature selection is typically used to identify a subset S of the most relevant features from a high-dimensional data matrix $X \in \mathbb{R}^{n \times p}$ of input variables to target a response variable Y , where $|S| = k \ll p$. Feature selection algorithms can be categorized into three main classes:

1. *Filter methods*: statistical metrics, e.g., the Pearson correlation coefficient or the mutual information, are used to identify the most relevant features.
2. *Wrapper methods*: a learning algorithm is used to train models on various combinations of the features of matrix X and then select the features, which yield the best out-of-sample performance.
3. *Embedded methods*: they perform the feature selection during the construction of the model.

8.3.1 Filter Methods Using Mutual Information

Mutual information is a measure that quantifies a relationship between two random variables, which have been sampled simultaneously, and it forms the building block for defining criteria for feature selection in machine learning. The mutual information is based on the concept of entropy, which quantifies the uncertainty present in the distribution of a variable X , and it is defined by

$$H(X) = - \sum_{x \in X} p(x) \log p(x), \quad \text{if } X \text{ is discrete,}$$

$$H(X) = - \int_X f(x) \log f(x) dx, \quad \text{if } X \text{ is continuous,}$$

where $p(x)$ (resp. $f(x)$) denotes the probability density function of X .

The conditional entropy of X given Y is defined by

$$H(X|Y) = - \sum_{y \in Y} p(y) \sum_{x \in X} p(x|y) \log p(x|y), \quad \text{if } X \text{ and } Y \text{ are discrete,}$$

$$H(X|Y) = - \int_Y f(y) \int_X f(x|y) \log f(x|y) dx dy, \quad \text{if } X \text{ and } Y \text{ are continuous,}$$

where $p(x|y)$ (resp. $f(x|y)$) denotes the conditional probability density functions of X given Y and $p(y)$ (resp. $f(y)$) denotes the probability density function of Y .

Mutual information estimates the amount of information about a given random variable that can be obtained through another random variable. The mutual information between two random variables X and Y is defined by

$$I(X; Y) = H(X) - H(Y|X) \tag{8.56}$$

$$= \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}, \text{ if } X \text{ and } Y \text{ are discrete,} \quad (8.57)$$

$$I(X; Y) = H(X) - H(Y/X) \quad (8.58)$$

$$= \int_Y \int_X f(x, y) \log \frac{f(x, y)}{f(x)f(y)} dx dy, \text{ if } X \text{ and } Y \text{ are continuous,} \quad (8.59)$$

where $p(x, y)$ (resp. $f(x, y)$) denotes the joint probability density function of X and Y , whereas $p(x)$ and $p(y)$ (resp. $f(x)$ and $f(y)$) are the marginal probability density functions of X and Y .

When the random variables X and Y are independent, then $p(x, y) = p(x)p(y)$. From this, it follows that $I(X, Y) = 0$. In other words, the mutual information allows to establish a similarity between $p(x, y)$ and $p(x)p(y)$.

The feature selection problem consists of finding the subset S , which has the maximum mutual information between its features X_S and the target variable Y . This can be formulated via the following constrained optimization problem:

$$\hat{S} = \arg \max_S I(X_S; Y) \quad (8.60)$$

$$\text{subject to} \quad (8.61)$$

$$S \subset X \quad (8.62)$$

$$|S| = k. \quad (8.63)$$

In general, the optimization problem in Eq. 8.60–8.63 is NP-hard since it requires searching all the possible subsets $S \in X$. Therefore, heuristic algorithms need to be used to find (possibly suboptimal) solutions to the problem. The most commonly used heuristic algorithms are based on greedy approximation, and they include:

- The *maximum relevance* method, which selects the most relevant features iteratively as follows:

Algorithm 5: Maximum Relevance heuristic.

Input: X, k

$S = \emptyset$

$iter = 1$

while $iter \leq k$ **do**

$S = S \cup \{\arg \max_{X_i \in X \setminus S} \underbrace{I(X_i; Y)}_{\text{relevance}}\}$

$iter = iter + 1$

Output: S

- The *minimum redundancy and maximum relevance* method [382], which selects the most relevant and least redundant features iteratively as follows:

Algorithm 6: Minimum Redundancy and Maximum Relevance heuristic.

Input: X, k

$S = \emptyset$

$iter = 1$

while $iter \leq k$ **do**

$S = S \cup \{\arg \max_{X_i \in X \setminus S} (\underbrace{I(X_i; Y)}_{\text{relevance}} - \frac{1}{|S|} \sum_{j \in S} \underbrace{I(X_i; X_j)}_{\text{redundancy}})\}$

$iter = iter + 1$

Output: S

The main difference between these two algorithms lies in the formulation of the objective function. While the maximum relevance algorithm extracts all the features, which contribute to maximize the relevance, the minimum redundancy and maximum relevance algorithm extracts only independent features (i.e., with minimum redundancy), which contribute to maximize the relevance.

8.4 Summary

In this chapter, we discussed feature extraction and feature selection. While both approaches aim to reduce the dimensionality of the features, i.e., p , in the original data matrix, they differ fundamentally in the way this is realized. For feature extraction, this is accomplished by transforming the original features into a lower-dimensional space. This results in synthetic (or new) features generated from the original features. Hence, the generated features contain information about all the original features to a certain degree. In contrast, feature selection performs a literal selection of a subset of features from the original features.

Feature extraction techniques such as PCA derive new synthetic features using a linear combination of the original ones. This process is not exempt of some loss of information, although the technique strive to minimize this loss. The derived synthetic features can have a high discrimination power and enable to control the overfitting problem. On the other hand, the synthetic features can be computationally expensive to obtain, and their interpretability is not obvious due to the synthetic nature of these new features.

The underlying problem for feature selection is NP-hard. Hence, the techniques available in the literature can only guarantee suboptimal solutions. On the other hand, since feature selection yields a subset of features from the original ones, there is no interpretability issue with the features, as opposed to feature extraction. This is

important in many contexts and applications such as genomics where the meaning of the features is important, e.g., when they are used as biomarkers.

Learning Outcome 8: Dimension Reduction

There are two conceptually different approaches to dimension reduction: feature extraction and feature selection. Both result in a low-dimensional representation of a data matrix, but the meaning of “features” is entirely different.

8.5 Exercises

1. Load the R built-in data set “decathlon” and carry out a PCA on the data set after removing the variables “Rank,” “Points,” and “Competition.”
 - a. Provide the summary of the PCA, and discuss the results.
 - b. Plot the scree plot, and identify the appropriate number principal components to be retained.
 - c. Plot the correlation graph between the first two principal components and the variables, and discuss the results.
 - d. Plot the graph representing the contribution of the variables in the construction of the principal components, and discuss the results.
 - e. Reconstruct the original data using all the principal components, and discuss the results.
 - f. Reconstruct the original data using the first four principal components, and discuss the results.
 - g. Reconstruct the original data using the first two principal components, and discuss the results.
2. Load the R built-in data set “decathlon” and carry out a NMF on the data set after removing the variables “Rank,” “Points,” and “Competition.”
 - a. Provide the summary of the NMF, and discuss the results.
 - b. Reconstruct the original data using all the features, discuss the results, and compare them with the results from 1.e.
 - c. Reconstruct the original data using the best four features, discuss the results, and compare them with the results from 1.f.
 - d. Reconstruct the original data using the best two features, discuss the results, and compare them with the results from 1.g.

Chapter 9

Classification



9.1 Introduction

In this chapter, we discuss classification methods. We start by clarifying what a classification means and what type of data are needed. Then we discuss aspects common to general classification methods. This includes also an extension of measures for binary decision-making to multiclass classification problems. As we will see, this extension is not trivial because the contingency table becomes multi-dimensional when conditioned on different classes.

There are many classification methods that have been developed in statistics and machine learning, making it impossible to provide a comprehensive coverage. For this reason, in this chapter, we selected six important and popular methods (namely, naive Bayes classifier, linear discriminant analysis, k-nearest neighbor classification, logistic regression, support vector machine, and decision tree) that provide a representative overview of the diverse ideas underlying classification methods widely used in many applications.

9.2 What Is Classification?

Classification is a *supervised learning* task. That means in addition to the data points, $X = \{x_1, \dots, x_n\}$ with $x_i \in \mathbb{R}^p$ and $p \in \mathbb{N}$, where n corresponds to the sample size and p to the number of features, information about the classes to which these data points belong to, y_i , is needed. Hence, to study a classification problem, paired data of the form $X = \{(x_1, y_1), \dots, (x_n, y_n)\}$ are required; see Fig. 9.1. It is important to highlight that the class labels, y_i , are *categorical variables*, because they just provide an indicator—a label—to name a certain class uniquely. For example, for a binary classifier, one could use the labels $y \in \{0, 1\}$, $y \in \{+1, -1\}$, or $y \in \{N, P\}$ to indicate the two classes. For the former choices, it is important to

Data type:

$X = \{((x_i, y_i))\}_{i=1}^n$ with $x_i \in \mathbb{R}^p$, $y_i \in \{l_1, l_2\}$ and l_1, l_2 are categorical variables

x_i is called a data point or feature vector

y_i is called a class label

p : number of features or number of variables

n : number of samples

Question addressed:

What class label should be assigned to a (new) data point?

Principles of major classification approaches:

Probability distribution-based classification $\implies$ estimation of **conditional probability distributions** to make probabilistic predictions for the class labels of new data points

Support Vector Machine $\implies$ mapping of the data points x_i into a **high-dimensional feature space** by utilizing the **kernel trick** allows the linear separation of these data points by hyperplanes

Decision Tree $\implies$ a **sequence of linear, binary decisions**, organized as a tree, leads to the classification of data points

Fig. 9.1 Overview of the classification problem with respect to the used data type, the question addressed, and the principal idea of methods discussed in this chapter

remember that the numbers do not have their usual meaning, e.g., that 0 is smaller than 1, i.e., $0 < 1$. Instead, they are merely used as labels (categorical variables) and not as numerical values.

The availability of the class labels provides valuable additional information that can be used in the learning process of a classification method. Specifically, when learning the parameters of a classification method, the error of this classifier can be quantified by means of the class labels. This can be seen as a “supervision” of the learning process, because a guided feedback can be evaluated and used to improve the classifier. This motivates the name “supervised learning.”

Before we discuss the different classification methods in detail, we review some aspects common to general classification problems.

9.3 Common Aspects of Classification Methods

9.3.1 Basic Idea of a Classifier

A classification method, also briefly called a classifier, is a prediction model, M , that makes a prediction about the class label of a data point x . That means

$$y' = M(x; \alpha), \quad (9.1)$$

where α corresponds to the (true) parameter(s) of the model and y' corresponds to the predicted class label. We distinguish predicted class labels y' from the true class labels y because the model can make errors.

The prediction results in the pair (x, y') that associates x with the class y' . Formally, the aim of a classifier M is to learn a mapping from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$ with

$$M : \mathcal{X} \rightarrow \mathcal{Y}. \quad (9.2)$$

9.3.2 Training and Test Data

For classification methods, it is important to distinguish the following two types of data sets: the training data, D_{train} , and test data, D_{test} .

The training data are used for learning the parameter(s) of the model given by α . That means the estimated model parameter(s) $\hat{\alpha}$ are a function of the training data

$$\hat{\alpha} = f(D_{\text{train}}). \quad (9.3)$$

Here, the “hat” indicates that the values of the parameters $\hat{\alpha}$ are estimates, which can be different from the true values given by α .

In contrast, test data are used to evaluate the prediction capabilities of a model by estimating an error measure

$$\hat{E} = g(M(x; \hat{\alpha}), D_{\text{test}}). \quad (9.4)$$

Here the function g corresponds to an error measure, e.g., accuracy or F-score (see Chap. 3). Since the estimated parameter(s) of the model $\hat{\alpha}$ are needed for this, the estimated error measure is a function of D_{train} and D_{test} .

Formally, supervised learning can be defined based on the definition of a domain and a task.

Definition 9.1 A *domain* $\mathcal{D}$ consists of a feature space, $\mathcal{X}$, and a marginal probability distribution, $P(X)$, where $X \in \mathcal{X}$, and it is given by $\mathcal{D} = \{\mathcal{X}, P(X)\}$.

Definition 9.2 A *task* $\mathcal{T}$ consists of an outcome space $\mathcal{Y}$ and a prediction function, $f(X)$, with $f : \mathcal{X} \rightarrow \mathcal{Y}$. Therefore, $\mathcal{T} = \{\mathcal{Y}, f(X)\}$.

The task provides a mapping from the feature space $\mathcal{X}$ into the outcome space $\mathcal{Y}$.

Before, we proceed, we would like to make a general remark. When we discuss model selection in Chap. 12 and the expected generalization error in Chap. 18, we will see that, in general, one needs to distinguish between training data, testing data, and validation data. However, in this chapter, we will focus on

individual classification methods, which require only model assessment but no model selection. That means the following setting serves an educational purpose allowing to focus on individual methods but does not suit real-world problems when one needs to choose among alternative classification models.

9.3.3 Error Measures

In order to evaluate a classifier, one needs to specify an error measure. In Chap. 3, we discussed already many error measures for binary decision-making, which corresponds to a two-class classification. We have seen that essentially all such error measures are based on the four fundamental errors, TP (true positives), TN (true negative), FP (false positives), and FN (false negatives). That means such measures are (nonlinear) functions of TP, TN, FP, and FN.

Due to the fact that a classifier makes a unique prediction for the class labels of input samples, the four fundamental errors are easy to estimate, for a given test data set, $D_{\text{Test}} = \{(x_1, y_1), \dots, (x_T, y_T)\}$, in the following way:

$$\text{TP} = \sum_i^T I(M(x_i) = y_i | y_i = +1) \quad (9.5)$$

$$\text{TN} = \sum_i^T I(M(x_i) = y_i | y_i = -1) \quad (9.6)$$

$$\text{FP} = \sum_i^T I(M(x_i) \neq y_i | y_i = -1) \quad (9.7)$$

$$\text{FN} = \sum_i^T I(M(x_i) \neq y_i | y_i = +1) \quad (9.8)$$

Here the function $I()$ is the so-called *indicator* function. It gives only two different values, either a 1 or a 0.

$$I(x) = \begin{cases} 1 & \text{if } x \text{ is true} \\ 0 & \text{if } x \text{ is false} \end{cases} \quad (9.9)$$

If the argument of the indicator function is “true,” it gives 1 otherwise 0. The separator “|” should be understood as “conditioned on” the argument on the right-hand side. For example, the explicit meaning of $I(M(x_i) = y_i | y_i = 1)$ is that if y is given and $y_i = 1$ holds, then for $M(x_i) = y_i$, the indicator function is 1. A moment of reflection will reveal the meaning of this evaluation, corresponding to a true positive prediction of the classifier. Hence, a summation over all elements i for $I(M(x_i) = y_i | y_i = 1)$ in the test data set gives the total number of true positive predictions (TP).

Table 9.1 Contingency table summarizing the results of a two-class classifier

Truth	Predicted	
	Class +1	Class -1
Class +1	TP	FN
Class -1	FP	TN

A convenient way to summarize these results for a two-class classification is by means of a contingency table or confusion matrix, shown in Table 9.1.

9.3.3.1 Error Measures for Multiclass Classification

In this chapter, we study also multiclass classification in addition to two-class classifiers. However, this requires an extension of error measures for binary decision-making. Specifically, there are two types of such extensions. The first type provides an error measure for all classes, while the second gives an error measure for each class, by providing information about “one class versus the remaining classes.”

An example for the first type of multiclass error measures is the accuracy (ACC). Specifically, in this case of multiclass classification, the accuracy (ACC) is defined by

$$\text{ACC} = \frac{1}{T} \sum_{i=1}^T I(M(x_i) = y_i) \quad (9.10)$$

where T is the total number of predictions. This is a simple generalization of the definition of accuracy for binary classification by maintaining its original meaning, i.e., the number of correct predictions is divided by the number of total predictions (see Chap. 3).

Examples of the second type of multiclass error measures are sensitivity, specificity, positive predictive value, and negative predictive value. Specifically, instead of defining a *global* error measure over all classes, as for the accuracy, the aforementioned measures are defined for each class separately. For a c -class classification problem, with $c > 2$, this means that one has, e.g., c sensitivity values (one for each class). The logic behind this is to define the four fundamental errors for “one against the rest.” For simplicity let’s assume we have a three-class classification problem with $c = 3$; however, an extension to more classes is straightforward.

Figure 9.2 (top) shows a numerical example of a contingency table (the numbers correspond to the results from a LDA classification, discussed in Sect. 9.5, but this is not important for our following discussion), and the following three tables show how the meaning of the cells changes with respect to the four fundamental errors when *conditioned* on different classes. That means a cell in a contingency table does not always correspond to, e.g., a TP, but this depends on the class designated as “positive.” Hence, this conditioning results in three contingency tables.

In order to emphasize the focus on the different classes, we highlighted one row and one column for each of these tables that have a succinct meaning for defining

Example:

		predicted outcome			
		Class 1	Class 2	Class 3	Total
actual outcome	Class 1	2767	217	16	b_1
	Class 2	110	2846	44	b_2
	Class 3	9	51	2940	b_3
Total		a_1	a_2	a_3	T

For class 1:

		predicted outcome			
		Class 1	Class 2	Class 3	Total
actual outcome	Class 1	TP	FN	FN	b_1
	Class 2	FP	TN	TN	b_2
	Class 3	FP	TN	TN	b_3
Total		a_1	a_2	a_3	T

For class 2:

		predicted outcome			
		Class 1	Class 2	Class 3	Total
actual outcome	Class 1	TN	FP	TN	b_1
	Class 2	FN	TP	FN	b_2
	Class 3	TN	FP	TN	b_3
Total		a_1	a_2	a_3	T

For class 3:

		predicted outcome			
		Class 1	Class 2	Class 3	Total
actual outcome	Class 1	TN	TN	FP	b_1
	Class 2	TN	TN	FP	b_2
	Class 3	FN	FN	TP	b_3
Total		a_1	a_2	a_3	T

Fig. 9.2 Contingency tables for a three-class classification. Top: Numerical results for a LDA classification (see Sect. 9.5). The remaining contingency tables show how the meaning of the cells change with respect to the four fundamental errors when conditioned on different classes

the four fundamental errors. Overall, each contingency table allows to estimate the sensitivity, specificity, positive predictive value, and negative predictive values in the usual way, as discussed in Chap. 3, however, with the meaning “one against the rest.”

For instance, for the numerical values in Fig. 9.2 (top), one obtains the following values for the sensitivity:

- Class 1: 0.92
- Class 2: 0.94
- Class 3: 0.98

Hence, for a c -class classification problem, the contingency table becomes multi-dimensional. Based on these class-specific errors, one can obtain a global error measure by summarizing the “local” errors. For instance, the global sensitivity (true positive rate (TPR)) is given by

$$\text{TPR}_{\text{global}} = \frac{1}{c} \sum_{i=1}^c \text{TPR}_i. \quad (9.11)$$

9.4 Naive Bayes Classifier

The first classification method we discuss is the naive Bayes classifier. The basic idea of a naive Bayes classifier is quite simple. This model is given by a conditional probability distribution $p(c|x)$ used to classify an instance x as follows:

$$c_p(x) = \operatorname{argmax}_{c \in C} \{p(c|x)\}. \quad (9.12)$$

That means for a given instance x , this classifier uses a conditional probability distribution to make a prediction for the class label of the data point x by selecting the class label with the maximum probability.

If one considers $p(c|x)$ as the posterior probability of the distribution, $p(x|c)$ (called likelihood; see Chap. 6), from which the samples are drawn, the name of this classifier becomes clear, and Eq. 9.12 can be written using Bayes's theorem (see Chap. 6) as follows:

$$c_p(x) = \frac{1}{p(x)} \operatorname{argmax}_{c \in C} \{p(x|c)p(c)\}. \quad (9.13)$$

9.4.1 Educational Example

In the following, we give a simple educational example to clarify the working mechanism of a naive Bayes classifier. First, we generate a training data set for two classes with labels 1 and 2. For each class, we draw n_i samples from a normal distribution with a mean of μ_i and a standard deviation of σ_i using Listing 9.1:

Listing 9.1: Simulating data for a classification

```
mu1 <- 1      # mean of class 1
sd1 <- 0.15   # standard deviation of class 1
mu2 <- 2
sd2 <- 0.35
n <- 10
n1 <- n
n2 <- n

x1 <- rnorm(n1, mu1, sd1)
x2 <- rnorm(n2, mu2, sd2)
df <- data.frame("x"=c(x1,x2),
  "y"=as.factor(c(rep(1,n1),rep(2,n2))))
```

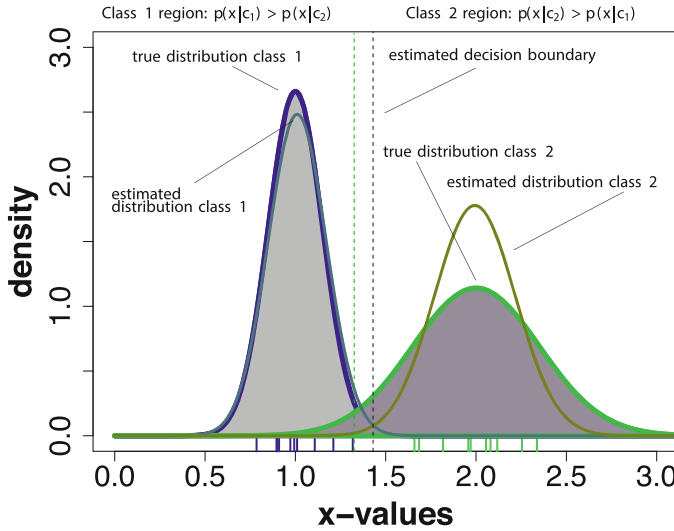


Fig. 9.3 True and estimated probability distributions for two classes from which data points are sampled, i.e., $x \sim p(c_i|x)$ with $i \in \{1, 2\}$. The estimated distributions are obtained from n_1 samples from class 1 and n_2 samples from class 2, as shown in the rug plot. The true decision boundary is shown as black vertical dashed line

Figure 9.3 shows the true distributions for class 1 (blue) and 2 (green) as simulated in Listing 9.1. In addition, the drawn samples n_1 and n_2 are shown as a rug plot (colored lines above the x -axis). It is important to note that the true distributions of both classes overlap, as indicated by the vertical dashed line (red); however, the samples drawn from these distributions are nicely separated. That means, on the left-hand side, first, come all samples from class 1 (blue) and then come all samples from class 2 (green).

A naive Bayes classifier learns a conditional probability distribution, i.e.,

$$p(x|c_i) = p(x|m_i, s_i), \quad (9.14)$$

for each class by assuming a normal distribution. That means the training samples from both classes are used separately to obtain maximum likelihood estimates for the mean and the standard deviation of a normal distribution that describes the data best. For this reason, in Eq. 9.14, we did not use the Greek symbols for the population mean and standard deviation, i.e., μ_i and σ_i respectively, but the sample estimates of the mean (given by m_i) and standard deviation (given by s_i) to indicate that these values are estimates from the training data.

In R, this can be accomplished by using the library *IklaR*, which provides the function `NaiveBayes()`, setting the option “usekernel=F.” The prior for the naive Bayes classifier can either be explicitly given, using the “prior” option, or the option

can be left unspecified, in which case the class proportions for the training set are used.

The numerical results for our sample data can be obtained by applying the function *predict()* to the estimated model with a certain data set as argument. Listing 9.2 shows the corresponding implementation using R. In this case, we are using the training data as the test data, called out-of-sample data; see Chap. 4. As a result, we obtain a perfect classification because the predicted classes correspond to the true classes for all the instances.

Listing 9.2: Results for the simulated data

```
# estimate model
> model <- NaiveBayes(y ~ x, data=df, usekernel=F)
> table(predict(model, df)$class, df[,2],
      dnn=list('predicted', 'true'))
      true
predicted  1  2
          1 10  0
          2  0 10
```

To understand the numerical outcome of this classification, we included in Fig. 9.3 the estimated distributions for classes 1 (light blue) and 2 (dark green), and we also added the estimated decision boundary as a vertical dashed line. As one can see, the estimated distributions are not identical to the true class distributions but provide reasonable estimates, especially, given the small number of samples used. Most importantly, the estimated decision boundary, which is directly obtained from the estimated distributions, separates perfectly samples from class 1 (left-hand side) from those in class 2 (right-hand side). For this reason, the classification is perfect for these data.

Descriptively, the decision boundary in Eq. 9.13 is given by the intersection point of the (estimated) distributions, enabling the classifier to predict class 1 to its left-hand side and class 2 for x values on its right-hand side. Formally, this can be written as

Predict class 1 if

$$p(x|c_1) > p(x|c_2);$$

Predict class 2 if

$$p(x|c_2) > p(x|c_1).$$

In Listing 9.3, we show how to estimate and visualize the probability distributions underlying a naive Bayes classifier.

Before we continue discussing further examples for the naive Bayes classifier, we would like to stop for a moment and reflect about the position of the decision boundary. Intuitively, if we would take into account only the position of the data points, any decision boundary between the last instance from class 1 and the first instance from class 2 would lead to the same numerical result of the classification

Listing 9.3: Visualization of the results, see Fig. 9.3

```

# get estimated class distributions
m1.e <- model$tables$x[1,1]
sd1.e <- model$tables$x[1,2]
m2.e <- model$tables$x[2,1]
sd2.e <- model$tables$x[2,2]
x.r <- seq(0, 4, 0.01)
d1.e <- dnorm(x.r, mean = m1.e, sd = sd1.e)
d2.e <- dnorm(x.r, mean = m2.e, sd = sd2.e)

# estimate decision boundary
ind <- max(which(d1.e>d2.e))
x.b <- x.r[ind]

# plot
d1 <- dnorm(x.r, mean = m1, sd = sd1)
d2 <- dnorm(x.r, mean = m2, sd = sd2)
plot(x.r, d1, xlim=c(0,3), ylim=c(0,3), lwd=5, col="blue",
main="", xlab="x values", ylab="density", cex.lab=2.0,
cex.axis = 1.8, font.lab = 2, type="l")
rug(x1, col="blue", lwd=2)
lines(x.r, d2, lwd=5, col="purple")
rug(x2, col="purple", lwd=2)
polygon(x.r, d1, col="gray90", border="blue", lwd=5)
polygon(x.r, d2, col="gray70", border="purple1", lwd=5)
lines(x.r, d1.e, lwd=3, col="skyblue3")
lines(x.r, d2.e, lwd=3, col="violetred3")
abline(v=x.b, lty=2)

```

for the above data. However, application of this strategy would in general not lead to an optimal decision boundary for *unseen data*. The latter point is very important because the goal of any classifier is to utilize a training sample to learn its parameters as good as possible so that unseen data (test data) can be classified optimally. This extrapolation of the training data toward test data, to obtain optimal classification results, is practically realized by a naive Bayes classifier by estimating the parameters of (normal) distributions instead of devising a learning rule directly based on the data points.

9.4.2 Example

In the following, we investigate the behavior of the accuracy (A), the precision (P), and the recall (R) values of a naive Bayes classifier as functions of the distance between the mean values of the distributions of the two classes. Specifically, for the true normal distributions of class 1 and 2, we assume the parameters $\mu_1 = 1$, $\sigma_1 = 0.35$ and $\mu_2 = f \times \mu_1$, $\sigma_2 = 0.25$. Here, f is a positive parameter that allows to increase the distance between the mean of class 1 and class 2. In the following, this parameter can assume the values $f \in \{1, 1.1, 1.2, 1.5, 2\}$. For $f = 1$ the mean

of both classes is identical, i.e., $\mu_1 = \mu_2 = 1$, and for all other values of f , the distance increases up to $\mu_2 = 2$ for $f = 2$.

In Listing 9.4, we evaluate the accuracy (A), the precision (P), and the recall (R) values of a naive Bayes classifier for every value of f . To do this, we generate independent test data of size $N_1^t = 1000$ and $N_2^t = 3000$. For the corresponding training data, we generate much smaller data sets of size $N_1 = 30$ and $N_2 = 20$. The results are shown in Fig. 9.4.

Listing 9.4: Example

```
N1 <- 30; N2 <- 20 # sample size
f <- c(1, 1.1, 1.2, 1.5, 2)
R <- length(f)
AC.res <- vector(mode="numeric", length=R)
P.res <- vector(mode="numeric", length=R)
R.res <- vector(mode="numeric", length=R)

# parameter of the true distributions
m1 <- 1; sd1 <- 0.35; m2 <- 1; sd2 <- 0.25
N1.t <- 1000; N2.t <- 3000

for(i in 1:R){
  # test data
  x1 <- rnorm(N1.t, m1, sd1); x2 <- rnorm(N2.t, f[i]*m2, sd2)
  df.test <- data.frame("x"=c(x1,x2),
    "y"=as.factor(c(rep(1,N1.t),rep(2,N2.t))))

  # training data
  x1 <- rnorm(N1, m1, sd1); x2 <- rnorm(N2, f[i]*m2, sd2)
  df <- data.frame("x"=c(x1,x2),
    "y"=as.factor(c(rep(1,N1),rep(2,N2))))

  # estimate model and determine error
  model <- NaiveBayes(y ~ x, data=df, usekernel=F)
  res <- table(predict(model, df.test)$class, df.test[,2])
  TP <- res[1,1]; FN <- res[2,1]
  FP <- res[1,2]; TN <- res[2,2]
  AC.res[i] <- (TP+TN)/(N1.t+N2.t)
  P.res[i] <- TP/(TP+FP); R.res[i] <- TP/(TP+FN)
}

par(mar=c(5,5,1,1))
plot(f, AC.res, lwd=6, col="blue", ylim=c(0,1), cex.lab=2.0,
  cex.axis = 1.8, font.lab = 2, type="b", ylab="A, P and R")
lines(f, P.res, lwd=6, col="green4", type="b", pch=4)
lines(f, R.res, lwd=6, col="red3", type="b", pch=6)
legend("bottomright", c("accuracy (A)", "precision (P)",
  "recall (R)"), pch = c(1,4,6), lty = c(1,1,1), col=c("blue",
  "green4", "red3"), text.font=2, cex=1.4)

> res
      true
predicted 1 2
1      931 106
2       69 2894
```

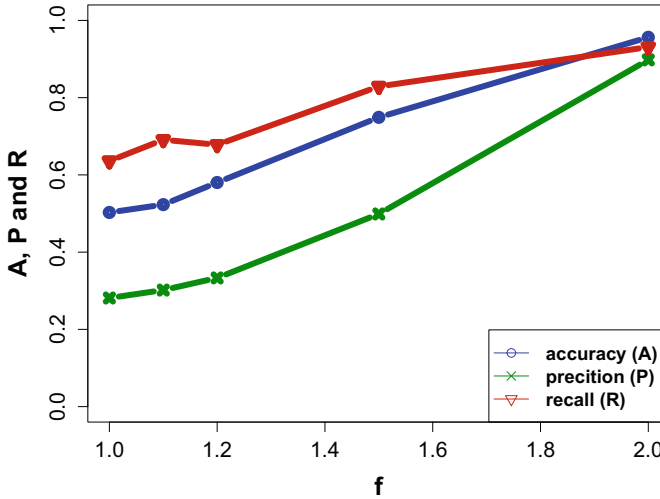


Fig. 9.4 Accuracy, precision, and recall values of a naive Bayes classifier against the distance between the mean of the two classes

One can see that with an increasing distance between the mean values of the class distributions, the accuracy (A), the precision (P), and the recall (R) values increase. This is expected, because the farther the distance, the easier the classification problem (see also Fig. 9.3). For $f = 2$ the obtained error measures assume almost perfect values (close to one), meaning that a further increase of the distance would not lead to a further improvement in the classification performance. This allows to make statements about a saturating convergence of the classification method.

Interestingly, for $f = 1$, the values of A, P, and R are not “equally bad,” but the recall values are quite high compared to A and P. This is a reflection of the nonlinear dependency of the error measures on the four fundamental errors, TP, TN, FP, and FN (see Chap. 3) and provides another argument why there is not just one error measure that is important but different error measures complement each other (see the discussion in Chap. 3). This is actually true not only for a naive Bayes classifier but also for any other classifier or statistical method for which TP, TN, FP, and FN can be obtained.

9.5 Linear Discriminant Analysis

A classification method that is related to a naive Bayes classifier is linear discriminant analysis (LDA). This method also estimates conditional probability distributions corresponding to normal distributions,

$$p(x|y = i) = \mathcal{N}(x|\mu_i, \Sigma_i) \quad (9.15)$$

$$= \frac{1}{(2\pi)^{p/2} |\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i)\right), \quad (9.16)$$

similar to the naive Bayes classifier. However, it additionally assumes that the covariance matrices for all classes are identical, i.e., $\Sigma_i = \Sigma$, for all i .

This constraint of the covariance matrices can be utilized to derive a simplified criteria for identifying a MAP solution given by

$$c_p(x) = \operatorname{argmax}_{c \in C} \{p(c|x)\}. \quad (9.17)$$

Applying Bayes theorem to Eq. 9.16 and taking the logarithm lead to

$$\delta_i(x) = \log p(y = i|x) = \log p(x|y = i) + \log p_i \quad (9.18)$$

$$= -\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) - \log(2\pi)^{p/2} |\Sigma_i|^{1/2} + \log p_i \quad (9.19)$$

$$= x^T \Sigma_i^{-1} \mu_i - \frac{1}{2} \mu_i^T \Sigma_i^{-1} \mu_i + \log p_i \quad (9.20)$$

$$\underbrace{-\frac{1}{2} x^T \Sigma_i^{-1} x - \log(2\pi)^{p/2} |\Sigma_i|^{1/2}}_{\text{this term is identical for all } \delta_i(x)} \quad (9.21)$$

Here, p_i is the prior for class i . Since LDA assumes $\Sigma_i = \Sigma$ for all classes, only the terms containing an μ_i and p_i are different for different classes. This leads to the simplified linear discriminant function given by

$$\delta_i(x) = x^T \Sigma_i^{-1} \mu_i - \frac{1}{2} \mu_i^T \Sigma_i^{-1} \mu_i + \log p_i \quad (9.22)$$

from which the MAP class predictions are obtained as follows:

$$c = \operatorname{argmax}_{i \in C} \{\delta_i(x)\}. \quad (9.23)$$

As an example of a LDA, let's study a three-class classification problem. For this, we simulate training and test data drawn from two-dimensional normal distributions with mean μ_i and a common covariance matrix Σ . Listing 9.5 and 9.6 shows how these data are simulated, and in Fig. 9.5 (top left and top right), we show a visualization of the training data for the three classes. For the training data, we generated $N_1 = 30, N_2 = 50, N_3 = 40$ samples, and for the test data $N_1 = N_2 = N_3 = 3000$.

Listing 9.5: Simulation of training data for the LDA, see Fig. 9.5 (top)

```
# training data
N1 <- 30; N2 <- 50; N3 <- 40
m1 <- c(1,1); sigma1 <- matrix(c(1,0.3,0.3,1), nrow=2, ncol=2,
  byrow=F)
m2 <- c(4,3); sigma2 <- matrix(c(1,0.3,0.3,1), nrow=2, ncol=2,
  byrow=F)
m3 <- c(2,6); sigma3 <- matrix(c(1,0.3,0.3,1), nrow=2, ncol=2,
  byrow=F)
x1 <- rmvnorm(N1, mu = m1, sigma = sigma1)
x2 <- rmvnorm(N2, mu = m2, sigma = sigma2)
x3 <- rmvnorm(N3, mu = m3, sigma = sigma3)
x1.train <- x1; x2.train <- x2; x3.train <- x3

cl.train <- c(rep(1, N1), rep(2, N2), rep(3, N3))
df.train <- data.frame("x"=rbind(x1, x2, x3),
  "y"=as.factor(cl.train))
```

In Fig. 9.5 (top right), we highlight the mean values of the normal distributions to see how far apart are the centers of mass from each other. Furthermore, one can see that the overlap of the three distributions is moderate, i.e., the mixing of data points from different classes is limited.

Listing 9.6: Simulation of test data for the LDA

```
N1 <- 3000; N2 <- 3000; N3 <- 3000
m1 <- c(1,1); sigma1 <- matrix(c(1,0.3,0.3,1), nrow=2, ncol=2,
  byrow=F)
m2 <- c(4,3); sigma2 <- matrix(c(1,0.3,0.3,1), nrow=2, ncol=2,
  byrow=F)
m3 <- c(2,6); sigma3 <- matrix(c(1,0.3,0.3,1), nrow=2, ncol=2,
  byrow=F)
x1 <- rmvnorm(N1, mean = m1, sigma = sigma1)
x2 <- rmvnorm(N2, mean = m2, sigma = sigma2)
x3 <- rmvnorm(N3, mean = m3, sigma = sigma3)

cl.test <- c(rep(1, N1), rep(2, N2), rep(3, N3))
df.test <- data.frame("x"=rbind(x1, x2, x3),
  "y"=as.factor(cl.train))
```

In R, the package *MASS* provides the function *lda()*, which implements the LDA classifier. Here, “df.train” is a data frame that contains the training data generated using Listing 9.5, and the column of this data frame named “y” contains the class labels corresponding to the two-dimensional data points, x , provided in the first two columns.

Listing 9.7 contains also the predictions of the model for the test data. It is important to note that the data frame of the test data needs to have the same column names as “df.train,” corresponding to the names of the $p = 2$ dimensional predictor variables.

Listing 9.7: Analysis of the LDA

```

library("MASS")
library("caret")

model <- lda(y~., df.train)
pred <- predict(model, df.test)

conf <- table(list(predicted=pred$class, observed=cl.test))
confusionMatrix(conf)
# output
Confusion Matrix and Statistics

      observed
predicted  1    2    3
  1 2767 110    9
  2  217 2846   51
  3   16   44 2940

Overall Statistics

               Accuracy : 0.9503
                95% CI : (0.9456, 0.9547)
   No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2.2e-16

               Kappa : 0.9255

  McNemar's Test P-Value : 3.628e-08

Statistics by Class:

               Class: 1 Class: 2 Class: 3
Sensitivity      0.9223   0.9487   0.9800
Specificity      0.9802   0.9553   0.9900
Pos Pred Value   0.9588   0.9139   0.9800
Neg Pred Value   0.9619   0.9738   0.9900
Prevalence       0.3333   0.3333   0.3333
Detection Rate   0.3074   0.3162   0.3267
Detection Prevalence 0.3207   0.3460   0.3333
Balanced Accuracy 0.9512   0.9520   0.9850

```

We would like to note that since we are analyzing a three-class classification problem, for the assessment, we need to use the error measures discussed in Sect. 9.3.3.1, which assess “one class against the rest.” As one can see, the LDA results in a very good classification, and the error measures for the individual classes are quite homogeneous.

In Fig. 9.5 (bottom-row), we show level plots for the decision boundaries for the training data (bottom left) and the posterior probabilities for the class predictions (bottom right). From this visualization, one can see why the LDA is a linear classifier, because the decision boundaries between two classes are linear lines. In a later section of this chapter, we will see that there are other classification methods

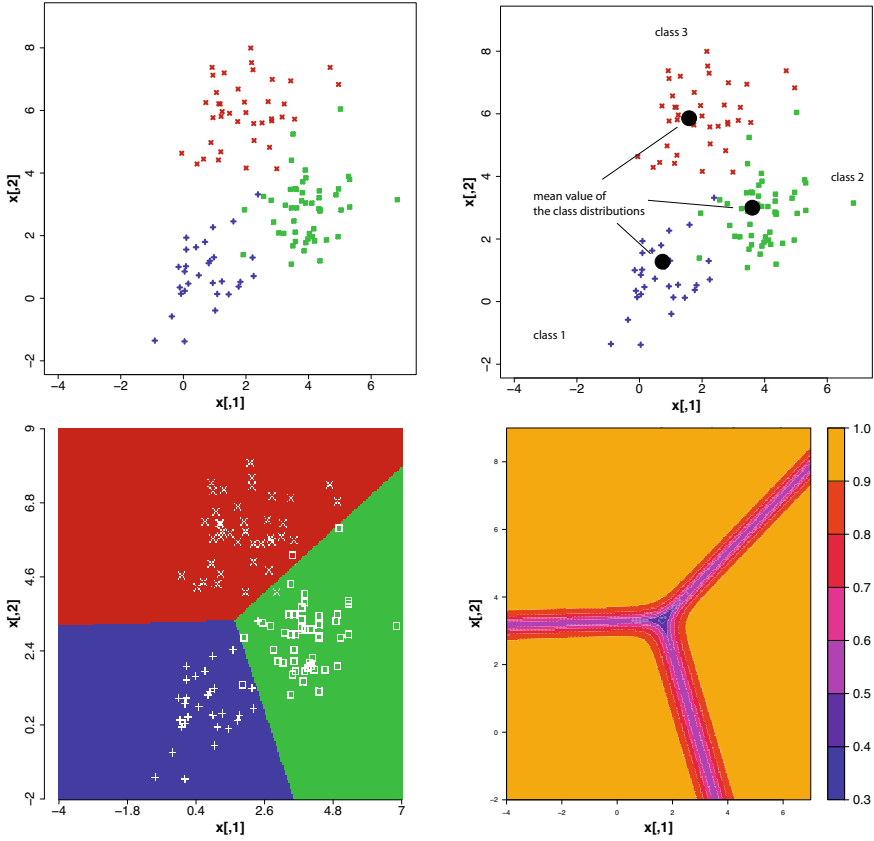


Fig. 9.5 Linear discriminant analysis (LDA). Top left: Distribution of the training data. Top right: Explanation of the estimation of the conditional probability values for each point. Bottom left: Decision boundaries obtained for the training data. Bottom right: Visualization of the corresponding probabilities for the decisions on the left side

allowing lacerated boundaries (see Fig. 9.8) to deal with more overlapping data points from different classes.

9.5.1 Extensions

As discussed above, the LDA assumes that the covariance matrices of all classes are identical, $\Sigma_i = \Sigma$. If we do not make this assumption, the linear discriminant function in Eq. 9.22 becomes nonlinear, because the quadratic terms in x are no longer identical for the different classes. Hence, allowing arbitrary covariance matrices leads to the quadratic discriminant function

$$\delta_i(x)' = -\frac{1}{2}\log|\Sigma_i| - \frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1}(x - \mu_i) + \log p_i \quad (9.24)$$

$$= \delta_i(x) - \frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1}(x - \mu_i). \quad (9.25)$$

Further extensions are possible and allow, e.g., mixtures of Gaussians or use non-parametric density estimates. In general, such methods are called Gaussian discriminant analysis.

9.6 Logistic Regression

The logistic regression method is a member of a larger family of methods called *generalized linear models (GLMs)*, which are discussed in Chap. 11. For this reason, we present here only the underlying idea of logistic regression and discuss further details in Sect. 11.6.4.4.

In contrast with common regression methods, logistic regression is used for binary data and, hence, can be used for classification. Binary data means that the response variable Y can assume two values, e.g., 1 and 0. The idea of logistic regression is similar to the previous methods because logistic regression also aims to estimate a conditional probability distribution. Specifically, a logistic regression model estimates

$$p(Y = 1|x) \quad (9.26)$$

providing a probability for $Y = 1$ given an input x . Due to the binary nature of the response, from this estimate, it follows that

$$p(Y = 0|x) = 1 - p(Y = 1|x). \quad (9.27)$$

Definition 9.3 The probability $p(Y = 1|x)$ corresponds to the proportion of responses giving $Y = 1$ for an input x . For brevity, we write

$$p(x) = p(Y = 1|x). \quad (9.28)$$

Let's visualize this definition for an example. In Listing 9.8, we show the first few lines of the data from the Michelin Guides for restaurants in New York. In order to fit each data line into one line, we skipped a few columns, which in the following we do not need anyway. Briefly, the data provide some information about whether a restaurant is recommended by the Michelin Guides ("InMichelin"). For the following analysis, we use "InMichelin" as the response variable, which can be either "Yes" or "No." The decision for including a restaurant in the Michelin Guides

is based on a number of covariates of which four are numerical (“Food,” “Decor,” “Service,” and “Cost”) and one is categorical (“Cuisine”). The numerical covariates provide scores for the corresponding categories, e.g., for service or food, whereas the categorical variable labels a restaurant with respect to the type of food provided. To discuss the idea of logistic regression, it is sufficient to restrict our analysis to one covariate. For our analysis, we select the “Food” score.

Listing 9.8: Michelin guide for New York restaurants

```
library("marginalmodelplots")
data(MichelinNY)
MichelinNY[1:5, c(1, 4, 5, 6, 7, 8)]
# output
```

	InMichelin	Cuisine	Food	Decor	Service	Cost
1	No	French (New)	19	20	19	50
2	No	French (New), Italian	17	17	16	43
3	No	French	23	17	21	35
4	Yes	American (New), French	19	23	16	52
5	No	Caribbean, French (Bistro)	23	12	19	24

In Table 9.2, we show summary data provided by the Michelin Guides, specifically, for obtaining the count data shown, e.g., how many restaurants get a food score of 15. Then we counted how many of these restaurants are recommended by the Michelin Guides (“InMichelin”). From these values, we estimate the proportion of recommended restaurants given by

$$\text{prop}(\text{restaurants in Michelin guide} | \text{food score}) = \frac{\text{InMichelin}}{m}. \quad (9.29)$$

For this estimate, it is important to highlight that the proportion has a conditional dependency on the food score.

In Fig. 9.6, we visualize the data in Listing 9.8. Specifically, the histograms in the top figure correspond to the response variable “Yes” (green histogram) and “No” (blue histogram) for the corresponding food scores. These food scores are also shown in the boxplots at the bottom of the figure. These are the raw data from Listing 9.8. In contrast, the (black) points in Fig. 9.6 correspond to the estimated values of the proportion in Listing 9.8. In order to emphasize the fact that the proportion has been estimated, we call it sample proportion (in contrast with the population proportion). Overall, for the estimated proportion, we can observe a tendency toward increasing values for increasing food scores (Fig. 9.6).

Formally, a logistic regression model is given by

$$\log \frac{p(x)}{1 - p(x)} = \beta_0 + \beta_1 x. \quad (9.30)$$

Table 9.2 Summary data from the Michelin guides for New York restaurants

	InMichelin	Food score	m	Prop
1	0	15.00	1	0.00
2	0	16.00	1	0.00
3	0	17.00	8	0.00
4	2	18.00	15	0.13
5	5	19.00	18	0.28
6	8	20.00	33	0.24
7	15	21.00	26	0.58
8	4	22.00	12	0.33
9	12	23.00	18	0.67
10	6	24.00	7	0.86
11	11	25.00	12	0.92
12	1	26.00	2	0.50
13	6	27.00	7	0.86
14	4	28.00	4	1.00

The term on the left-hand-side, i.e., $\log \frac{p(x)}{1-p(x)}$, is called logit or log odds. The latter name comes from the fact that

$$\text{odds}(p) = \frac{p}{1-p}. \quad (9.31)$$

Hence, Eq. 9.30 can also be written as

$$\log(\text{odds}(p(x))) = \text{logit}(p(x)) = \beta_0 + \beta_1 x. \quad (9.32)$$

The odds in Eq. 9.31 can assume values between 0 and ∞ , i.e.,

$$\text{odds} : [0, 1] \rightarrow [0, \infty). \quad (9.33)$$

However, taking the logarithm, the logit can assume values between $-\infty$ and ∞ . This makes sense because the regression term on the right-hand side is unbound.

Solving for $p(x)$, in Eq. 9.30, gives

$$p(x) = \frac{\exp(\beta_0 + \beta_1 x)}{1 + \exp(\beta_0 + \beta_1 x)} = \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x))} \quad (9.34)$$

$$= S(\beta_0 + \beta_1 x) \quad (9.35)$$

Here, S is the logistic function, which is an example of a sigmoid function describing a “S”-shaped curve. The logistic function assumes values between 0 and 1, i.e.,

$$\text{logistic function } S : (-\infty, \infty) \rightarrow [0, 1]. \quad (9.36)$$

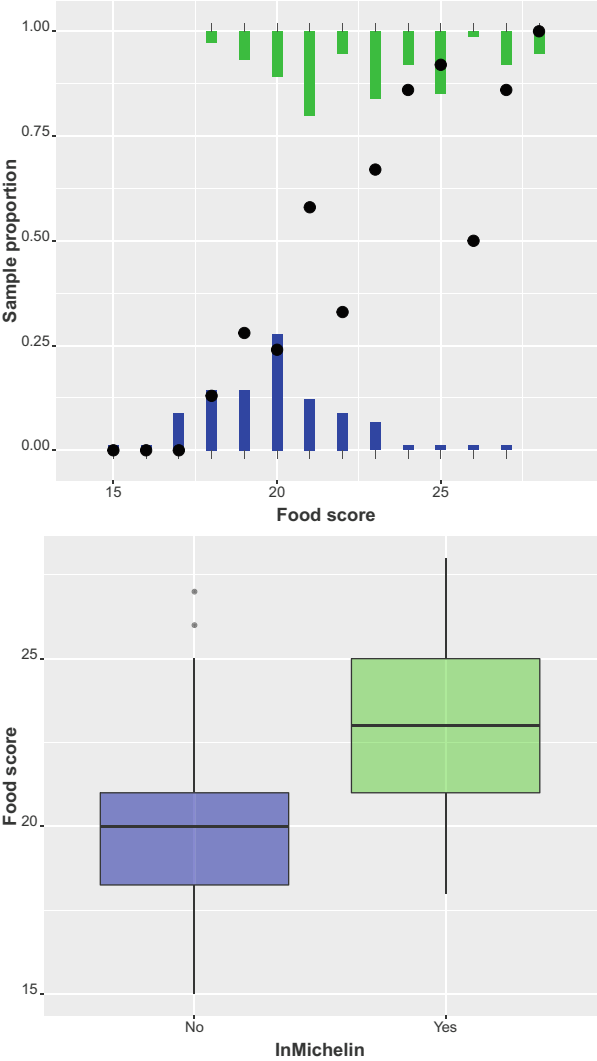


Fig. 9.6 Visualization of the Michelin data. Top figure: Estimated proportions of the response variable. Bottom figure: Boxplots of the food scores in dependence on the recommendations given in the Michelin guides

This is great since the probability on the left-hand side assumes values between 0 and 1.

In Listing 9.9, we fit the logistic regression model to our data and the results are shown in Fig. 9.7. To use the estimates of the model, given by $\hat{\beta}_0$ and $\hat{\beta}_1$, to predict the class of a new instance x , we proceed as follows. First we calculate $p(x)$, given by

Listing 9.9: Logistic regression for the Michelin guide

```
df <- data.frame(MichelinNY)
res.logistic <- glm(InMichelin ~ Food, data=df,
  family=binomial(link=logit))
summary(res.logistic)
# output
Call:
glm(formula = InMichelin ~ Food, family = binomial(link = logit),
  data = df)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.3484  -0.8555  -0.4329   0.9028   1.9847

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -10.84154    1.86234  -5.821 5.83e-09 ***
Food          0.50124    0.08767   5.717 1.08e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 225.79  on 163  degrees of freedom
Residual deviance: 175.73  on 162  degrees of freedom
AIC: 179.73

Number of Fisher Scoring iterations: 4
```

$$p(x) = \frac{\exp(\hat{\beta}_0 + \hat{\beta}_1 x)}{1 + \exp(\hat{\beta}_0 + \hat{\beta}_1 x)}. \quad (9.37)$$

Then, we need to define a decision boundary. For this, we define

$$\text{class}_{LR}(x) = \begin{cases} 0, & \text{if } p(x) \leq 0.5, \\ 1, & \text{if } p(x) > 0.5. \end{cases} \quad (9.38)$$

Hence, based on the definition of $\text{class}_{LR}(x)$, we can make a class prediction for every new instance x .

9.7 k-Nearest Neighbor Classifier

The next classification method we discuss is the k-nearest neighbor (KNN).

In this section, we give the basic idea of k-nearest neighbor classification. From a given training data set, each data point can be represented by a set of variables. So, the data points are plotted in a high-dimensional vector space; the axes in the space

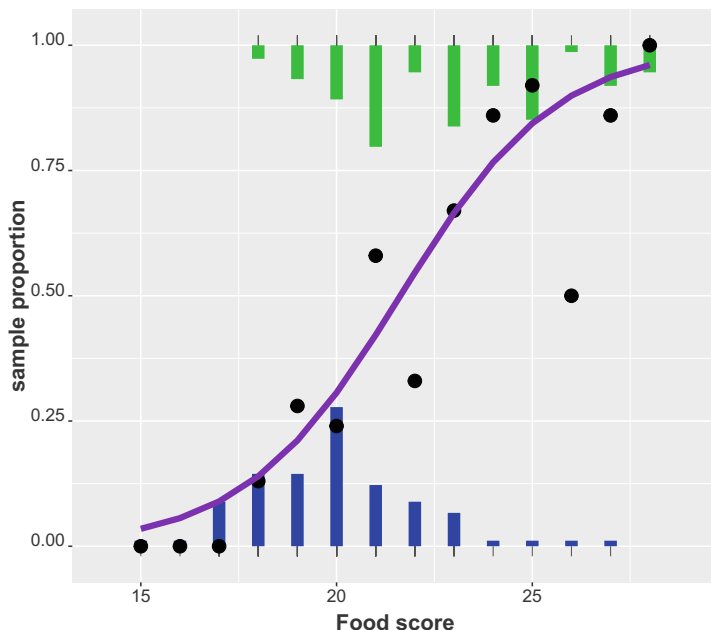


Fig. 9.7 Visualization of the Michelin data and the estimated proportions

correspond to the variables under consideration. Also, we assume that we have a fixed number of classes, which are usually represented by colors, e.g., in a two-dimensional space. Now, a new data point from a test data set can be classified by determining the k nearest neighbors that are most similar (i.e., minimum distance) to this test point. In order to classify this point, we need to use existing similarity or distance measures, which have been discussed in Chap. 7. It is clear that the result of the classification depends on the selected distance/similarity measure. Another question relates to find an appropriate value for k . It is well-known that $\sqrt{N}$ is a good choice for k , where N is the number of data points.

We start with an educational example to understand the k -nearest neighbor classification. In Fig. 9.8 (top-left), we show the training data for three classes generated with the following code: For each class $i \in \{1, 2\}$, we assume an underlying two-dimensional normal distribution with mean μ_i and covariance matrix Σ_i . Class 3 is generated using a random process of selecting either $\mathcal{N}(\mu_{3,1}, \Sigma_3)$ or $\mathcal{N}(\mu_{3,2}, \Sigma_3)$. The results of this random process for class 3 are that this class is separated by class 1 and 2, as one can see in Fig. 9.8 (top left).

We use these training data to visualize the numerical estimation of the conditional probabilities

$$p(y = c|x) = \frac{N(c, x)}{k} = \frac{1}{k} \sum_{i \in N_e(x)} I(y_i = c). \quad (9.39)$$

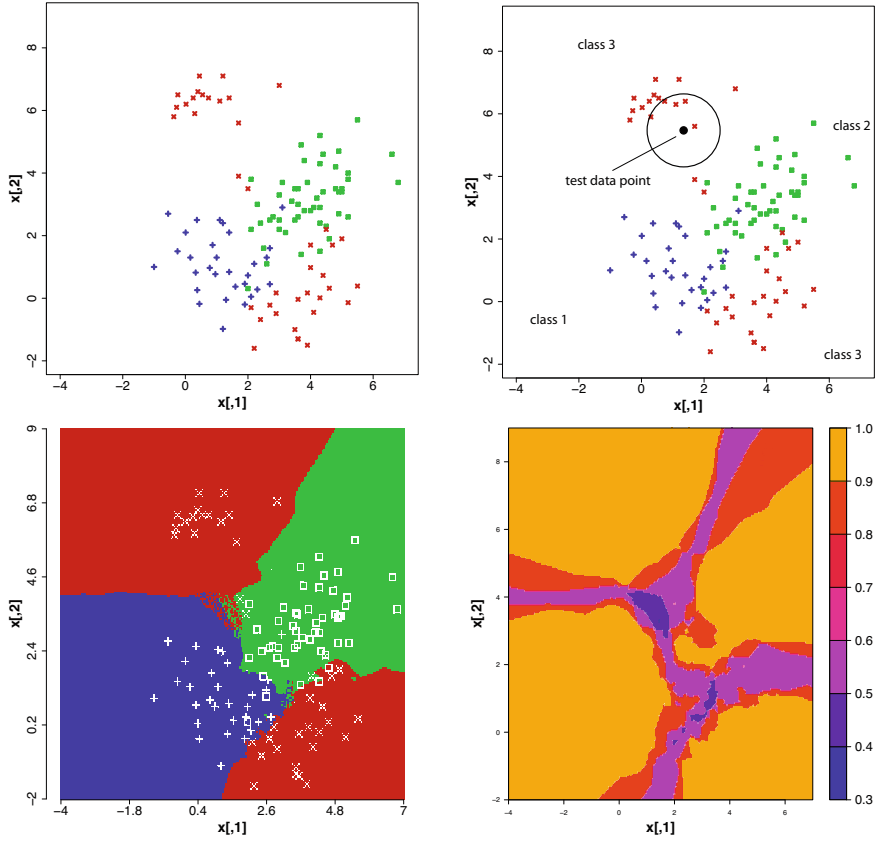


Fig. 9.8 K-nearest neighbor classification with $k = 5$. Top left: Distribution of the training data. Top right: Explanation of the estimation of the conditional probability values for each point. Bottom left: Decision boundaries obtained for these training data. Bottom right: Visualization of the corresponding probabilities for the decisions on the left side

In Fig. 9.8 (top-right), we show the training data plus one additional test data point (in black). Around this test data point is a circle including exactly $k = 5$ training data points corresponding to a KNN classifier with 5 neighbors. Within this circle we observe

$$\text{\#red instances - corresponding to class 3} = 5 \quad (9.40)$$

$$\text{\#blue instances - corresponding to class 1} = 0 \quad (9.41)$$

$$\text{\#green instances - corresponding to class 2} = 0 \quad (9.42)$$

Listing 9.10: Training data for a KNN classifier, see Fig. 9.8

```

N1 <- 30; N2 <- 50; N3 <- 40
m1 <- c(1,1); sigma1 <- diag(length(m1))
m2 <- c(4,3); sigma2 <- matrix(c(1,0.3,0.3,1), nrow=2, ncol=2,
  byrow=F)
m3.1 <- c(1,6); m3.2 <- c(4,0); sigma3 <- diag(length(m3.1))
x1 <- rmvnorm(N1, mu = m1, sigma = sigma1)
x2 <- rmvnorm(N2, mu = m2, sigma = sigma2)
x3 <- matrix(0, nrow=N3, ncol=2)
for(i in 1:N3){
  coin <- round(runif(1))
  if(coin<0.5){ x3[i,] <- rmvnorm(1, mu = m3.1, sigma = sigma3)}
  else{ x3[i,] <- rmvnorm(1, mu = m3.2, sigma = sigma3)}
}
x1.train <- x1; x2.train <- x2; x3.train <- x3
df.train <- data.frame("x"=rbind(x1, x2, x3))
cl.train <- c(rep(1, N1), rep(2, N2), rep(3, N3))

par(mar=c(5,5,1,1))
plot(x1, col="blue", lwd=5, xlim=c(-4,7), ylim=c(-2,9),
  xlab="x[,1]", ylab="x[,2]", cex.lab=2.0, cex.axis = 1.8,
  font.lab = 2, pch=3)
points(x2, col="green3", lwd=5, pch=7)
points(x3, col="red3", lwd=5, pch=4)

```

leading to the following estimates for the conditional probabilities:

$$p(y = 1|x) = 0, \quad (9.43)$$

$$p(y = 2|x) = 0, \quad (9.44)$$

$$p(y = 3|x) = 1. \quad (9.45)$$

It is important to note that the diameter, d , of the circle is not fixed for every test data point but it is determined by the k neighbor training data points because every circle needs to contain exactly k training data points. Furthermore, the shape of the neighborhood around a test data point is given by a circle because the distance between data points is measured using the Euclidean metric.

In order to analyze the KNN classifier, we generate a test data set using a normal distributions with the same parameters as in Listing 9.10. For this configuration, we generate $N_1 = N_2 = N_3 = 3000$ samples.

In R, the package *class* provides the function *knn()* with the following options. The first argument is the training data set, “df.train.”. It assumes the form of a matrix or a data frame, where each row corresponds to one sample and the number of columns corresponds to the number of the predictor variables (features). The second argument is another matrix or data frame containing the test data set. The third argument is a vector giving the class labels of the training data. The option “k” sets the number of neighbors to be considered. The function *knn()* returns the predictions for each sample in “df.test”, corresponding to the class labels. Furthermore, by setting “prob=TRUE”, the probability values corresponding to the predictions, as defined in Eq. 9.39, are returned.

Listing 9.11: Analysis of the KNN classifier, see Fig. 9.8 (bottom)

```

library("class")
pred <- knn(df.train, df.test, cl.train, k = 5, prob=TRUE)

conf <- table(list(predicted=pred, observed=cl.test))
confusionMatrix(conf)
# output
Confusion Matrix and Statistics

      observed
predicted  1    2    3
      1 2676   88 109
      2  194 2755 162
      3  130  157 2729

Overall Statistics

                Accuracy : 0.9067
                  95% CI : (0.9005, 0.9126)
    No Information Rate : 0.3333
      P-Value [Acc > NIR] : < 2.2e-16

                Kappa : 0.86

  McNemar's Test P-Value : 4.495e-09

Statistics by Class:

               Class: 1 Class: 2 Class: 3
Sensitivity    0.8920    0.9183    0.9097
Specificity    0.9672    0.9407    0.9522
Pos Pred Value 0.9314    0.8856    0.9048
Neg Pred Value 0.9471    0.9584    0.9547
Prevalence     0.3333    0.3333    0.3333
Detection Rate 0.2973    0.3061    0.3032
Detection Prevalence 0.3192    0.3457    0.3351
Balanced Accuracy 0.9296    0.9295    0.9309

```

From Listing 9.11, one can see that the obtained results for accuracy, sensitivity, specificity, etc. are quite good.

In Fig. 9.8 (bottom-left), we visualize the decision boundaries of the KNN classifier learned from the training data, and in Fig. 9.8 (bottom-right), the probability values corresponding to these decisions are shown. First, it is important to note that in contrast with the classifiers we have discussed so far, the observed decision boundaries are not straight lines or concentric circles. Instead, they can assume any shape. This is due to the non-parametric character of the KNN classifier, because no assumptions are made about the conditional probability distributions in Eq. 9.39 to limit these to a certain family of probability distributions. Instead, the conditional probability distributions are estimated numerically from the training data and, hence, are very flexible.

From Fig. 9.8 (bottom-left), one can see that there are several regions that change many times the class labels. These are regions having a tie for different classes, i.e., the same number of votes for more than one class. In general, there are multiple ways ties can be broken. For the analysis above, we extended the neighborhood successively by one, until a unique maximal vote is reached. For the function *knn()*, this can be done by setting the option “use.all=TRUE,” which is the default option of this function. These “tie regions” are also easily to spot in Fig. 9.8 (bottom right) by having the lowest probabilities (blue values) for the decisions.

9.8 Support Vector Machine

The next classification method we discuss is called support vector machine (SVM). Originally developed by Vladimir Vapnik and Alexey Chervonenkis in the 1960s for linearly separable data, the “kernel trick” introduced in the 1990s allowed to extend this framework to nonlinear problems [51]. We will see that a SVM is quite different from the other classification methods discussed so far, because a SVM is not a probabilistic classifier, which aims to estimate a conditional probability distribution. Instead, a SVM aims to estimate optimal separating hyperplanes in a transformed feature space.

We start our discussion of SVMs by first introducing its underlying idea and motivation. Thereafter, we turn to a more mathematical formulation and discuss the extensions.

9.8.1 Linearly Separable Data

Suppose that we have a binary classification problem and the training data are linearly separable. Then we can formulate this problem by linear equations assuming the following form:

$$w \cdot x_i + b > 0 \text{ for } y_i = +1 \quad (9.46)$$

$$w \cdot x_i + b < 0 \text{ for } y_i = -1 \quad (9.47)$$

Here x_i are data points in $\mathbb{R}^p$, w is the normal vector of the hyperplane, and b is the shift vector. Since we assume that the data points are linearly separable, there is no misclassification, and all data points that belong to the same class lie on the same side of the hyperplane, identified by the sign of the linear equation. This is visualized in Fig. 9.9.

In Fig. 9.9, we included two (of many) possible hyperplanes (in blue and red), separating the data points and providing a perfect classification. However, the question is which of the possible separating hyperplanes should be chosen? We

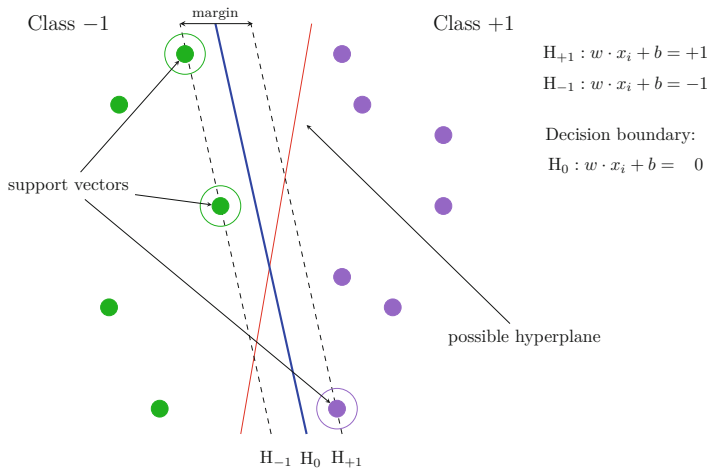


Fig. 9.9 Maximum margin (hard-margin) SVM for linear separable classes. The optimal hyperplane, H_0 , maximizes the distance between the auxiliary hyperplanes H_{+1} and H_{-1}

would like to remind that the position of a hyperplane can be modified by changing w and b , resulting in a change in the angle and the position of the hyperplane (see [149]).

The idea of the hyperplane we consider as “optimal” is depicted in blue in Fig. 9.9. This hyperplane, H_0 , is located “in the middle,” between the two classes. In order to understand this better, we included two auxiliary hyperplanes, H_{+1} and H_{-1} (in dashed lines), which are parallel to the optimal hyperplane and are characterized by

$$H_{+1} : w \cdot x_i + b = +1, \quad (9.48)$$

$$H_{-1} : w \cdot x_i + b = -1. \quad (9.49)$$

Here, the minimal distance from the optimal hyperplane to either of these auxiliary hyperplanes is the same.

In order to find the parameters of the linear equations formally, one rescales Eqs. 9.46 and 9.47 in the following way. First, we determine

$$\min \{ w \cdot x_i + b \} = +m_{+1} \text{ for } y_i = +1, \quad (9.50)$$

$$\max \{ w \cdot x_i + b \} = -m_{-1} \text{ for } y_i = -1, \quad (9.51)$$

for the training data. Here m_{+1} and m_{-1} are positive numbers, i.e., $m_{+1}, m_{-1} \in \mathbb{R}^+$, which correspond to the minimal distances from the data points in class +1 and -1 to the hyperplane. Since the data points in class -1 assume negative values, we

wrote $-m_{-1}$ to make m_{-1} positive. The optimal hyperplane shall have an equal distance to both classes, leading to $m = m_{+1} = m_{-1}$. If we substitute w and b with their rescaled values $w' = w/m$ and $b' = b/m$, we obtain

$$w' \cdot x_i + b' \geq +1, \quad \text{for all } i \text{ with } y_i = +1, \quad (9.52)$$

$$w' \cdot x_i + b' \leq -1, \quad \text{for all } i \text{ with } y_i = -1. \quad (9.53)$$

Let us reflect for a moment what the distances of these hyperplanes are from the origin. For the hyperplane in Eq. 9.52, we have a perpendicular distance to the origin, denoted $d_{+1} = |1 - b|/||w||$, and for Eq. 9.53, we have $d_{-1} = |-1 - b|/||w||$. Hence, the margin between the two auxiliary hyperplanes is $d_{+1} - d_{-1} = 2/||w||$. From this, we can see that if we want to maximize the distance between the two auxiliary hyperplanes, we need to minimize $||w||$ by fulfilling the constraints in Eqs. 9.52 and 9.53. By re-writing these equations in the form

$$y_i(w' \cdot x_i + b') - 1 \geq 0 \text{ for all } i \quad (9.54)$$

we are now in a position to express this problem as an optimization problem. Specifically, we can define a Lagrangian by

$$L_P = \frac{1}{2}w^2 - \sum_i \alpha_i \left(y_i(w' \cdot x_i + b') - 1 \right). \quad (9.55)$$

Minimization of L_P with respect to w and b for positive Lagrange multipliers α_i gives the desired solution. We just want to note that when solving this optimization problem, the data points x_i with positive Lagrange multipliers, i.e., $\alpha_i > 0$, are called “support vectors.” These lie exactly on the two auxiliary hyperplanes (see Fig. 9.9). All other Lagrange multipliers are zero, and the corresponding data points are not relevant for learning the SVM.

9.8.2 Nonlinearly Separable Data

The above formulation does not allow data that are not linear separable. That means, the above formalism can only be applied to linear separable data, which limits the application severely. In order to extend the above classifier to general data that are not linear separable, some so-called slack variables need to be introduced to the problem.

Using positive *slack variables* $\xi_i \in \mathbb{R}^+$, errors in the classification can be counterbalanced so that the following equations hold:

$$w \cdot x_i + b \geq +1 - \xi_i, \quad \text{for all } i \text{ with } y_i = +1 \quad (9.56)$$

$$w \cdot x_i + b \leq -1 + \xi_i, \quad \text{for all } i \text{ with } y_i = -1 \quad (9.57)$$

Specifically, if there is a classification error for data point x_i , the corresponding slack variable is $\xi_i > 0$, and its value corresponds to the distance from an auxiliary hyperplane given by its class label, i.e., y_i . For instance, for class +1, a slack variable is between zero and one if x_i is located between H_{+1} and H_0 and larger than one if located beyond H_0 (see Fig. 9.10 for a visualization). On the other hand, if the classification is correct, then $\xi_i = 0$. That means $\sum_i \xi_i$ is an indicator for the classification error.

The corresponding Lagrangian (for the hinge loss [339]) can be written as

$$L_P = \frac{1}{2}w^2 - \sum_i \alpha_i \left(y_i (w \cdot x_i + b) - 1 + \xi_i \right) + C \sum_i \xi_i - \sum_i \mu_i \xi_i. \quad (9.58)$$

Here, the μ_i are additional Lagrange multipliers included to enforce positivity of the slack variables, and C is a constant representing the cost of constraint violation.

9.8.3 Nonlinear Support Vector Machines

Finally, we are in a position to extend the classifier also to nonlinear decision boundaries. In order to do this, we first reformulate the Lagrangian in Eq. 9.58. Specifically, the dual Lagrangian of Eq. 9.58 is given by

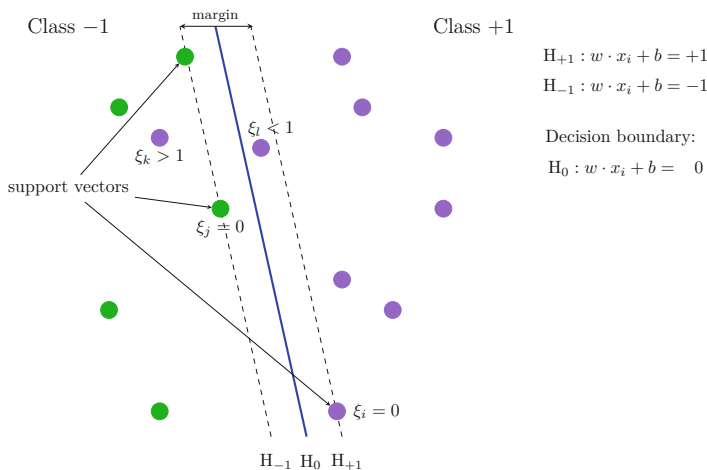


Fig. 9.10 Softmargin SVM for nonseparable classes. Falsely classified data points have a value of the slack variable larger than 1 (see ξ_k), and correctly classified data points within the margin have a negative value (see ξ_l)

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j. \quad (9.59)$$

The crucial point here is that L_D contains the data only in the form of a scalar product $x_i \cdot x_j$. This allows to “plug in” a transformation that behaves like a scalar product, namely, a kernel. This is in general called the *kernel trick*.

In order to understand this, we are mapping from the space of our data to another (possibly infinite dimensional) Euclidian space $\mathcal{H}$ as follows:

$$\Phi : \mathbb{R}^p \rightarrow \mathcal{H}, \quad (9.60)$$

and define a kernel by

$$K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j). \quad (9.61)$$

That means, a kernel is just a function that depends on two arguments and behaves like a scalar product in the Euclidian space $\mathcal{H}$. Interestingly, if one finds such a kernel, one does no longer need the auxiliary mapping Φ explicitly but can use it implicitly. This becomes clear when looking at the kernels given below in Eqs. 9.62–9.64 because the mapping Φ is not immediately identifiable.

It can be shown that the following functions are kernels:

$$K(x_i, x_j) = \exp\left(-\frac{x_i - x_j^2}{2\sigma^2}\right) \quad (\text{Radial base function}) \quad (9.62)$$

$$K(x_i, x_j) = (x_i \cdot x_j + 1)^q \quad (\text{Polynomial of degree } q) \quad (9.63)$$

$$K(x_i, x_j) = \tanh(\kappa x_i \cdot x_j - \delta) \quad (\text{Sigmoid function}) \quad (9.64)$$

Further examples of kernel functions are the Mahalanobis kernel and graph kernels. We would like to mention that for the linear kernel

$$K(x_i, x_j) = x_i \cdot x_j \quad (9.65)$$

one obtains a (linear) SVM.

We would like to mention that Mercer’s condition [339] can be used to check whether a function is a valid kernel.

9.8.4 Examples

The first example we discuss is for a binary classification. In Listing 9.12, we generate training data with $N_1 = 30$, $N_2 = 50$ samples. Each of the data points is

sampled from a two-dimensional normal distribution with mean μ_i and covariance matrix σ_i .

Listing 9.12: Training data for a SVM classifier, see Fig. 9.11

```
N1 <- 30; N2 <- 50
m1 <- c(1,1); sigma1 <- diag(length(m1))
m2 <- c(4,3); sigma2 <- matrix(c(1,0.3,0.3,1), nrow=2, ncol=2,
  byrow=F)
x1 <- rmvnorm(N1, mean = m1, sigma = sigma1)
x2 <- rmvnorm(N2, mean = m2, sigma = sigma2)
x1.train <- x1; x2.train <- x2;
cl.train <- c(rep(1, N1), rep(2, N2))
df.train <- data.frame("x"=rbind(x1, x2), y=cl.train)
```

To perform an analysis with a SVM, we use the package *e1071* available in R. Listing 9.13 shows this analysis for a radial base kernel, and the resulting decision boundaries are visualized in Fig. 9.11 (top). As one can see, the decision boundaries are nonlinear between the two class regions. In order to highlight the support vectors from the other data points, they are indicated using the symbol “s.”

To highlight the difference compared to the other kernels, we repeated a similar analysis for the sigmoidal kernel. The results of this analysis are shown in Fig. 9.11 (bottom). As one can see, this kernel also performs a nonlinear separation; however, the decision boundary has a different shape. This indicates a flexibility induced by the kernels, which is a significant improvement over linear decision boundaries. Other possible kernels provided in the package *e1071* are:

- linear
- polynomial
- radial basis
- sigmoid.

The option “type” is not of importance here, because we want to focus on the base version of a SVM for classification problems only. However, we would like to note that a SVM can also be used for regression problems. Furthermore, there exist different algorithmic variations of SVM from which one can select.

In Listing 9.13, we show also an example for making predictions. For this, we simulated test data similar to Listing 9.12; however, for $N_1 = 300$, $N_2 = 300$ samples.

The next example we study is a three-class classification. For this, we generate data similar to Listing 9.10. The resulting decision boundaries are visualized in Fig. 9.12, and Listing 9.14 shows the results of the prediction.

We would like to note that usually, kernels are parameter dependent. For instance, the radial base function depends on σ (see Eq. 9.62) and the polynomial function on q (see Eq. 9.64). Furthermore, the Lagrangian of a SVM allows to specify the cost of constraints violation, C . For our above analysis, we used default values given in the R package, however, these are not the optimal values for these parameters. Instead,

Listing 9.13: Analysis of the SVM classifier, see Fig. 9.11

```

library("e1071")
library("caret")
model.radial <- svm(y ~., df.train, kernel="radial",
  type="C-classification")
plot(model.radial, df.train.n, col = c("purple", "green3"),
  svSymbol = "s", grid=50)

pred <- predict(model.radial, df.test)
conf <- table(list(observed=cl.test, predicted=pred))
confusionMatrix(conf)
# output
Confusion Matrix and Statistics

      predicted
observed  1    2
      1 278   22
      2   15 285

      Accuracy : 0.9383
      95% CI   : (0.916, 0.9562)
No Information Rate : 0.5117
P-Value [Acc > NIR] : <2e-16

      Kappa   : 0.8767

McNemar's Test P-Value : 0.3239

      Sensitivity : 0.9488
      Specificity : 0.9283
      Pos Pred Value : 0.9267
      Neg Pred Value : 0.9500
      Prevalence : 0.4883
      Detection Rate : 0.4633
      Detection Prevalence : 0.5000
      Balanced Accuracy : 0.9386

      'Positive' Class : 1

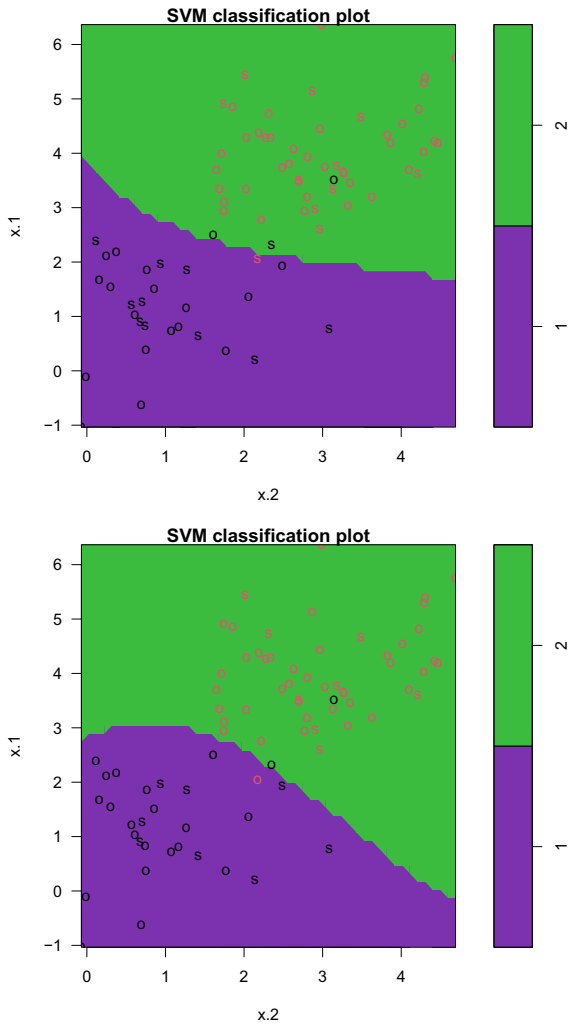
```

these parameters need to be estimated via model selection, which will be discussed in Chap. 12.

9.9 Decision Tree

The systematic investigation of decision trees goes back to the 1960s and 1970s, and automatic interaction detection (AID) and theta automatic interaction detection (THAID) are widely considered as the first algorithms for regression trees and classification trees, respectively [314]. However, a breakthrough came with the development of Classification and Regression Tree (CART) [57] in the 1980s

Fig. 9.11 Support vector machine classification for a radial based kernel (top) and a sigmoidal kernel (bottom). Shown are the decision boundaries between the two classes. Support vectors are highlighted using the symbol “s.”

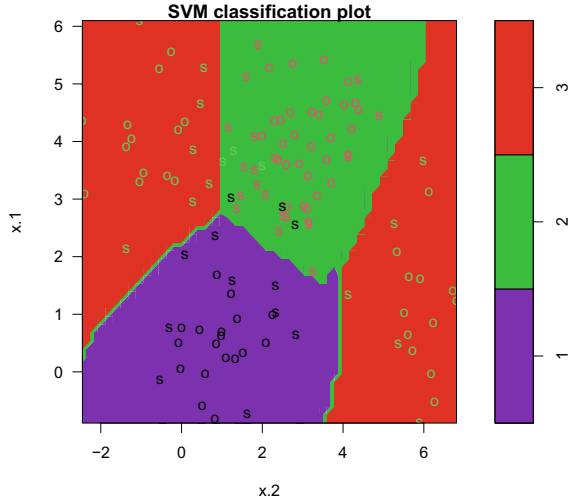


because this algorithm addressed successfully the underfitting and overfitting problems of AID and THAID through a pruning process (discussed below). This section will be dedicated to the CART method.

9.9.1 What Is a Decision Tree

Let’s start with an example of a decision tree and its components. For our example, we will use the package *rpart* available in R. This package provides an implementation of CART (classification and regression tree) as described in [57].

Fig. 9.12 Three-class classification with a support vector machine for a radial-based kernel. Shown are the decision boundaries between the three classes. Support vectors are highlighted using the symbol “s.”



The reason behind the name “rpart” (recursive partitioning) for this package instead of CART is that the latter is a protected trademark name.

In Fig. 9.13 (top-left), we show a set of training data for three classes. These data are sampled from 3 normal distributions, as in Listing 9.10. A decision tree learned from these training data is shown in Fig. 9.13 (top-right). Let’s for the moment just accept that we have a decision tree without asking how we obtained it. This question will be answered below.

In general, a decision tree consists of two types of nodes:

1. Decision nodes, and
2. Leaf nodes.

In Fig. 9.13 (top right), the decision nodes are shown in gray, whereas the leaf nodes are shown in color. Furthermore, for every decision node, there is an inequality. For example, decision node 1 contains the inequality

$$x.1 \leq 1.9. \quad (9.66)$$

In general, these inequalities are used to make decisions regarding the partitioning of the data. Because such a partitioning of the data is applied until the data samples reach a leaf node, this is a recursive partitioning of the data, explaining the name of the package *rpart*. For example, for decision node 1, the data are split into two parts depending on whether the inequality is true or false for the $x.1$ component of a data point. The data points for which the inequality holds reach decision node 2, while the remaining data points reach decision node 3. Hence, a decision tree performs, at each decision node, a bipartitioning of the data, because either an inequality is true or false. Overall, this makes the tree a binary decision tree.

Listing 9.14: Analysis of a 3-class SVM classifier, see Fig. 9.12

```

model.radial <- svm(y ~., df.train, kernel="radial",
  type="C-classification")
pred <- predict(model.radial, df.test)
conf <- table(list(observed=cl.test, predicted=pred))
confusionMatrix(conf)
#output
Confusion Matrix and Statistics

      predicted
observed  1    2    3
      1 259  17  24
      2   6 280  14
      3   9  16 275

Overall Statistics

                Accuracy : 0.9044
                  95% CI : (0.8833, 0.9229)
    No Information Rate : 0.3478
    P-Value [Acc > NIR] : < 2e-16

                Kappa : 0.8567

    McNemar's Test P-Value : 0.00669

Statistics by Class:

               Class: 1 Class: 2 Class: 3
Sensitivity    0.9453    0.8946    0.8786
Specificity    0.9345    0.9659    0.9574
Pos Pred Value 0.8633    0.9333    0.9167
Neg Pred Value 0.9750    0.9450    0.9367
Prevalence     0.3044    0.3478    0.3478
Detection Rate 0.2878    0.3111    0.3056
Detection Prevalence 0.3333    0.3333    0.3333
Balanced Accuracy 0.9399    0.9302    0.9180

```

As a consequence of the form of decisions (see, e.g., Eq. 9.66), one obtained linear decision boundaries for the tree. For our example, this is visualized in Fig. 9.13 (bottom left). That means more decision nodes will not lead to nonlinear decisions but more fine-grained boundaries.

The information shown in the leaf nodes corresponds to the predicted class label (number in the middle of the first row in each leaf node) and the fraction of training samples classified as class 1, 2, or 3. For instance, leaf node 5 makes the prediction for each data sample being in class 3. However, for the training data, about 5% of these are actually from class 1 and from class 2, and about 90% are from class 3. The total numbers that correspond to these percentages are shown in Fig. 9.13 (bottom right). This figure includes not only the training samples that reach the leaf nodes but in addition shows also these numbers for the decision nodes. In this way, it may become more clear how the data are actually processed during the decision-making.

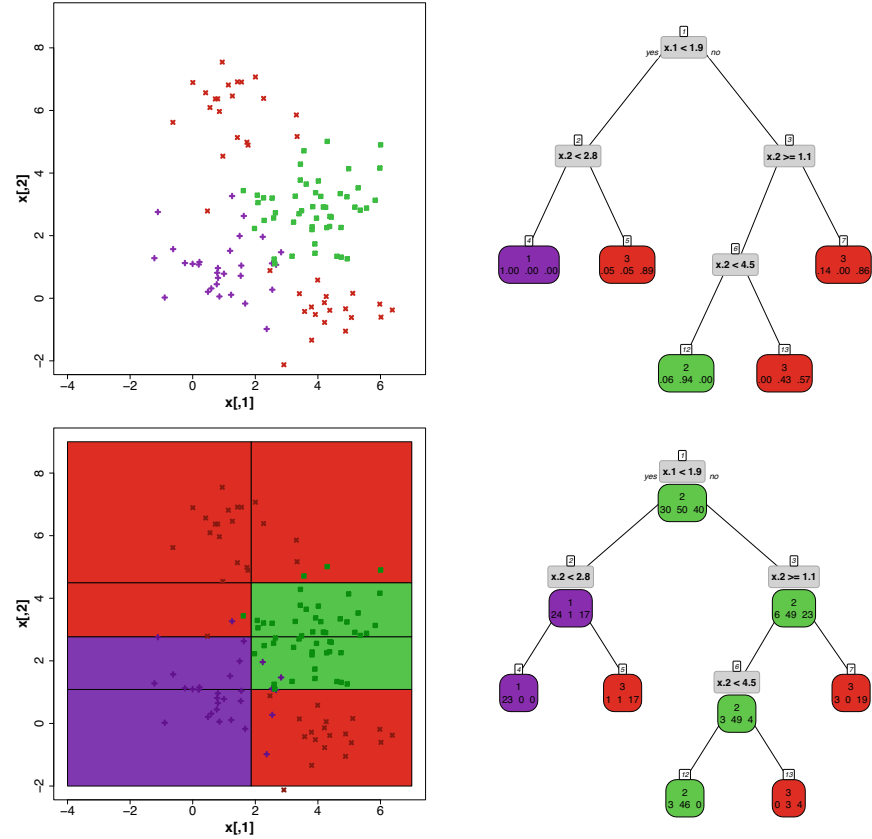


Fig. 9.13 A decision tree for three classes. Top left: Distribution of the training data. Top right: Decision tree learned from the training data. Bottom right: Decision boundaries of the decision tree. Bottom left: Same decision tree as top right but showing the number of training samples in each class at each node

9.9.1.1 Three Principal Steps to Get a Decision Tree

As we have seen above, a decision tree has a structure, which can be intuitively understood via a simple graphical representation. Now that we understand a decision tree, we turn to the question as to how we actually learn, i.e., construct, a decision tree from training data. We answer this question by subdividing it into three sub-problems:

1. Growing a decision tree.
2. Assessing the size a decision tree.
3. Pruning a decision tree.

The general idea behind this is to, first, define a splitting criterion, called impurity function, which will be used at the decision nodes to separate the data. This process grows a tree, but does not provide a stopping criterion, which prevents further subdivisions of the data. For this reason, this process will result in a fully grown tree that contains in each leaf node either just one training sample or only samples from the same class. Second, for finding the optimal size of the tree, we need to define a criterion that provides us with information about the predictive performance of the tree. Third, based on this, the tree will be pruned by getting rid of branches that would most likely lead to overfitting of the data. This describes the logical way to obtain a decision tree. In the following, we describe each of these three steps in detail.

9.9.2 Step 1: Growing a Decision Tree

To grow a tree, we need to define (1) a decision criterion and (2) a decision function that selects a decision criterion. For simplicity, we are focusing here on continuous variables, and we chose as decision criterion an inequality for one variable of the form

$$x_i \leq \gamma. \quad (9.67)$$

A common choice is to select the variable x_i randomly from all available variables, which leaves us to specify the numerical value of the parameter γ . To do this, we need to define a decision function, called *impurity function*. The purpose of the impurity function is to assess a split.

As the name “impurity” suggests, it is bad to split a set of data points in a way that results in a high impurity. Here, the impurity is assessed with respect to the discrete distribution of the samples, i.e., for C different classes, there will be a certain number of training samples f_j in class $j \in \{1, \dots, C\}$ with $f_j =$ number of training samples in class j . From this, one obtains the distribution

$$p_j = \frac{f_j}{\sum_k f_k} \quad (9.68)$$

that is used to assess the impurity of a node. There are two impurity functions widely used, namely, the entropy and the Gini index defined as follows:

$$i.e(n) = - \sum_j p_j \log(p_j) \quad (\text{Entropy}); \quad (9.69)$$

$$i.g(n) = \sum_{i \neq j} p_i p_j = \frac{1}{2} \left(1 - \sum_j p_j^2 \right) \quad (\text{Gini index}). \quad (9.70)$$

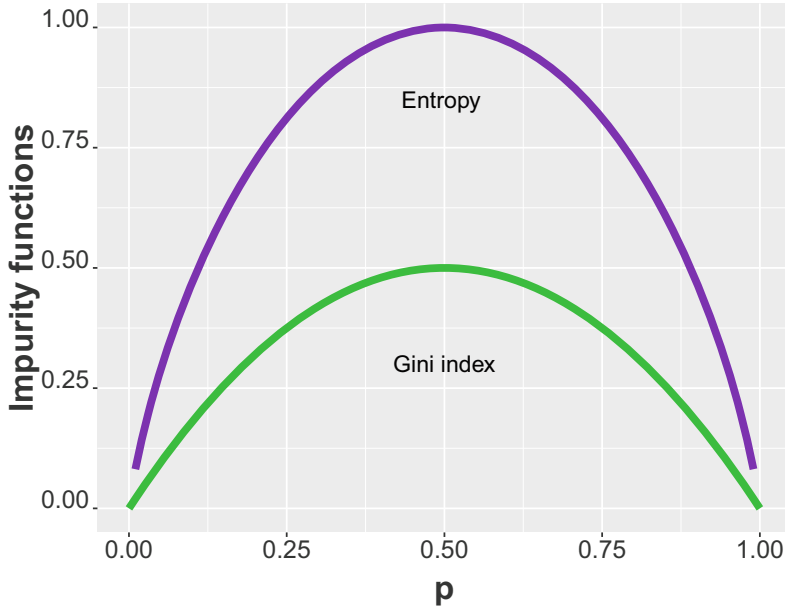


Fig. 9.14 Two examples for impurity functions: Gini index (lower curve) and entropy (upper curve) for a binary classification

The above impurity functions depend on n , which is the index of the node being evaluated within a decision tree.

In Fig. 9.14, we show the entropy and the Gini index for a binary classification problem, i.e., the number of classes $C = 2$. As one can see, in both cases, the impurity is high for intermediate values of p and lowest for $p = 0$ and $p = 1$. That means, both functions are useful for identifying desirable splits.

In order to identify the optimal value of γ at a given node m , one needs a measure to evaluate outcomes of splits, corresponding to a goodness of split, $\Delta i(m, \gamma)$. Formally, this measure evaluates, for a selected impurity function i , the impurity reduction at node m and at its two offsprings (children) m_L and m_R in the following way:

$$\Delta i(m; \gamma) = i(m; \gamma) - \text{Prob}_L \cdot i(m_L; \gamma) - \text{Prob}_R \cdot i(m_R; \gamma). \quad (9.71)$$

For a given γ , Prob_L is the probability of training samples to reach the left (child) node and $\text{Prob}_R = 1 - \text{Prob}_L$ the probability to reach the right (child) node. Evaluating $\Delta i(m; \gamma)$ for different values of γ , the optimal γ will maximize $\Delta i(m; \gamma)$, i.e.,

$$\gamma^* = \underset{\gamma}{\operatorname{argmax}} \Delta i(m; \gamma) \quad (9.72)$$

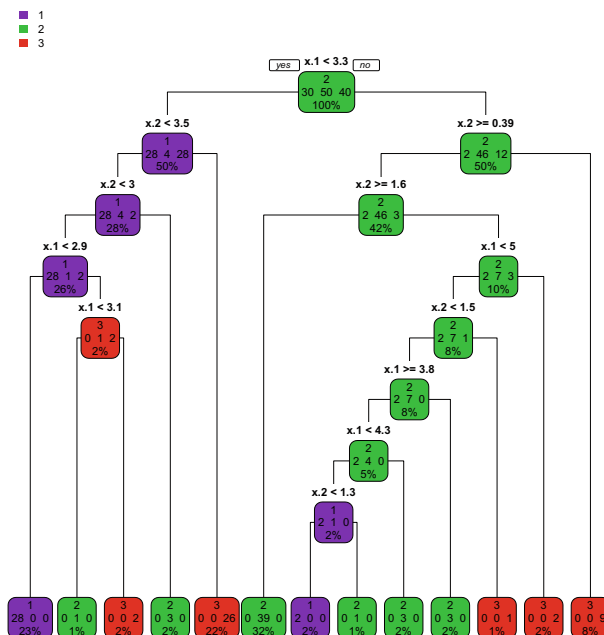


Fig. 9.15 Fully grown decision tree for a Gini impurity function

resulting in a maximal impurity reduction. Hence, $\Delta i(m; \gamma)$ describes the *goodness of split* based on an impurity function.

Successive application of the above procedure, to each decision node in a tree, will result in a full tree that stops growing when one of the following two conditions hold:

1. All samples belong to the same class.
2. There is only one sample left. These nodes correspond then to the leaf nodes.

In both cases, there are only sample(s) of one class left, and, hence, the impurity of the node reached already zero.

In Fig. 9.15, we show an example for such a tree. For this example, we used a Gini impurity, and Listing 9.15 gives the corresponding implementation using R.

The function used to generate a decision tree using R is `rpart()`. The option “method,” for this function, allows to select between a regression tree and a decision tree, among others. Since, here, we are only focusing on classification problems, this is specified by “class.” The next option, “parms,” allows to select an impurity function. Available options are “information,” corresponding to the entropy impurity, and “gini” giving the Gini impurity.

The next option, “control,” is very important, because it allows to set stopping criteria via the auxiliary function “`rpart.control`.” Specifically, “`minsplit`” sets the minimum number of observations that must exist in a node in order for a split to

Listing 9.15: Growing a full decision tree, see Fig. 9.15

```

library("rpart")
library("rpart.plot")
# training data
N1 <- 30; N2 <- 50; N3 <- 40
m1 <- c(1,1); sigma1 <- diag(length(m1))
m2 <- c(4,3); sigma2 <- matrix(c(1,0.3,0.3,1), nrow=2, ncol=2,
  byrow=F)
m3.1 <- c(1,6); m3.2 <- c(4,0); sigma3 <- diag(length(m3.1))
x1 <- rmvnorm(N1, mean = m1, sigma = sigma1)
x2 <- rmvnorm(N2, mean = m2, sigma = sigma2)
x3 <- matrix(0, nrow=N3, ncol=2)
for(i in 1:N3){
  coin <- round(runif(1))
  if(coin<0.5){ x3[i,] <- rmvnorm(1, mean = m3.1, sigma = sigma3)}
  else{ x3[i,] <- rmvnorm(1, mean = m3.2, sigma = sigma3)}
}
x1.train <- x1; x2.train <- x2; x3.train <- x3
cl.train <- as.factor(c(rep(1, N1), rep(2, N2), rep(3, N3)))
df.train <- data.frame(rbind(x1, x2, x3), cl.train)
colnames(df.train) <- c("x.1", "x.2", "y")

model <- rpart(y~x.1 + x.2, data = df.train, method="class",
  parms=list(split='gini'), control=rpart.control(minsplit = 2,
  cp = 0.0))
rpart.plot(model, fallen.leaves = TRUE, type = 1, extra = 101,
  box.palette = list("purple", "green3", "red"))

```

be attempted. That means, if the number of samples at a node is smaller than k (for “minsplit= k ”), then this will be a leaf node. The option “cp” is a (positive) complexity parameter. Any split that does not decrease the overall lack of fit by a factor of “cp” is not attempted. By setting “cp=0.0,” we are effectively not making use of this option; hence, we accept any attempted split, no matter how much improvement it will bring. The reason for this is that we want to grow a tree to its maximal size, which we call a full tree.

9.9.3 Step 2: Assessing the Size of a Decision Tree

After having grown a full tree, we need to assess it and then cut it to the right size. A common way to determine the optimal size of a decision tree is to assess the classification error of a tree according to its complexity. This will allow us to identify the optimal model with a minimal complexity in order to avoid overfitting. As usual, there is more than one way to accomplish this goal. In the following, we first outline an intuitive way to do this, and then, we show a more elegant, formal approach, hence more involved.

9.9.3.1 Intuitive Approach

Having a fully grown tree T_f , we can successively reduce its size by cutting off branches one node after another. If we do this for all possible cuts, this results in a set of trees $\{T_f, \dots, T_0\}$, where T_f is the full tree and T_0 is the root tree consisting just of one node. Using the training data, we can evaluate the prediction error of all trees $\{T_f, \dots, T_0\}$, e.g., via cross-validation. Finally, we select the smallest tree achieving an acceptable error. Here, we define the size of a tree as its number of nodes.

In principle, this is a valid approach; however, the meaning of “acceptable error” is quite vague because we did not define with sufficient detail what numerical decision could be used to call the error of a tree sufficiently low. For this reason, we are presenting now a precise formulation of this problem.

9.9.3.2 Formal Approach

In order to approach this problem formally, we define first the *tree cost-complexity* as follows:

$$R_\alpha(T) = R(T) + \alpha|\tilde{T}|. \quad (9.73)$$

Here α is a positive cost-complexity parameter and $\tilde{T}$ is the set of terminal nodes in T which means $|\tilde{T}|$ measures the tree complexity with respect to the number of terminal nodes in tree T . For this reason, $|\tilde{T}|$ is given by

$$|\tilde{T}| = \text{number of terminal nodes in tree } T. \quad (9.74)$$

There is a relationship between the complexity of a tree and its size $|T|$ given by

$$|T| = 2|\tilde{T}| - 1. \quad (9.75)$$

The term $R(T)$, called *resubstitution error*, is defined by

$$R(T) = \sum_{\tau \in \tilde{T}} R(\tau) = \sum_{\tau \in \tilde{T}} p(\tau)r(\tau) \quad (9.76)$$

where τ is a terminal node in tree T . The underlying idea of $R(T)$ is to assume that the training data provide a good representation of the population, allowing a representative estimation of the error. Hence, $R(T)$ estimates the misclassification error of a tree T , using the training data.

The resubstitution error of a node, $R(\tau)$, is given by a product of $p(\tau)$ and $r(\tau)$. The probability $p(\tau)$ is just the fraction of training samples that are at node τ divided by the total number of training samples

$$p(\tau) = \frac{\text{training samples at node } \tau}{\text{total number of samples}}, \quad (9.77)$$

and $r(\tau)$ is the within-node (conditional) misclassification error. Formally, $r(\tau)$ is defined by

$$r(\tau) = \sum_i c(j|i) p(Y = i|\tau). \quad (9.78)$$

The term $c(j|i)$ is the (conditional) misclassification cost for classifying a sample from class i as j . Practically, $r(\tau)$ can be estimated by

$$r(\tau) = 1 - \max_i p(Y = i|\tau), \quad (9.79)$$

and the conditional probability $p(Y = i|\tau)$ can be estimated from the training data as follows:

$$p(Y = i|\tau) = \frac{\text{training samples at node } \tau \text{ in class } i}{\text{training samples at node } \tau}. \quad (9.80)$$

The key idea is to use $R_\alpha(T)$, which is a scalar, positive, real valued number, i.e., $R_\alpha(T) \in \mathbb{R}^+$, as a representation of the complexity of tree T . Breiman [57] showed that the following holds:

- If $\alpha > \beta$ then either $T_\alpha = T_\beta$ or T_α is a strict sub-tree of T_β

In other words, if one increases α successively, then the resulting subtrees are getting smaller and smaller and are strict sub-trees of each other.

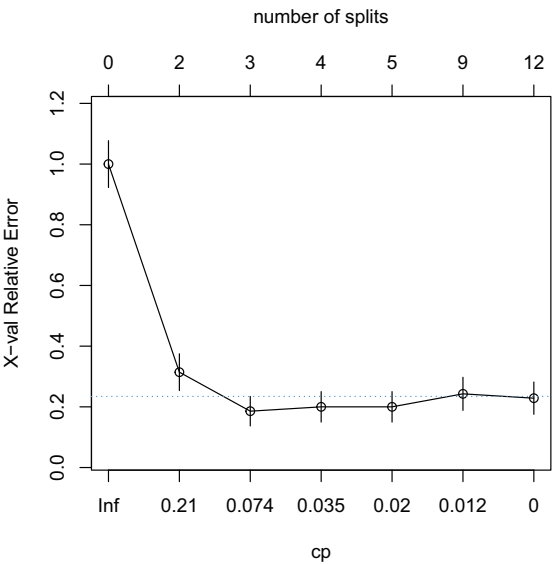
Now, we are in a position to present a precise realization of the intuitive approach, discussed in the previous section using $R_\alpha(T)$. Specifically, by varying α from 0 to $+\infty$, we get a set of nested trees where $T_{\alpha=0}$ corresponds to the full tree and $T_{\alpha \rightarrow +\infty}$ to the root node. Then for each of these trees T_α , we estimate its classification error using a cross-validation approach. In Fig. 9.16, we show this error as a function of the complexity of a tree. In R, α is denoted by cp . The only point that is left is to select where to cut the decision tree.

To find the optimal complexity, cp^* , there are a number of different approaches. However, one of the simplest and most frequently applied criterion is the '*one-standard error rule*' (1-SE rule) [222]. The 1-SE rule selects the smallest tree (least complex tree) that is within one standard error (SE) of the tree, T' , with the best cross-validation error, i.e., $E_{min} = E(T')$. That means from the candidate set

$$S = \{cp | E(T(cp)) \leq E_{min} + SE(E_{min})\}, \quad (9.81)$$

the optimal complexity is given by

Fig. 9.16 Cross-validation classification error as a function of the complexity of decision trees. The size of the trees indicates the number of terminal nodes, and the dashed horizontal line corresponds to $E_{min} + SE(E_{min})$



$$cp^* = \underset{cp}{\operatorname{argmin}} \left\{ cp \in S \right\}. \tag{9.82}$$

Figure 9.16 shows the classification error as a function of cp . From this, one can see that the optimal complexity is obtained for 3 splits. Unfortunately, neither the shown classification error nor the complexity values are absolute values but scaled, and the values are not consistent among different functions available in R. Specifically, the “ cp ” values obtained with “ $\text{model\$cptable}$ ” (see Listing 9.16) are different from the values shown in Fig. 9.16.

Listing 9.16: Find the optimal complexity value, see Fig. 9.16

```
plotcp(model, upper = "splits", col = 4)

model$cptable
      CP nsplit  rel error    xerror    xstd
1 0.35714286    0 1.00000000 1.0000000 0.07715167
2 0.12857143    2 0.28571429 0.3142857 0.06055301
3 0.04285714    3 0.15714286 0.1857143 0.04863791
4 0.02857143    4 0.11428571 0.2000000 0.05023753
5 0.01428571    5 0.08571429 0.2000000 0.05023753
6 0.00952381    9 0.02857143 0.2428571 0.05457006
7 0.00000000   12 0.00000000 0.2285714 0.05319710
```

The transformation in Listing 9.17 gives cp_{fig} from the “plotcp” function for cp_{tab} from “cptable.”

Listing 9.17: Transformation of cp values, see Fig. 9.16

```

cp.tab <- model$cptable
L <- length(cp.tab)
cp.fig <- vector(mode="numeric", length=L)
for(i in 2:L){
  cp.fig[i] <- sqrt(cp.tab[i] * cp.tab[i-1])
}
cp.fig[1] <- Inf

```

This difference is important because using $cp_{fig} = 0.074$ from Fig. 9.16 gives a different tree than using $cp_{tab} = 0.042$ from “model\$cptable.” The correct values are given in the table; hence, for our example, the optimal complexity value is $cp^* = 0.042$.

9.9.4 Step 3: Pruning a Decision Tree

The final step left is to actually prune the tree at the selected complexity level. In R, this can be done by using the function `prune()`.

Listing 9.18: Pruning a decision tree at the optimal complexity level, see Fig. 9.15

```

model_pruned <- prune(model, cp =
  model$cptable[which.min(model$cptable[, "xerror"] +
    model$cptable[, "xstd"]), "CP"] )
rpart.plot(model_pruned, fallen.leaves = FALSE, type = 1, extra =
  101, box.palette = list("purple", "green3", "red"))

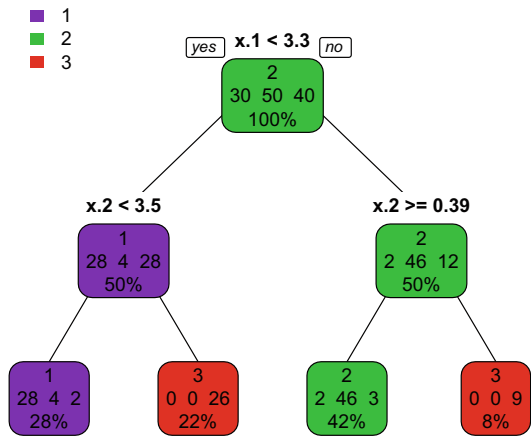
```

Figure 9.17 shows the resulting decision tree. One can see that this tree uses three splits as indicated in the “cptable” shown in Listing 9.16. We would like to note that one should always cross-check if the resulting cut corresponds to the desired tree complexity.

9.9.4.1 Alternative Way to Construct Optimal Decision Trees: Stopping Rules

There is a second way to obtain an optimal decision tree, which is fundamentally different from the approach described above. Instead of growing a full tree and pruning it as suggested by Breiman, one can use a stopping rule to prevent a tree from further growing. This is called *early stopping* or pre-pruning. This idea is very appealing at first because it appears simpler and, in fact, is computationally less demanding. However, the problem with such an approach is that one needs to

Fig. 9.17 Pruned decision tree of the full tree in Fig. 9.15



use a stopping rule that is capable of *looking ahead*. What we mean by that is the following: Suppose that you grow a decision tree, and the stopping criterion you selected suggests not to further grow a specific branch. Unfortunately, it is possible that when actually making this split, further down this branch, the resulting leaf nodes may be in fact better terminal nodes than the ones further up in the tree. This problem is common to *greedy optimization* methods, and a stopping rule is just one of them.

Practical criteria used as stopping rules are, e.g., requiring a minimal number of samples in a node in order to attempt a split. This rule can be used in combination or instead of requiring a minimal value of impurity reduction (see Eq. 9.71).

9.9.5 Predictions

Finally, after obtaining the optimal decision tree, we can now use it for making predictions. In Listing 9.19, we show an example using the test data.

9.10 Summary

In this chapter, we discussed classification problems. In general, a classification requires a form of supervised learning, where each data point is associated with a label. Importantly, the label provides only categorical information and does not correspond to a numerical value. The case of numerical response variables will be the topic of Chap. 11, where we discuss regression models.

We discussed a naive Bayes classifier, linear discriminant analysis, k-nearest neighbor classification, logistic regression, support vector machine, and decision

Listing 9.19: Prediction with a pruned decision tree

```
# test data
N1 <- 300; N2 <- 300; N3 <- 300
m1 <- c(1,1); sigma1 <- diag(length(m1))
m2 <- c(4,3); sigma2 <- matrix(c(1,0.3,0.3,1), nrow=2, ncol=2,
  byrow=F)
m3.1 <- c(1,6); m3.2 <- c(4,0); sigma3 <- diag(length(m3.1))
x1 <- rmvnorm(N1, mean = m1, sigma = sigma1)
x2 <- rmvnorm(N2, mean = m2, sigma = sigma2)
x3 <- matrix(0, nrow=N3, ncol=2)
for(i in 1:N3){
  coin <- round(runif(1))
  if(coin<0.5){ x3[i,] <- rmvnorm(1, mean = m3.1, sigma = sigma3)}
  else{ x3[i,] <- rmvnorm(1, mean = m3.2, sigma = sigma3)}
}
x1.test <- x1; x2.test <- x2; x3.test <- x3
cl.test <- c(rep(1, N1), rep(2, N2), rep(3, N3))
df.test <- data.frame("x"=rbind(x1, x2, x3), y=cl.test)

pred <- predict(model_pruned, df.test, type = 'class')
conf <- table(list(observed=df.test$y, predicted=pred))
confusionMatrix(conf)
#output
Confusion Matrix and Statistics
```

	predicted		
observed	1	2	3
1	282	7	11
2	12	285	3
3	12	51	237

```
Overall Statistics

          Accuracy : 0.8933
          95% CI   : (0.8713, 0.9127)
    No Information Rate : 0.3811
    P-Value [Acc > NIR] : < 2.2e-16

          Kappa   : 0.84

    McNemar's Test P-Value : 1.49e-09

Statistics by Class:
```

	Class: 1	Class: 2	Class: 3
Sensitivity	0.9216	0.8309	0.9442
Specificity	0.9697	0.9731	0.9029
Pos Pred Value	0.9400	0.9500	0.7900
Neg Pred Value	0.9600	0.9033	0.9767
Prevalence	0.3400	0.3811	0.2789
Detection Rate	0.3133	0.3167	0.2633
Detection Prevalence	0.3333	0.3333	0.3333
Balanced Accuracy	0.9456	0.9020	0.9236

tree. Interestingly, despite the fact that all of these methods provide approaches for classifying data, their underlying working mechanisms are quite different from each other. Specifically, while a naive Bayes classifier, linear discriminant analysis, logistic regression, and k-nearest neighbor classification learn conditional probability distributions, a support vector machine solves an optimization problem that optimizes the regularized separation of data points by hyperplanes. Yet a different approach is used by a decision tree, which is a non-parametric procedure that decomposes the classification problem into separate (linear) decision rules.

When we discussed general prediction models in Chap. 2, we saw that in data science, there is no unifying principle, which would allow to categorize all models uniquely. In this chapter, we have seen an example of this plurality for classification methods.

Learning Outcome 9: Classification

Classification methods are supervised learning approaches that require labeled data for the training. The labels provide only categorical information and do not correspond to numerical values.

9.11 Exercises

1. The classification error of a binary classification is defined by

$$\text{classification error} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \quad (9.83)$$

How is this error related to the accuracy given by

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}? \quad (9.84)$$

2. Study a naive Bayes classifier for a two-class classification problem using simulated data.
 - a. Estimate the four fundamental errors, TP, FP, TN, and FN, for the values given in Listing 9.4.
 - b. Reproduce the results with Listing 9.2.
 - c. Investigate the influence of σ_1 and σ_2 for the true class probability distributions, on the accuracy, the precision, and the recall, respectively. Visualize the results by plotting the error measures against σ_1 and σ_2 , respectively.
 - d. Repeat the above analysis for in-sample and out-of-sample data (see Chap. 4). Discuss the differences.

3. Determine a linear classifier that perfectly classifies the Boolean training data set $\text{True} = \{(1, 1), (0, 1), (1, 0)\}$ and $\text{False} = \{(0, 0)\}$. The classifier, to be determined, learns the logical OR-function.
4. Determine a linear classifier that perfectly classifies the Boolean training data set $\text{True} = \{(1, 1)\}$ and $\text{False} = \{(0, 0), (0, 1), (1, 0)\}$. The classifier, to be determined, learns the logical AND-function.
5. Consider the following two-class problem in a two-dimensional space, where the training data are given by:
 $\text{Class}_1 = \{(11, 11), (13, 11), (8, 10), (9, 9), (7, 7), (7, 5), (16, 3)\}$,
 $\text{Class}_2 = \{(8, 11), (15, 9), (15, 7), (13, 5), (14, 4), (9, 3), (11, 3)\}$.
 - a. Draw a scatter plot of the data using R.
 - b. Can the data be classified perfectly by a hyperplane?
 - c. Determine the decision boundary in the scatter plot for a 1nn classifier.
 - d. Classify the data point $(5.5, 11)$ by using a Gaussian classifier.
6. Consider the linear SVM-Model:
 - a. Construct the typical linear function used for SVM-classification. How can one assign an input vector x to the positive or negative class?
 - b. Suppose that the training examples are linearly separable. How many decision boundaries can separate positive data points from negative ones? Which decision boundary does the SVM algorithm calculate and why?
 - c. Does the linear SVM-Model always work in case we use noisy training data? If the answer is no, how can the basic linear SVM-Model be modified to cope with noisy training data?

Chapter 10

Hypothesis Testing



10.1 Introduction

Statistical hypothesis testing is among the most misunderstood quantitative analysis methods from data science despite its seeming simplicity. Originated from statistics, hypothesis testing has complex interdependencies between its procedural components, which makes it hard to obtain a coherent comprehension. In this chapter, we discuss the underlying logic behind statistical hypothesis testing and the formal meaning of its components and their connections. Furthermore, we discuss some examples of hypothesis tests.

Despite the fact that the core methodology of statistical hypothesis testing dates back many decades, questions regarding its interpretation and practical usage are to date under discussion [11, 31, 259, 493, 494]. Furthermore, there are constantly new statistical hypothesis tests being developed [105, 395, 396]. Given the need to make sense of the increasing flood of data, which we are currently facing in all areas of science and industry, statistical hypothesis testing provides a valuable tool for binary decision-making. Hence, a future without statistical hypothesis testing is hard to imagine.

The first method that can be considered a hypothesis test is related back to J. Arbuthnot in 1710 [191, 212]. However, the modern form of statistical hypothesis testing originated from the combination of work from R. A. Fisher, J. Neyman, and E. Pearson [164–166, 359, 360]. Examples of applications of hypothesis testing can be found in all areas of science, including medicine, biology, business, marketing, finance, psychology, and social sciences. Specific examples in biology include the identification of differentially expressed genes or pathways; in marketing, it is used to identify the efficiency of marketing campaigns or the alteration of consumer behavior; and in medicine, it has been used to assess surgical procedures, treatments, or the effectiveness of medications [102, 121, 340, 438].

In this chapter, we provide a basic discussion of statistical hypothesis testing and its components. First, we discuss the basic idea of hypothesis testing. Then

we discuss its seven main components and their interconnections. Thereafter, we address potential errors resulting from hypothesis testing and the meaning of the power. Furthermore, we show that a confidence interval complements the value provided by a test statistic. Finally, we present different examples of statistical tests that can be applied to a wide range of problems.

10.2 What Is Hypothesis Testing?

The principal idea of a statistical hypothesis test is to decide whether a data sample is “typical” or “atypical” compared to a population, assuming that a hypothesis we formulated about the population is true. Here, a “data sample” refers to a small portion of entities taken from a population, e.g., measured via an experiment, whereas the population comprises all possible entities.

In Fig. 10.1, we give an intuitive example of the basic idea of hypothesis testing. In this particular example, the population consists of all cats, and the data sample is one individual cat randomly drawn (or sampled) from the entire population. In statistics, “randomly drawn” is referred to as “sampling,” and we discussed this in detail in Chap. 4. In order to perform the comparison between the data sample and the population, one needs to introduce a quantification of the situation. In our case, this quantification consists in a mapping from a cat to a number. This number could correspond to, e.g., the body weight, body size, fluffiness, or the hair length of a cat. In statistics, this mapping is called test statistic.

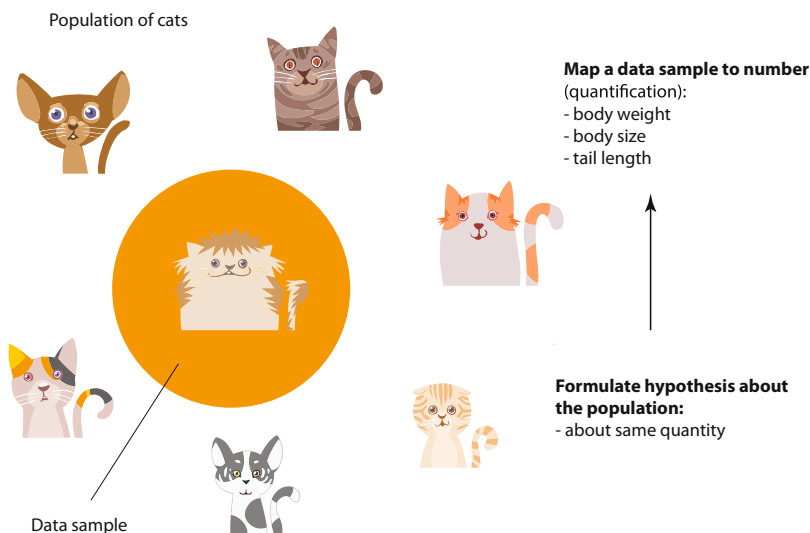


Fig. 10.1 Intuitive example explaining the basic idea underlying an one-sample hypothesis test

A key component in hypothesis testing is of course a “hypothesis.” The hypothesis is a quantitative statement formulated about the value of the test statistic for the population. In our case it could be about the body parts of a cat, e.g., body size. A particular hypothesis we can formulate is as follows: The mean body size equals 30 cm. Such a hypothesis is called the “null hypothesis,” and it is denoted as H_0 .

Now, assume that we have a population of cats, having a body size of 30 cm including some natural variations. Due to the fact that the population consists of (infinitely) many cats and for each cat we obtain such a quantification, this results in a probability distribution, called the sampling distribution for the mean body size. Here, it is important to note that our population is a hypothetical population, which obeys our null hypothesis. In other words, the null hypothesis specifies the population completely.

Now, having a numerical value of the test statistic, representing the data sample, and the sampling distribution, representing the population, we can compare them to evaluate the null hypothesis that we have formulated. From this comparison, we obtain another numerical value, called the p -value, which quantifies the typicality or atypicality of the configuration assuming the null hypothesis is true. Finally, based on the p -value, a decision is made to accept or reject the null hypothesis.

On a technical note, we want to remark that since in the above example there is only one population involved, this is called a one-sample hypothesis test. However, the principal idea extends also to hypothesis tests involving more than one population.

10.3 Key Components of Hypothesis Testing

In the following sections, we will formalize the example discussed above. In general, regardless of the specific hypothesis test one is conducting, there are seven components common to all hypothesis tests. These components are summarized in Fig. 10.2. We listed these components in the order they are entering the process when performing a hypothesis test. For this reason, they can be also considered as

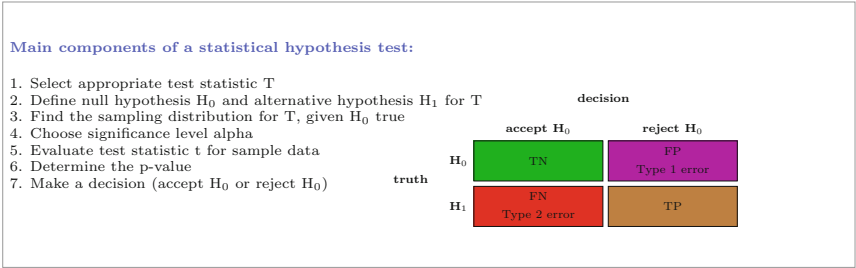


Fig. 10.2 The seven main components that are common to all hypothesis tests

steps of a hypothesis test. Due to the fact that they are interconnected with each other, their logical order is important. Overall, this means that a hypothesis test is a procedure that needs to be executed. In the following subsections, we will discuss each of these seven procedural components in detail.

10.3.1 Step 1: Select Test Statistic

Put simply, a test statistic quantifies a data sample. In statistics the term “statistic” refers to any mapping (or function) between a data sample and a numerical value. Popular examples are the mean value or the variance. Formally, the test statistic can be written as

$$t_n = T(D(n)), \quad (10.1)$$

where $D(n) = \{x_1, \dots, x_n\}$ is a data sample with sample size n . Here we denoted the mapping by T and the value we obtain by t_n . Typically, the test statistic can assume real values, i.e., $t_n \in \mathbb{R}$, but restrictions are possible.

A test statistic assumes a central role in a hypothesis test because by deciding which test statistic to use, one determines/defines a hypothesis test to a large extend. The reason is that it will enter the hypotheses we will formulate in step 2. Hence, one needs to carefully select a test statistic that is of interest and importance for the conducted study.

We would like to emphasize that in this step, we select the test statistics, but we neither evaluate it nor we use it yet. This is postponed till step 5.

10.3.2 Step 2: Null Hypothesis H_0 and Alternative Hypothesis H_1

At this step, we define two hypotheses, which are called the null hypothesis H_0 and the alternative hypothesis H_1 . Both hypotheses make statements about the population value of the test statistic and are mutually exclusive. For the test statistic $t = T(D)$, we selected in step 1, we call the population value of t as θ . Based on this, we can formulate the following hypotheses:

Null hypothesis: $H_0: \theta = \theta_0$;

Alternative hypothesis: $H_1: \theta > \theta_0$.

As one can see, due to the way the two hypotheses are formulated, the value of the population parameter θ can only be true for one statement but not for both. For instance, either $\theta = \theta_0$ is true, but then the alternative hypothesis H_1 is false, or $\theta > \theta_0$ is true, but then the null hypothesis H_0 is false.

In Fig. 10.2, we show the four possible outcomes of a hypothesis test. Each of these outcomes has a specific name that is commonly used. For instance, if the null hypothesis is false and we reject H_0 , this is called a “true positive” (TP) decision. The reason for calling it “positive” is related to the asymmetric meaning of a hypothesis test, because rejecting H_0 when H_0 is false is more informative than accepting H_0 when H_0 is true. In this case, one can consider the outcome of a hypothesis test a positive result.

The alternative hypothesis formulated above is an example for a one-side hypothesis. Specifically, we formulated a right-sided hypothesis because the alternative assumes values larger than θ_0 . In addition, we can formulate a left-sided alternative hypothesis stating:

Alternative hypothesis: $H_1: \theta < \theta_0$.

Furthermore, we can formulate a two-side alternative hypothesis, which is indifferent to the side as follows:

Alternative hypothesis: $H_1: \theta \neq \theta_0$.

Despite the variety of hypothesis tests [430], the above description holds for all of them. However, this does not mean that if you understand one hypothesis test, you understand all, but if you understand the *principle* of one hypothesis test, you understand the *principle* of all.

In order to connect the test statistic t , which is a sample value, with its population value θ , one needs to know the probability distribution of the test statistic. Because of this connection, this probability distribution received a special name and is called the *sampling distribution*. It is important to emphasize that the sampling distribution represents the values of the test statistic, assuming that the null hypothesis is true. This means that, in this case, the population value for θ is θ_0 .

Let's assume for now that we know the sampling distribution of our test statistic. By *comparing* the particular value t_n of our test statistic with the sampling distribution, we obtain a quantification for the “typicality” of this value with respect to the sampling distribution, assuming that the null hypothesis is true.

10.3.3 Step 3: Sampling Distribution

In our general discussion above about the principal idea of a hypothesis test, we mentioned that the connection between a test statistic and its sampling distribution is crucial for any hypothesis test. For this reason, we elaborate on this point in more detail, in this section.

In this section, we want to answer the following questions:

1. What is the sampling distribution?
2. How to obtain the sampling distribution?
3. How to use the sampling distribution?

For question (1): First of all, the sampling distribution is a probability distribution. It is the distribution of the test statistic T , which is a random variable, given some assumptions. We can make this statement more precise by defining the sampling distribution of the null hypothesis as follows.

Definition 10.1 (Sampling Distribution) Let $X(n) = \{X_1, \dots, X_n\}$ be a random sample from a population with $X_i \sim P_{\text{pop}} \forall i$, and let $T(X(n))$ be a test statistic. Then, the probability distribution $f_n(x|H_0 \text{ true})$ of $T(X(n))$, assuming H_0 is true, is called the sampling distribution of the null hypothesis.

Similarly, one defines the sampling distribution of the alternative hypothesis by $f_n(x|H_1 \text{ true})$. Since there are only two different hypotheses, H_0 and H_1 , there are only two different sampling distributions in this context. However, we would like to note that sampling distributions are also playing a role outside statistical hypothesis testing, e.g., for estimation theory or bootstrapping [74].

There are several points, which are important in the above definition. For this reason, we would like to highlight these explicitly. First, the distribution P_{pop} from which the random variables X_i are sampled can assume any form and is *not limited to*, e.g., a normal distribution. Second, the test statistic is a random variable itself because it is a function of random variables. For this reason, there exists a distribution from which the values of this random variable are sampled. Third, the test statistic is a function of the sample size n , and for this reason, the sampling distribution is also a function of n . That means, if we change the sample size n , we change the sampling distribution. Fourth, the fact that $f_n(x|H_0 \text{ true})$ is the probability distribution of $T(X(n))$ means that by taking an infinite number of samples from $f_n(x|H_0 \text{ true})$, in the form $T(X(n)) \sim f_n(x|H_0 \text{ true})$, we can perfectly reconstruct the distribution $f_n(x|H_0 \text{ true})$ itself. The last point allows, under certain conditions, a numerical approximation of the sampling distribution. We will take a closer look at the last point in the following example.

10.3.3.1 Examples

Suppose that we have a random sample $X(n) = \{X_1, \dots, X_n\}$ of size n where each data point X_i is sampled from a gamma distribution with parameters $\alpha = 4$ and $\beta = 2$, i.e., $X_i \sim \text{gamma}(\alpha = 4, \beta = 2)$. Hence, here, we have $P_{\text{pop}} = \text{gamma}(\alpha = 4, \beta = 2)$.

Furthermore, let's use the mean value as a test statistic, i.e.,

$$t_n = T(X(n)) = \frac{1}{n} \sum_{i=1}^n X_i. \quad (10.2)$$

In Fig. 10.3a–c, we show three examples for three different values of n (in A $n = 1$, in B $n = 3$, and in C $n = 10$) when drawing $E = 100,000$ samples of $X(n)$, from which we estimate $E = 100,000$ different mean values T . Specifically,

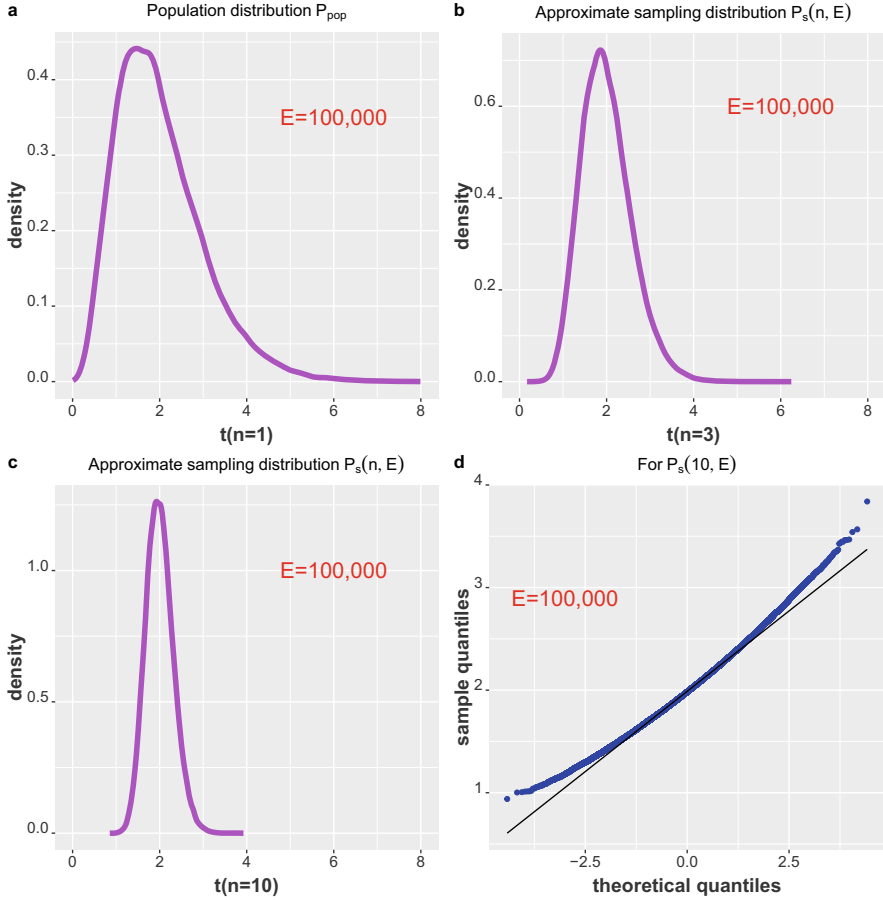


Fig. 10.3 Panels a–c show approximate sampling distributions for different values of the sample size n . Panel a shows $P_s(n = 1, E = 100,000)$, which is equal to the population distribution of X_i . Panel d shows a qq-plot comparing $P_s(n = 10, E = 100,000)$ with a normal distribution

in Fig. 10.3a–c, we show density estimates of these 100,000 values. As indicated above, in the limit of infinite number of samples E , the approximate sampling distribution $P_s(n, E)$ will become the (theoretical) sampling distribution,

$$f_n(x|H_0 \text{ true}) = \lim_{E \rightarrow \infty} P_s(n, E), \quad (10.3)$$

as a function of the sample size n .

For $n = 1$, we obtain the special case that the sampling distribution is the same as the underlying distribution of the population P_{pop} , which is in our case a gamma distribution with the parameters $\alpha = 4$ and $\beta = 2$ as shown in Fig. 10.3a. For all other $n > 1$, we observe a transformation in the distributional shape of the sampling

distribution, as shown in Fig. 10.3b and c. However, this transformation should be familiar to us because from the central limit theorem, we know that the mean of $\{X_1, \dots, X_n\}$ independent samples with mean μ and variance σ^2 follows a normal distribution with mean μ and standard deviation $\sigma/\sqrt{n}$, i.e.,

$$\bar{X} \sim N\left(\mu, \frac{\sigma}{\sqrt{n}}\right). \quad (10.4)$$

Note that this result is only strictly true when n is large.

A question to ask is, what is a large n ? In Fig. 10.3d, we show a qq-plot that demonstrates that already for $n = 10$ the resulting distribution, $P_s(n = 10, E = 100,000)$, is quite close to such a normal distribution (with the appropriate parameters).

We would like to remind that the central limit theorem holds for arbitrarily iid (independent and identically distributed) random variables $\{X_1, \dots, X_n\}$. Hence, the sampling distribution for the mean is always the normal distribution given in Eq. 10.4.

At this point, we could stop because we found the sampling distribution for our problem. However, by going a step further, we can obtain a numerical simplification. Specifically, by applying a so-called z-transformation given by

$$Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}}, \quad (10.5)$$

which transform the mean value of $\bar{X}$ to Z , we obtain a simplification because the distribution of Z is a standard normal distribution, i.e.,

$$Z \sim N(0, 1). \quad (10.6)$$

This is a simplification because the standard normal distribution does not depend on any parameter, in contrast with the above normal distribution (see Eq. 10.4), which depends on μ , σ and n .

While statistics is certainly amazing at times, it is not magic. In our case, this means that the parameters μ , σ and n did not disappear entirely, but they were merely shifted into the z-transformation (see Eq. 10.5).

Now we need to distinguish between two cases.

- Case 1: σ is known
- Case 2: σ is unknown

If we know the variance σ^2 , the sampling distribution of our transformed mean $\bar{X}$, we called Z , is a standard normal distribution. However, if we do not know the variance σ^2 , we cannot perform the z-transformation in Eq. 10.5, because this transformation depends on σ . In this case, we need to estimate the variance from the sample $\{X_1, \dots, X_n\}$ as follows:

Table 10.1 Sampling distributions of the z-score and the t-score. Here “dof” means degree of freedom

Test statistic	Sampling distribution	Prior knowledge about parameters
z-score	$N(0,1)$	σ^2 needs to be known
t-score	Student’s t-distribution, $n-1$ dof	none

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \hat{X})^2. \quad (10.7)$$

Then, we can use the estimate of the variance, s , with the so-called t-transformation

$$T = \frac{\bar{X} - \mu}{s/\sqrt{n}}. \quad (10.8)$$

Although this t-transformation is formally very similar to the z-transformation in Eq. 10.5, the resulting random variable T does not follow a standard normal distribution but a student’s t-distribution with $n - 1$ degrees of freedom (dof). We want to mention that this holds only for $X_i \sim N(\mu, \sigma)$, i.e., for normally distributed samples.

Table 10.1 summarizes the results from this section regarding the sampling distribution of the z-score (Eq. 10.5) and the t-score (Eq. 10.8).

10.3.4 Step 4: Significance Level α

The significance level α is a number between zero and one, i.e., $\alpha \in [0, 1]$. It has the meaning

$$\alpha = P(\text{Type 1 error}) = P(\text{reject } H_0 | H_0 \text{ true}), \quad (10.9)$$

which is the probability to reject the null hypothesis H_0 given that H_0 is true. Alternatively, this gives us the probability of making a Type 1 error, resulting in a false-positive decision.

When conducting a hypothesis test, we have the freedom to choose the value of α . However, when deciding about its numerical value, one needs to be aware of potential consequences. Possibly the most frequent choice of α is 0.05; however, for genome-wide association studies (GWAS), values as low as 10^{-8} are used [373].

The reason for such a wide variety of used values is in the possible consequences incurred in different application domains. Specifically, we discuss student’s t-test, correlation tests and a hypergeometric test. For GWAS, Type 1 errors can result

in wasting millions of dollars because follow-up experiments in this field are very costly. Hence, α is chosen very small to avoid Type 1 errors.

Finally, we want to remark that formally, we obtain the value of the right-hand side of Eq. 10.9 by integrating the sampling distribution, as given by Eq. 10.13. This is discussed in detail in Step 6.

10.3.5 Step 5: Evaluate the Test Statistic from Data

In this step, we connect everything discussed above with the real world, as represented by the data, since everything until this step has been theoretical. Specifically, for $D(n) = X(n) = \{x_1, \dots, x_n\}$, we estimate the numerical value of the test statistic, selected in Step 1, giving

$$t_n = T(D(n)). \quad (10.10)$$

Here, t_n represents a particular numerical value obtained from the observed data $D(n)$. Due to the fact that our data set depends on the number of samples n , the numerical value of t_n also will depend on n . This is explicitly indicated by the subscript.

10.3.6 Step 6: Determine the p -Value

For determining the p -value of a hypothesis test, we need to use the sampling distribution (see Step 3) and the estimated test statistic t_n (see Step 5). That means the p -value results from a comparison of theoretical assumptions, as represented by the sampling distribution, with real observations, as represented by the data sample, assuming H_0 is true.

This situation is visualized in Fig. 10.4 for a right-sided alternative hypothesis. The p -value is the probability for observing more extreme values than the test statistic t_n , assuming H_0 is true:

$$\begin{aligned} p &= P(\text{observe } x \text{ at least as extreme as } t \mid H_0 \text{ is true}) \\ &= P(x \geq t \mid H_0 \text{ is true}) \end{aligned} \quad (10.11)$$

Formally, the p -value is obtained by an integral over the sampling distribution:

$$p = \int_{t_n}^{\infty} f_n(x' \mid H_0 \text{ true}) dx'. \quad (10.12)$$

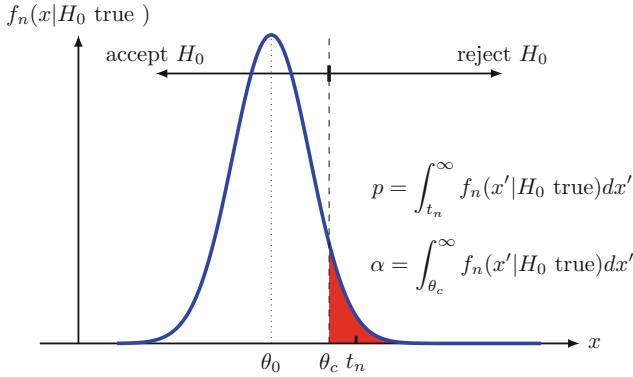


Fig. 10.4 Determining the p -value from the sampling distribution of the test statistic

We would like to emphasize that since the test statistic is a random variable, the p -value is also a random variable as it depends on the test statistic [350].

Furthermore, we can use the following integral:

$$\alpha = \int_{\theta_c}^{\infty} f_n(x'|H_0 \text{ true}) dx' \quad (10.13)$$

to solve for θ_c . That means the significance level α implies a threshold θ_c . In Step 7, we will see that the final decision to reject or accept the null hypothesis is either based on the p -value or the test statistic t_n .

Remark 10.1 The sample size, n , has an influence on the numerical analysis of a problem. For this reason, the test statistic and the sampling distribution are indexed by n . However, the sample size has no effect on the formulation and expression of the hypothesis (see Step 2) because we make statements about a population value that holds for any value of n .

10.3.7 Step 7: Make a Decision About the Null Hypothesis

In the last step, we are finally making a decision about the null hypothesis formulated in Step 2. As mentioned above, to do this, there are two alternative ways. We can make a decision based on either the p -value or the value of the test statistic t_n :

1. Decision based on the p -value:

$$\text{If } p < \alpha \implies \text{reject } H_0 \quad (10.14)$$

2. Decision based on the value of the test statistic t_n :

$$\text{If } t_n > \theta_c \implies \text{reject } H_0 \quad (10.15)$$

Here, θ_c is obtained from solving the integral in Eq. 10.13.

In case we cannot reject the null hypothesis, we accept it. For clarity, we want to mention that when we reject the null hypothesis, it means we accept the alternative hypothesis. Conversely, when we accept the null hypothesis, it means we reject the alternative hypothesis.

10.4 Type 2 Error and Power

When making binary decisions, there are a number of errors one can make. In this section, we go one step back and take a more theoretical look on a hypothesis test with respect to the possible errors that can be made. In the section “Step 2: Null Hypothesis H_0 and Alternative Hypothesis H_1 ,” we mentioned that there are two possible errors one can make, a false positive and a false negative, and when discussing Step 4, we introduced formally the meaning of a Type 1 error. Now we extend this discussion to the Type 2 error.

As discussed above, there are only two possible configurations that need to be distinguished. Either H_0 is true or it is false. If H_0 is true (respectively, false), it is equally correct to say H_1 is false (respectively, true). Now, let’s assume that H_1 is true. To evaluate the Type 2 error, we require the sampling distribution, assuming that H_1 is true. However, to perform a hypothesis test, as discussed in the previous sections (see Fig. 10.2), we do not need to know the sampling distribution, assuming that H_1 is true. Instead, we need the sampling distribution, assuming that H_0 is true because this distribution corresponds to the null hypothesis. The good news is that the sampling distribution, assuming that H_1 is true, can be easily obtained if we make the alternative hypothesis more precise. Let’s assume we are testing the following hypothesis.

Null hypothesis: $H_0: \theta = \theta_0$

Alternative hypothesis: $H_1: \theta > \theta_0$

In this case, H_0 is precisely specified because it sets the population parameter θ to θ_0 . In contrast, H_1 only limits the range of possible values for θ , but does not set it to a particular value.

To determine the Type 2 error, we need to set θ , in the alternative hypothesis, to a particular value. So let’s set the population parameter $\theta = \theta_1$ in H_1 for $\theta_1 > \theta_0$. That means we define:

Alternative hypothesis: $H_1: \theta = \theta_1$ with $\theta_1 > \theta_0$

In Fig. 10.5, we visualize the corresponding sampling distribution for H_1 and H_0 .

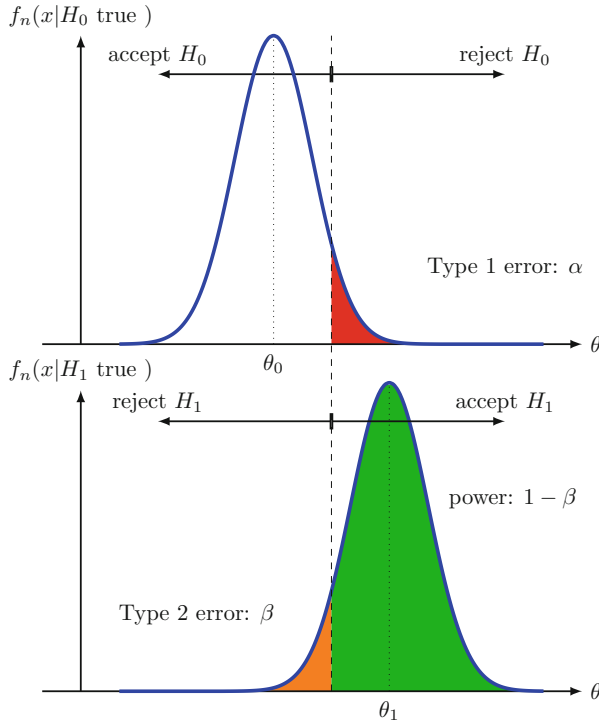


Fig. 10.5 Visualization of the sampling distribution for H_0 and H_1 assuming a fixed sample size n and setting the value of θ to θ_1 in the alternative hypothesis

If we reject H_0 when H_1 is true, this is a correct decision, and the green area, in Fig. 10.5, represents the corresponding probability for this, formally given by

$$1 - \beta = P(\text{reject } H_0 | H_1 \text{ is true}) = \int_{\theta_c}^{\infty} f_n(x' | H_1 \text{ true}) dx'. \quad (10.16)$$

In short, this probability is usually denoted by $1 - \beta$ and called the *power* of a test.

On the other hand, if we do not reject H_0 when H_1 is true, we make an error, given by

$$\beta = P(\text{Type 2 error}) = P(\text{do not reject } H_0 | H_1 \text{ is true}). \quad (10.17)$$

However, this is the same as

$$\beta = P(\text{Type 2 error}) = P(\text{accept } H_0 | H_1 \text{ is true}). \quad (10.18)$$

This is called the Type 2 error. In Fig. 10.5, we highlight the Type 2 error probability in orange.

We would like to emphasize that the Type 1 and the Type 2 errors are both long-run frequencies for repeated experiments. That means both probabilities give the error when repeating exactly the same test many times. This is in contrast with the p -value, which is the probability for a given data sample. Hence, the p -value does not allow to draw conclusions about repeated experiments.

10.4.1 Connections Between Power and Errors

From Fig. 10.5, we can see the relationship between the power ($1 - \beta$), the Type 1 error (α), and the Type 2 error (β), summarized in the table given in Fig. 10.6. Ideally, one would like to have a test with a high power and low Type 1 error and low Type 2 error. However, from Fig. 10.5, we see that these three entities are not independent from each other. Specifically, if we increase the power ($1 - \beta$) by changing α , we increase the Type 1 error (α) because this will reduce the critical value θ_c . In contrast, reducing α leads to an increase in Type 2 error (β) and a reduction in power. Hence, in practice, one needs to make a compromise between the ideal goals.

For the discussion above, we assumed a fixed sample size n . However, as we discussed in the example of section “Step 3: Sampling Distribution,” the variance of the sampling distribution depends on the sample size via the standard error (SE) as follows:

$$SE = \frac{\sigma_{\text{pop}}}{\sqrt{n}}. \quad (10.19)$$

Importantly, this provides a way to increase the power and to minimize the Type 2 error by increasing the sample size n . That means, by keeping the population means θ_0 and θ_1 unchanged, but increasing the sample size n' to a value larger than n , i.e., $n' > n$, the sampling distributions for H_0 and H_1 become narrower because their variances decrease according to Eq. 10.19. Hence, with an increased sample size, the overlap between the distributions, represented by β , is reduced. This leads to an increase in the power and a decrease in the Type 2 error for an unchanged value of

Truth	Decision	
	accept H_0	reject H_0
H_0	$1 - \alpha = P(\text{accept } H_0 H_0 \text{ true})$	$\alpha = P(\text{reject } H_0 H_0 \text{ true})$
H_1	$\beta = P(\text{accept } H_0 H_1 \text{ true})$	$1 - \beta = P(\text{reject } H_0 H_1 \text{ true})$

Fig. 10.6 Overview of the different errors, resulting from hypothesis testing and their probabilistic meaning

the significance level α . In the extreme case, $n \rightarrow \infty$, the power approaches 1 and the Type 2 error 0, for a fixed Type 1 error α .

From this discussion, the importance of the sample size in a study becomes apparent, as it is a control mechanism to influence the resulting power and the Type 2 error.

10.5 Confidence Intervals

The test statistic is a function of the data (see Step 1 in Sect. 10.3.1), and hence, it is a random variable. That means there is a variability of a test statistic because its value changes for different samples. In order to quantify the interval within which such values fall, one can use a confidence interval (CI) [23, 54].

Definition 10.2 The interval $I = [a, b]$ is called a confidence interval for parameter θ if it contains this parameter with probability $1 - \alpha$ for $\alpha \in [0, 1]$, i.e.,

$$P(a \leq \theta \leq b) = 1 - \alpha. \quad (10.20)$$

The interpretation of a CI, $I = [a, b]$, is that for repeated samples, the corresponding confidence intervals are expected to contain the true value of θ with probability $1 - \alpha$. Here, it is important to note that θ is fixed because it is a population value. What is random is the estimate of the boundaries of the CI, i.e., a and b . Hence, for repeated samples, θ is fixed, but I is a random interval.

The connection between a $1 - \alpha$ confidence interval and a hypothesis test for a significance level of α is that if the value of the test statistic falls within the CI, then we don't reject the null hypothesis. On the other hand, if the confidence interval does not contain the value of the test statistic, we reject the null hypothesis. Hence, the decisions reached by both approaches agree always with each other.

If one does not make any assumption about the shape of the probability distribution, e.g., symmetry around zero, there is an infinite number of CIs because neither the starting nor the ending values of a and b are uniquely defined, but they follow from assumptions. Frequently, one is interested in obtaining a CI for a quantile separation of the data in the form

$$P(q_{\alpha/2} \leq \theta \leq q_{1-\alpha/2}) = 1 - \alpha, \quad (10.21)$$

where $q_{\alpha/2}$ and $q_{1-\alpha/2}$ are quantiles of the sampling distribution with respect to $(100\alpha/2)\%$ and $100(1 - \alpha/2)\%$ of the data, respectively.

10.5.1 Confidence Intervals for a Population Mean with Known Variance

From the central limit theorem, we know that the sum of random variables $\hat{\theta} = 1/n \sum x_i$ is normally distributed. If we normalize this with a z-transformation as follows:

$$Z = \frac{\hat{\theta} - \mathbb{E}[\hat{\theta}]}{\text{SE}}, \quad (10.22)$$

then, Z follows a standard normal distribution, i.e., $N(0, 1)$, where SE is the standard error of $\hat{\theta}$ given by

$$\text{SE} = \frac{\sigma}{\sqrt{n}}. \quad (10.23)$$

Adjusting the definition of a confidence interval in Eq. 10.21 to our problem gives

$$P(q_{\alpha/2} \leq Z \leq q_{1-\alpha/2}) = 1 - \alpha \quad (10.24)$$

with

$$q_{\alpha/2} = -z_{\alpha/2}; \quad (10.25)$$

$$q_{1-\alpha/2} = z_{\alpha/2}. \quad (10.26)$$

Here, the values of $\pm z_{\alpha/2}$ are obtained by solving the equations for a standard normal distribution probability

$$P(Z < -z_{\alpha/2}) = \alpha/2; \quad (10.27)$$

$$P(Z > z_{\alpha/2}) = \alpha/2. \quad (10.28)$$

Using these and solving the inequality in Eq. 10.24 for the expectation value gives the confidence interval $I = [a, b]$ with

$$a = \hat{\theta} - z_{\alpha/2} \sigma(\hat{\theta}) = \hat{\theta} - z_{\alpha/2} \frac{\sigma}{\sqrt{n}}; \quad (10.29)$$

$$b = \hat{\theta} + z_{\alpha/2} \sigma(\hat{\theta}) = \hat{\theta} + z_{\alpha/2} \frac{\sigma}{\sqrt{n}}. \quad (10.30)$$

Here, we assumed that σ is known. Hence, the above CI is valid for a z-test.

10.5.2 Confidence Intervals for a Population Mean with Unknown Variance

If we assume that σ is not known, then the sampling distribution of a population mean becomes the Student's t-distribution. For this, σ needs to be estimated from samples using the sample standard deviation s . In this case, a similar derivation as above results in

$$a = \hat{\theta} - t_{\alpha/2} \frac{s}{\sqrt{n}}; \quad (10.31)$$

$$b = \hat{\theta} + t_{\alpha/2} \frac{s}{\sqrt{n}}. \quad (10.32)$$

Here, $\pm t_{\alpha/2}$ are critical values for a student's t-distribution, obtained similarly as in Eqs. 10.27 and 10.28. Such a CI is valid for a t-test; see Sect. 10.6.1.

10.5.3 Bootstrap Confidence Intervals

When a sampling distribution is not given in an analytical form, numerical approaches need to be used. In such a situation, a CI can be numerically obtained via nonparametric Bootstrap [128]. This is the most generic way to obtain a CI. Using the augmented definition in Eq. 10.21, for any test statistic $\hat{\theta}$, the CI can be obtained from

$$P(\hat{q}_{\alpha/2} \leq \hat{\theta} \leq \hat{q}_{1-\alpha/2}) = 1 - \alpha, \quad (10.33)$$

where the quantiles $\hat{q}_{\alpha/2}$ and $\hat{q}_{1-\alpha/2}$ are directly obtained from the data, resulting in $I = [\hat{q}_{\alpha/2}, \hat{q}_{1-\alpha/2}]$. Such a confidence interval can be used for any statistical hypothesis test.

We would like to emphasize that in contrast with Eq. 10.21, here, the quantiles $\hat{q}_{\alpha/2}$ and $\hat{q}_{1-\alpha/2}$ are estimates of the quantiles $q_{\alpha/2}$ and $q_{1-\alpha/2}$ from the sampling distribution. Hence, the obtained CI is merely an approximation.

An example of this is shown in Listing 10.1.

10.6 Important Hypothesis Tests

In the following, we discuss some important hypothesis tests that are frequently used in many application domains.

Listing 10.1: Bootstrap confidence intervals with R

```

library(boot)
n <- 10
alpha <- 4
beta <- 2

b.mean = function(x, indices){
  return(mean( x[indices] ))
}

x <- rgamma(n, shape = alpha, rate = beta)
mean.boot = boot(x, statistic=b.mean, R=10000)

boot.ci(boot.out = mean.boot, type = "basic")

```

10.6.1 Student's t-Test

The student's t-test, also known as the t-test in short, can be used to test hypothesis about the mean. In Sect. 10.3.3.1, we have already seen that a t-transformation shows that the sampling distribution of a t-score is given by student's t-distribution, which explains the name of the test. For this reason, the test statistics of a t-test is a t-score given by

$$T = \frac{\bar{X} - \mu}{s/\sqrt{n}}, \quad (10.34)$$

$$\text{with } \bar{X} = \frac{1}{n} \sum_{i=1}^n X_i. \quad (10.35)$$

Here, $\{X_i\}_{i=1}^n$ are n observations drawn from an underlying distribution associated with a population. If the variance, σ , of this population is known, the t-score would become a z-score (see Sect. 10.3.3.1) and the t-test a z-test. However, for real-world problems, this is usually not the case. This means that s needs to be estimated from the observations as follows:

$$s = \sqrt{\frac{\sum_{i=1}^n X_i^2 - \frac{(\sum_{i=1}^n X_i)^2}{n}}{n-1}}. \quad (10.36)$$

10.6.1.1 One-Sample t-Test

There are different versions of a t-test, but the simplest one is the one-sample test. In this case, there is just one population, f_1 , from which observations are drawn, i.e., $x_i \sim f_1$, and the hypotheses are formulated as follows:

$$H_0 : \mu = \mu_0; \quad (10.37)$$

$$H_1 : \mu > \mu_0. \quad (10.38)$$

Practically, a t-test is used as follows. Given the n observations $\{X_i\}_{i=1}^n$, we first estimate $\bar{X}$ and s from Eqs. 10.35 and 10.36. Then, we estimate the test statistics from Eq. 10.34 using our null hypothesis, i.e.,

$$t_n = \frac{\bar{X} - \mu_0}{s/\sqrt{n}}. \quad (10.39)$$

This gives a numerical value which can be used to integrate along the sampling distribution, as specified by your alternative hypothesis. Above we used a right-sided alternative hypothesis, which corresponds to an integration as shown in Fig. 10.4, i.e.,

$$p = \int_{t_n}^{\infty} f_{St-t}(x) dx. \quad (10.40)$$

Here, f_{St-t} is student's t-distribution corresponding to the sampling distribution.

Using R, a t-test can be easily performed as shown in the example in Listing 10.2.

Listing 10.2: One-sample Student's t-test with R

```
> X <- rnorm(5)
> X
[1] 0.8451536 -0.7697120 1.0000454 1.1227266 0.3316087
>
> t.test(X, alternative="greater", mu = 0.0)

One Sample t-test

data: X
t = 1.4614, df = 4, p-value = 0.1089
alternative hypothesis: true mean is greater than 0
95 percent confidence interval:
 -0.2321151      Inf
sample estimates:
mean of x
0.5059645
```

10.6.1.2 Two-Sample t-Test

An extension of student's t-test to two samples is needed when there are two independent underlying populations, f_1 , f_2 , from which samples are drawn. Specifically, let's assume that we have n observations $X = \{X_i\}_{i=1}^n$ from population one (f_1) and m observations $Y = \{Y_i\}_{i=1}^m$ from population two (f_2). In this case, we want to formulate the hypothesis about both populations in the form:

$$H_0 : \mu_1 = \mu_2 \quad (10.41)$$

$$H_1 : \mu_1 > \mu_2 \quad (10.42)$$

Here, μ_1 corresponds to the population mean of the first population and μ_2 to the population mean of population two. In this case, the test statistics is given by

$$t_n = \frac{\bar{X} - \bar{Y}}{\sqrt{\frac{s_X^2}{n} + \frac{s_Y^2}{m}}}, \quad (10.43)$$

where $\bar{X}$, $\bar{Y}$, and s_X , s_Y are estimated according to Eqs. 10.35 and 10.36.

One technical detail, we would like to note that the above test statistic can be used for unequal sample sizes n and m and unequal variances s_X and s_Y . In this case, the t-test is formally called the Welch's t-test, and this is the default when using the function `t.test()` available in R.

Listing 10.3: Two-sample Student's t-test with R

```
> X <- rnorm(5, mean=0, sd=1)
> Y <- rnorm(5, mean=0, sd=0.5)
> X
[1] 0.56300873 1.07737677 -0.98127418 0.03698645 0.94406400
> Y
[1] -0.27336372 -0.23651343 0.05721891 0.63782333 0.38833657
> t.test(X, Y, alternative="greater")

Welch Two Sample t-test

data: X and Y
t = 0.51582, df = 5.7055, p-value = 0.3127
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.597822      Inf
sample estimates:
mean of x mean of y
0.3280324 0.1147003
```

10.6.1.3 Extensions

A generalization of student's t-test, called *Hotelling's t-squared test*, is used for the multivariate. This is the case when the mean value of a population, μ , is not given by a scalar value but a vector, i.e., $\mu \in \mathbb{R}^p$, with $p > 1$. In such a case, observations are drawn from, e.g., a multivariate normal distribution, $X \sim N(\mu, \Sigma)$, where Σ is the covariance matrix.

Another extension is needed when there are more than two populations from which observations are drawn. For instance, for testing the null hypothesis

$$H_0 : \mu_1 = \mu_2 = \dots \mu_k \quad (10.44)$$

with $k > 2$ and $k \in \{3, 4, \dots\}$, an *ANOVA (Analysis of Variance)* test is used.

10.6.2 Correlation Tests

Another test statistic frequently of interest is the correlation value. This statistic measures the association between two variables. Given two variables $X = \{X_i\}_{i=1}^p$ and $Y = \{Y_i\}_{i=1}^p$, one can estimate the sample Pearson product-moment correlation coefficient by

$$r = \frac{S_{x,y}}{\sqrt{S_{x,x}}\sqrt{S_{y,y}}} \quad (10.45)$$

with

$$S_{x,y} = \sum_{i=1}^p x_i y_i - \frac{(\sum_{i=1}^p x_i)(\sum_{i=1}^p y_i)}{p} \quad (10.46)$$

$$S_{x,x} = \sum_{i=1}^p x_i^2 - \frac{(\sum_{i=1}^p x_i)^2}{p} \quad (10.47)$$

$$S_{y,y} = \sum_{i=1}^p y_i^2 - \frac{(\sum_{i=1}^p y_i)^2}{p} \quad (10.48)$$

For a two-sided alternative, the hypothesis about the population correlation is formulated as follows:

$$H_0 : \rho = 0; \quad (10.49)$$

$$H_1 : \rho \neq 0. \quad (10.50)$$

In Listing 10.4, we provided an example of this case since for a simple linear regression, $y \sim \beta_0 + \beta_1 x$, one can show that the regression coefficient β_1 corresponds to the correlation coefficient, r . The results are plausible because according to the way we simulated the data, the null hypothesis is false.

Listing 10.4: Pearson product-moment correlation test with R

```
p <- 100
X <- rnorm(p, mean=1, sd=1)
Y <- 0.3 * X + rnorm(p, mean=0, sd=1)
cor.test(X, Y, alternative="two.sided", method="pearson")
# output
Pearson's product-moment correlation

data: X and Y
t = 3.5161, df = 98, p-value = 0.0006651
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.1480039 0.4983491
sample estimates:
      cor
0.3346923
```

Aside from the Pearson correlation, there is also the Spearman's correlation. The Spearman's rank-order correlation is the nonparametric version of the Pearson product-moment correlation. Spearman's correlation coefficient measures the strength between two ranked variables. That means such values are at least on an ordinal scale (see Sect. 10.6.4). This implies that even when the observations are real valued, i.e., $x_i, y_i \in \mathbb{R}$, Spearman's rank-order correlation uses only information about the rank order of these values. In R, this information is obtained using the function *rank()*.

An example for a Spearman rank-correlation test is shown in Listing 10.5. Here, we included two versions of the test that give both the same results.

Listing 10.5: Spearman rank-correlation test with R

```
p <- 20
X <- rnorm(p, mean=1, sd=1)
Y <- 0.3 * X + rnorm(p, mean=0, sd=1)
#cor.test(X, Y, alternative="two.sided", method="spearman")
cor.test(rank(X), rank(Y), alternative="two.sided",
         method="spearman")
# output
Spearman's rank correlation rho

data:  rank(X) and rank(Y)
S = 668, p-value = 0.02707
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.4977444
```

The hypothesis tested in Listing 10.5 can be formulated as follows:

H_0 : There is no (monotonic) association between the two variables.

H_1 : There is a (monotonic) association between the two variables.

In Fig. 10.7, we show an example making the difference between Pearson and Spearman correlation clear. The alternative form of the Spearman correlation leads to the same result, because Spearman correlation utilizes only the ranks of the observations X and Y .

X	Y	rank(X)	rank(Y)	
-0.26	-0.18	2	2	Pearson correlation: $r = 0.806$ Spearman correlation: $r_S = 0.90$
0.94	0.38	3	3	
1.27	2.17	5	4	
1.11	2.22	4	5	
-1.36	-0.24	1	1	

R code:

```
Pearson correlation: cor(X, Y, method="pearson")
Spearman correlation: cor(X, Y, method="spearman")
Spearman correlation: cor(rank(X), rank(Y), method="spearman")
```

Fig. 10.7 An example of Pearson and Spearman correlation. The alternative form of the Spearman correlation leads to the same result

10.6.3 Hypergeometric Test

The hypergeometric test, also known as Fisher's exact test, is used to determine the enrichment of one subpopulation in another.

In order to explain the idea behind a hypergeometric test, we consider a problem frequently studied in genomics. Suppose that an experiment is conducted involving a large number of genes and the experiment shows that some of these genes are active. Such genes are called differentially expressed. The question we want to answer is whether the genes that are differentially expressed are overrepresented in a specific biological process. An alternative formulation used to describe this is to ask whether the differentially expressed genes are enriched in this biological process.

The basic idea of a hypergeometric test is shown in Fig. 10.8. As one can see, there are different colors for the genes shown as dots and (big) circles that enclose the dots. The reason for this is that each gene/dot is characterized by two properties. The first property distinguishes if a gene is differentially expressed or not. If a gene is differentially expressed, it is enclosed by the red circle; otherwise, it is outside the red circle and enclosed by the purple circle. The second property distinguishes if a gene is a member of a biological process or not. If it is a member of a biological process, its dot is shown in green, otherwise in orange.

More formally, one can summarize the above groupings of genes in a tabular form. In Fig. 10.9, we show a contingency table corresponding to the visualization in Fig. 10.8. The two properties of the genes, i.e., “differentially expressed” and “member in biological process,” correspond to the rows and columns, respectively. Specifically, the total number of differentially expressed genes is given by n_{1+} , and the number of all genes is n . Hence, n_{2+} corresponds to the number of genes that are not differentially expressed. Similarly, the total number of genes, which are

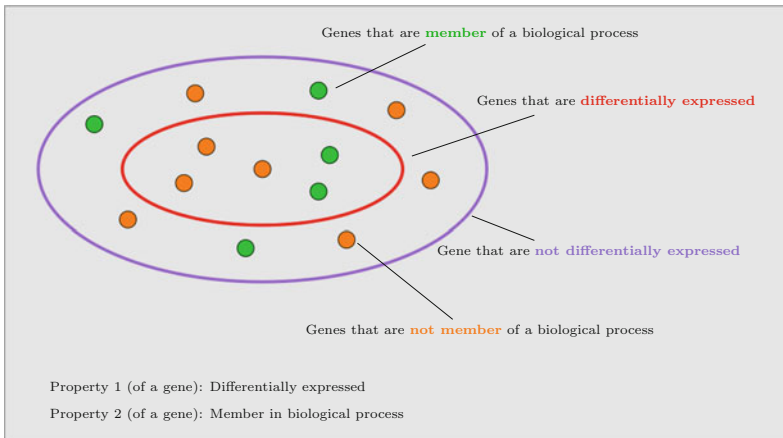


Fig. 10.8 For a hypergeometric test, one needs to distinguish between two properties. This is visualized by the colors given to the dots and circles

Fig. 10.9 Contingency table that summarizes a hypergeometric test as visualized in Fig. 10.8. The shown colors are the same as in Fig. 10.8

Differentially expressed	Member in biological process		
	Yes	No	Total
Yes	n_{11}	n_{12}	n_{1+}
No	n_{21}	n_{22}	n_{2+}
Total	n_{+1}	n_{+2}	n

Fig. 10.10 Contingency table that summarizes a hypergeometric test as visualized in Fig. 10.8. The shown colors are the same as in Fig. 10.8

Differentially expressed	Member in biological process		
	Yes	No	Total
Yes	x	y	n_{1+}
No	n_{21}	n_{22}	n_{2+}
Total	n_{+1}	n_{+2}	n

associated with the biological process of interest, is given by n_{+1} , and the number of genes, which are not associated with this biological process, is given by n_{+2} .

We would like to highlight that there are a number of constraints that hold for the rows and columns. These are given by:

$$n_{1+} = n_{11} + n_{12}; \quad (10.51)$$

$$n_{2+} = n_{21} + n_{22}; \quad (10.52)$$

$$n_{+1} = n_{11} + n_{21}; \quad (10.53)$$

$$n_{+2} = n_{12} + n_{22}; \quad (10.54)$$

$$n = n_{1+} + n_{2+}; \quad (10.55)$$

$$n = n_{+1} + n_{+2}. \quad (10.56)$$

Hence, the sums of the row and columns are conserved. This implies that from the four numbers, i.e., n_{11} , n_{12} , n_{21} , and n_{22} follow all other entities.

10.6.3.1 Null Hypothesis and Sampling Distribution

In order to emphasize the entities of interest for formulating a proper statistical hypothesis, we show a revised contingency table in Fig. 10.10. The crucial point is to realize that to address our original question about the enrichment of differentially expressed genes that are also involved in a biological process, the entities given by x and y are important. Specifically, since both numbers are random variables, there are underlying probability distributions from which those numbers are drawn. Both distributions correspond to binomial distributions, however, characterized by different parameters, i.e.,

$$x \sim \text{Binom}(X = x | n_{+1}, p_1), \quad (10.57)$$

$$y \sim \text{Binom}(Y = y | n_{+2}, p_2). \quad (10.58)$$

The binomial distribution for a random variable, Z , is given by

$$P(Z = k) = \text{Binom}(k|n, p) = \binom{n}{k} p^k (1 - p)^{n-k}. \quad (10.59)$$

Importantly, the parameter p defines the probability to draw a gene with a certain property. In our case, the property is either to be differentially expressed, given by p_1 , or not, given by p_2 . At this point, we need to realize that this is the test statistic we are looking for to describe our initial hypothesis formally. That means, we can formulate the null hypothesis as:

$$H_0 : p_1 = p_2. \quad (10.60)$$

Assuming that the null hypothesis is true, i.e., $p = p_1 = p_2$, the independence of X and Y , and $z = x + y = n_{1+}$, we derive the null distribution as follows:

$$P(X = x | X + Y = z) = \frac{P(X = x, X + Y = z)}{P(X + Y = z)} \quad (10.61)$$

$$= \frac{P(X = x, Y = z - x)}{P(X + Y = z)} \quad (10.62)$$

$$= \frac{P(X = x)P(Y = z - x)}{P(X + Y = z)} \quad (10.63)$$

$$= \frac{\binom{n+1}{x} p^x (1 - p)^{n+1-x} \cdot \binom{n+2}{z-x} p^{z-x} (1 - p)^{n+2-z+x}}{\binom{n+1+n+2}{z} p^z (1 - p)^{n+1+n+2-z}} \quad (10.64)$$

$$= \frac{\binom{n+1}{x} \cdot \binom{n+2}{z-x}}{\binom{n+1+n+2}{z}} \quad (10.65)$$

Equation 10.65 is nothing but a hypergeometric distribution. Hence, the null distribution corresponding to the null hypothesis $H_0 : p_1 = p_2$ is a hypergeometric distribution given by

$$P(X = x | X + Y = n_{1+}) = \frac{\binom{n+1}{x} \cdot \binom{n+2}{z-x}}{\binom{n+1+n+2}{z}}. \quad (10.66)$$

Depending on the formulation of the alternative hypothesis H_A , one obtains a p -value. Specifically, for the alternative hypothesis

$$H_A : p_1 > p_2, \quad (10.67)$$

Fig. 10.11 Numerical values of the contingency table corresponding to the example shown in Fig. 10.8

Differentially expressed	Member in biological process		
	Yes	No	Total
Yes	$x = 3$	$y = 9$	$n_{1+} = 12$
No	$n_{21} = 13$	$n_{22} = 4$	$n_{2+} = 17$
Total	$n_{+1} = 16$	$n_{+2} = 13$	$n = 29$

which corresponds to an enrichment of X compared to Y , the p -value is given by

$$p = P(X \geq n_{11} | X + Y = n_{1+}) = \sum_{x=n_{11}}^{n_{1+}} P(X = x | X + Y = n_{1+}). \tag{10.68}$$

Now, we have everything we need to conduct a hypergeometric test.

10.6.3.2 Examples

Let’s consider the example visualized in Fig. 10.8. The numerical values of the contingency table are shown in Fig. 10.11. From these values, we can estimate the p -value either by directly utilizing a hypergeometric distribution or by using the function `fisher.test()` provided in R.

Listing 10.6: Hypergeometric test with R

```
#1. Direct estimation
p <- 0. # p-value
for(i in 3:12){
  p <- p + dhyper(i, 16, 13, 12)
}

#2. Utilize an R function
dat <- matrix(c(3, 9, 13, 4), 2)
res <- fisher.test(dat, alternative = "greater")
p <- res$p.val
```

The application of both methods, shown in Listing 10.6, results in $p = 0.9993$.

10.6.4 Finding the Correct Hypothesis Test

The above tests are just few examples. In fact, there are many different hypothesis tests, and it is impossible to discuss them all. Practically, the question is how to find the correct test for a given problem? For this reason, every comprehensive book about hypothesis testing distinguishes them according to three properties.

First, how many populations are involved? For instance, for one population, one needs a one-sample test, for two populations a two-sample test, etc. Second, what is the hypothesis that should be tested? This defines the test statistic and determines the sampling distribution. Third, what is the level of measurement of the data? In statistics, one distinguishes between nominal data (categorical data), ordinal data (rank-order data), interval data, and ratio data. Moving from nominal data to the other levels provides more and more information.

- *Nominal data*: Data points are merely identifiers, for example, license plate of a car.
- *Ordinal data*: Data points have properties of nominal data; plus they can be ordered. However, differences between data points have no meaning, for example, final position in a car race.
- *Interval data*: Data points have properties of ordinal data, and equal intervals have the same meaning, for example, physical position of the cars finishing a race.
- *Ratio data*: Data points have properties of interval data, and they have a true zero point, for example, weight of a car.

The understanding of all data types is straightforward, except the ratio data. Let's consider two examples to show what "have a true zero point" means.

Example: (weight) Suppose that we have two bags given by $X = 70$ kg (kilogram) and $Y = 140$ kg. From this, it follows that $Y/X = 2$, i.e., bag Y is twice as heavy as bag X . By going to the unit "stone" instead of "kg," this result does not change because 70 kg corresponds to 11 stone and 140 kg corresponds to 22 stone.

Example: (temperature) Suppose that the temperatures we measure at two different locations are $X = 20$ F (Fahrenheit) and $Y = 40$ F, respectively. From this, it follows that $Y/X = 2$. However, going to the unit "Celsius" (C)

$$X = 20\text{F} \rightarrow Y = -6.6\text{C} \quad (10.69)$$

$$Y = 40\text{F} \rightarrow Y = 4.4\text{C} \quad (10.70)$$

one obtains $Y/X = -1.5$. Also, for $X = 0$ kg, one obtains $X = 0$ stone; however, $X = 0$ C does not mean $X = 0$ F ($X = 0\text{C}$ corresponds to $X = 32\text{F}$). This means that for temperature, the data do not have a true zero because depending on the unit, its meaning changes. Also, in this case, it does not make sense to say that at location X , it is twice as warm as at location Y since by changing the unit, this assertion is no longer valid.

A comprehensive book about hypothesis tests is [431]. There, one can find a very large collection of different tests that can be distinguished with the three properties we have discussed earlier. In total 32 main test categories were presented and discussed over nearly 1200 pages.

10.7 Permutation Tests

The above hypothesis tests, e.g., the t-test or Fisher's exact test, are examples for parametric tests. In general, a parametric test is a hypothesis test which makes certain assumptions about the underlying distribution(s), which can be used to derive analytical solutions for the sampling distribution. Analytical solution means that the functional form of the sampling distribution is precisely given. For instance, for the t-test, this resulted in student's t-distribution and for Fisher's exact test in the hypergeometric distribution. Whenever it is possible and justified, this results in elegant solutions. However, practically, there are two problems. First, this is not always possible, and, second, the derivation is mathematically demanding as we have seen for the derivation of the hypergeometric distributions based on binomial distributions.

Luckily, there is another category of hypothesis tests that avoids the above problems, called permutation tests. In contrast with the t-test or the Fisher's exact test, a permutation test is an approach rather than a particular method that can be applied to general two-sample problems. Also, a permutation test does not require any assumption about the underlying distribution(s). For this reason, permutation tests are nonparametric tests that provide a numerical approximation of a sampling distribution rather than its analytical solution.

The underlying idea of a permutation test is simple, and we provide its visualization in Fig. 10.12. Suppose that we have data from two populations, corresponding to sample 1 given by $X = \{X_i\}_{i=1}^n$ and sample 2 given by $Y = \{Y_i\}_{i=1}^m$. Based on the data, a test statistic is estimated. An example could be the mean value given by

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i, \quad (10.71)$$

$$\bar{Y} = \frac{1}{m} \sum_{i=1}^m Y_i, \quad (10.72)$$

$$t_{n,m} = \bar{X} - \bar{Y}. \quad (10.73)$$

Based on this, a null and alternative hypothesis can be formulated as follows:

$$H_0 : \mu_1 = \mu_2, \quad (10.74)$$

$$H_1 : \mu_1 > \mu_2, \quad (10.75)$$

where μ_1 and μ_2 are the mean values of population 1 and population 2, respectively.

The assumption made by a permutation test is that, given that the null hypothesis is true, the observations have an equal probability to be drawn from population one or population two. This assumption can be used to randomize the data. In Fig. 10.12, we show two realizations of such a randomization. Importantly, for

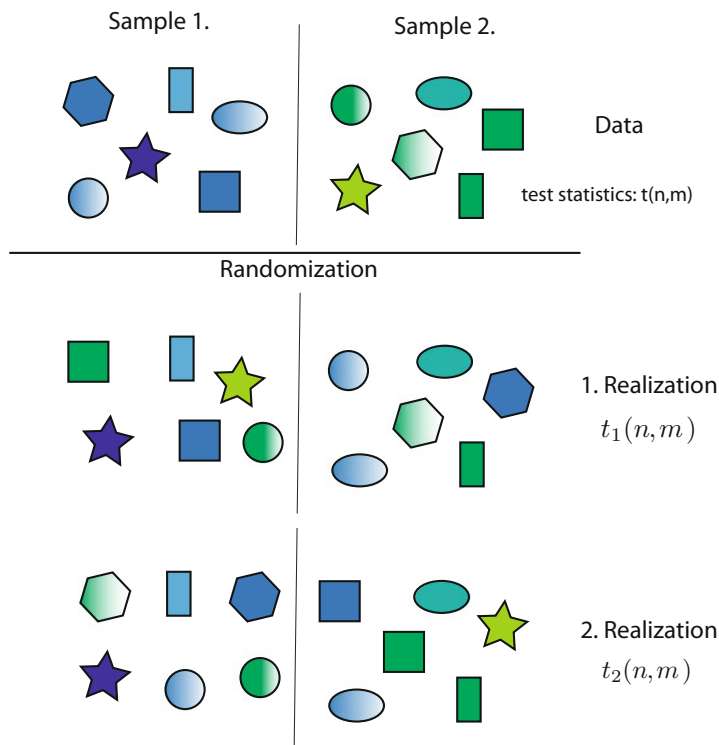


Fig. 10.12 A visualization of the basic idea of a randomization test. Shown are two realizations of randomized data

each realization, the test statistic can be estimated, i.e., $t_1(n, m)$ and $t_2(n, m)$. Hence, by combining such estimates from many randomizations $\{t_r(n, m)\}_{r=1}^R$, the corresponding sampling distribution is estimated.

In total there are $(n + m)!$ different realizations. When all possible realizations are used, the test is called a permutation test. However, if only a random subset of all possible realizations is used, then the test is called a randomization test, which is an approximation of a permutation test.

In Listing 10.7, we present an example using the mean as test statistic. As one can see, the estimation of the p -value involves merely the counting of realizations that are larger than $t(n, m)$ as this approximates the integration over the sampling distribution.

In the example in Listing 10.7, we used $R = 1000$ realization. However, the general question is what value of R should be used when performing a randomization test? Since the second smallest possible p -value is $1/R$ (the smallest p -value is zero), the value of R determines the resolution of possible p -values. For $R = 1000$, this gives $p_{res} = 0.001 = 1/R$, which is sufficient for (single) hypothesis tests and a significance level α larger or equal to p_{res} . Hence, in general,

Listing 10.7: Permutation test with R

```

n <- 10; m <- 5;
X <- rnorm(n, mean=0, sd=1)
Y <- rnorm(m, mean=0.5, sd=1)
t <- mean(X) - mean(Y)

R <- 1000 # Realizations
tr <- vector(mode = "numeric", length = R)
for(i in 1:R){
  data_r <- sample(c(X, Y))
  tr[i] <- mean(data_r[1:n]) - mean(data_r[(n+1):(n+m)])
}
p <- length(which(tr > t))/R # p-value for right-sided
alternativehypothesis

```

the value of R needs to be chosen as follows:

$$\text{Given the significance level } \alpha \rightarrow R \geq \frac{1}{\alpha}. \quad (10.76)$$

10.8 Understanding vs Applying Hypothesis Tests

Finally, we present an example to demonstrate the difference between “understanding” and “applying” a hypothesis test. In Fig. 10.13, we show a worked example for a t-test.

On the left-hand side of Fig. 10.13, a summary of the test is presented, and on the right-hand side of Fig. 10.13, we show the numerical solution to the problem

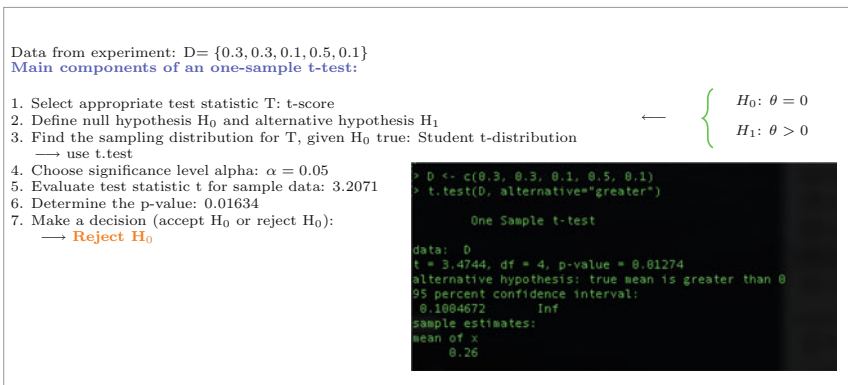


Fig. 10.13 Example of a one-sample t-test conducted using the statistical programming language R

using R. The script of the solution is only two lines. In the first line, the data sample is defined, and in the second, the hypothesis test is conducted. Then, the function `t.test()` has arguments that specify the used data and the type of the alternative hypothesis. In our case, we are using a right-sided alternative indicated by “`alternative=‘greater’`.” In addition, the null hypothesis needs to be specified. In our case, we used the default, which is $\theta = 0$; however, by using the argument “`mu`,” one can set different values.

From this example, one can learn the following. First, the practical execution of a hypothesis test using R is very simple. In fact, every hypothesis test assumes a form similar to the provided example. Second, with this simplicity, all the complexity of a hypothesis test, discussed in the previous sections of this chapter, is hidden behind the abstract computer command “`t.test()`.” Therefore, a deeper understanding of a hypothesis test cannot be gained by the practical execution of a black box (in the above example “`t.test()`” is the black-box). The last point may seem counterintuitive, especially, if one skips the above discussion. This is one of the causes of the widespread misunderstanding of statistical hypothesis tests, in general. Hence, to understand the logic and mechanics of the inner working of hypothesis testing, one needs to open up the black box and study its seven main components one by one.

10.9 Historical Notes and Misinterpretations

We want to end this chapter by providing some historical information about the development of statistical hypothesis testing, useful for understanding some problems that accompany this method.

The modern formulation of statistical hypothesis testing, as presented in this chapter, has not been established as one theory, but it evolved from two separately introduced theories. The first is due to Fisher [164] and the second due to Neyman and Pearson [360]. Since the 1960s, a unified form, which is sometimes called Null Hypothesis Significance Testing (NHST) [367, 455], was established. Fisher introduced the concept of a p -value, while Neyman and Pearson introduced the alternative hypothesis as complement to the null hypothesis, the Type I and Type II errors, as well as the power.

There is an ongoing discussion about the differences of both concepts (see, e.g., [39, 302, 383]), which is in general very difficult to follow because these involve also philosophical interpretations of these theories. Unfortunately, these differences are of interest not only for historical reasons, but they also lead to contaminations and misunderstandings of the modern formulation of statistical hypothesis testing. In particular, arguments are often taken out of context, whereas properties differ between the various theories [203, 209]. For this reason, we discuss some of these properties in the following:

1. Is the p -value the probability that the null hypothesis is true given the data?
No, it is the probability of observing more extreme values than the test statistic, if the null hypothesis is true, i.e., $P(x \geq |t| \mid H_0 \text{ is true})$ (see Eq. 10.11). Hence, one assumes already that H_0 is true for obtaining the p -value. Instead, the question aims to find $P(H_0 \mid D)$.
2. Is the p -value the probability that the alternative hypothesis is true given the data?
No, see question (1). This would be $P(H_1 \mid D)$.
3. If the null hypothesis is rejected, is the p -value the probability of your rejection error?
No, the rejection error is the Type I error given by α .
4. Is the p -value the probability to observe our data sample given the null hypothesis is true?
No, this would be the Likelihood.
5. If one repeats an experiment, does one obtain the same p -value?
No, because p -values do not provide information about the long-run frequencies of repeated experiments like the Type I or Type II errors. Instead, they give the probability resulting from comparing the test statistic (as a function of the data), while the null hypothesis is assumed to be true.
6. Does the p -value give the probability that the data were produced by random chance alone?
No, despite the fact that the data were produced by assuming that H_0 is true.
7. Does the same p -value from two studies provide the same evidence against the null hypothesis?
Yes, but only in the very rare case where everything in the two studies and the formulated hypotheses is identical, including the sample sizes. In any other case, p -values are difficult to compare with each other, and no conclusion can be drawn.

We think that many of the above confusions are a result from *verbal* interpretations of the theory while neglecting the *mathematical* definitions of used entities. This is understandable since many people are interested in the application of statistical hypothesis testing but not in the underlying probability theory. A related problem may be that a hypothesis test is exactly set up to answer one question (based on a data sample to reject a null hypothesis). However, there are certainly many more questions experimentalists would like to have answers for, but a statistical hypothesis test is not designed to answer more than one question at a time.

In general, to address questions regarding interpretations of a hypothesis test, it is a good strategy to start answering such questions by looking at the basic definitions of the involved entities, because only these are exact and provide insightful interpretations.

10.10 Summary

In this chapter, we provided a basic introduction to the concepts underlying statistical hypothesis testing. Our goal was to make the seven individual components, upon which a general hypothesis test is based on, as clear as possible. Furthermore, we discussed some hypothesis tests that are frequently used in applications, e.g., student's t-test or a hypergeometric test (also known as Fisher's exact test).

In our experience, gaining an understanding of hypothesis testing is usually considered difficult. For this reason, we were aiming at an accessible level of description, and we presented the bare backbone of the method. We tried to avoid application domain-specific jargon in order to make the knowledge transfer easier to different application areas in data science, including biomedical science, social sciences, marketing, medicine, or psychology [85, 138, 254, 364, 396, 415].

Learning Outcome 10: Statistical Hypothesis Testing

A statistical hypothesis test compares a test statistic with a reference population distribution (sampling distribution) by assuming that the null hypothesis is true to decide if the null hypothesis about the test statistic is true.

Finally, we would like to note that in many practical applications, one is required to perform not only one but multiple hypothesis tests simultaneously, for instance, to identify the differential expression of genes when comparing the effect of a medication or to identify which marketing campaign is more successful. In such a situation, one needs to apply a *multiple testing correction (MTC)* for controlling the resulting errors [32, 143, 157]. This is a highly nontrivial problem and a complex topic for itself, which can lead to erroneous outcomes if not addressed properly [37]. For this reason, we discuss multiple testing corrections in detail in Chap. 15.

10.11 Exercises

1. Discuss the seven main components of a statistical hypothesis test.
2. How many different alternative hypotheses can be formulated? Make a sketch similar to Fig. 10.4, and define the corresponding integrals for obtaining the p -values.
3. Plot the gamma distribution, $\text{gamma}(\alpha = 4, \beta = 2)$, corresponding to Fig. 10.3a. Hint: When using the function `rgamma()` available in R, the arguments “shape” and “rate” correspond to α and β , respectively.
4. Estimate the approximate sampling distribution for a sample of size $n = 5$, where the individual data points are samples from the gamma distribution, defined in the previous exercise.

Chapter 11

Linear Regression Models



11.1 Introduction

Another widely used analysis method originating from statistics is linear regression [217, 222]. Since many application problems require the prediction of a numerical output variable, e.g., for forecasting stock prices, temperatures or sales, such models are often used in economics, climate science, marketing, etc. [29, 204].

Similar to classification, linear regression is also a type of a supervised learning model. However, in contrast to classification, a regression model generates a numerical variable as output, whereas a classification results in a categorical variable (which is merely a label). Although classical *ordinary least squares (OLS)* regression models have been known for a long time, they are still frequently used till today. Importantly, in recent years, there are many new developments that extend classical regression models significantly and which require a thorough understanding of the more basic models.

In this chapter, we introduce OLS linear regression models, including methods for diagnosing such models. Furthermore, we discuss extended models allowing interaction terms, nonlinearities, or categorical predictors. Finally, we introduce *generalized linear models (GLMs)*, which allow the response variable to have a distribution other than a normal distribution enabling a flexible modeling of the response.

11.1.1 What Is Linear Regression?

In a linear regression problem, one tries to find a linear function that fits the observed values of the form $\{(x_i, y_i)\}_{i=1}^n$ best. Here “best” means that we need to select a criterion that allows the quantification of the quality of a fit. It is important to emphasize that one tries to find a linear function and not a function of general

Data type:

$X = \{((x_i, y_i))\}_1^n$ with $x_i \in \mathbb{R}^p$, $y_i \in \mathbb{R}^m$

x_i is called input or predictor vector

y_i is called output or response vector

p : number of covariates or number of predictors

n : number of samples

m : number of response variables

Question addressed:

Is there a (linear) functional dependency between input and output?

Principles of major linear regression approaches:

Simple Linear Regression $\implies m = 1, p = 1$

Multiple Linear Regression $\implies m = 1, p > 1$

General Linear Regression $\implies m > 1, p > 1$

} Find coefficients β_i that
optimize an error measure

Fig. 11.1 Overview of linear regression methods with respect to the used data type, the question addressed, and the principles of major approaches

shape. This can be seen as a constraint on the class of functions we consider for this problem. At the beginning of this chapter, we will limit our discussion to linear regression problems; however, at the end we will also discuss extensions allowing nonlinear functions.

If x_i and y_i are scalar values, i.e., $x_i, y_i \in \mathbb{R}$, the regression problem is called a simple linear regression. However, when x_i is a vector of dimension p and y_i is a vector of dimension m , the problem is called a general linear regression problem, see Fig. 11.1.

11.1.2 Motivating Example

In order to get an intuitive understanding of linear regression, we present an example in Fig. 11.2. The scatterplot in this figure shows 20 data points corresponding to the number of murders per annum (per 1 million inhabitants) depending on the unemployment rate (in percentage) [281]. That means the number of murders per annum per 1 million inhabitants corresponds to the y values and the unemployment rate to the x values. Formally, the values of all pairs can be written as (x_i, y_i) where $i \in \{1, \dots, n = 20\}$; see Table 11.1.

From the visualization in Fig. 11.2, one gets the intuitive impression that the relation between y and x can be approximated by a linear function of the form

$$y \approx \beta_0 + \beta_1 x \quad (11.1)$$

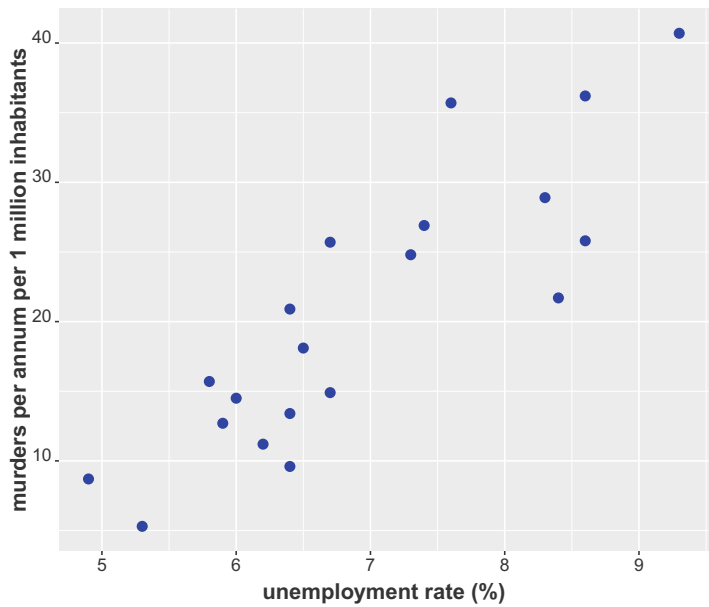


Fig. 11.2 Scatterplot of the number of murders per annum per 1 million inhabitants (y -axis) against the unemployment rate (x -axis)

Table 11.1 Data points shown in Fig. 11.2. Here, the x -values correspond to the unemployment rate (in percentage) and the y -values to the number of murders per annum per 1 million inhabitants [281]

Index	x	y
1	6.2	11.2
2	6.4	13.4
3	9.3	40.7
4	5.3	5.3
5	7.3	24.8
6	5.9	12.7
7	6.4	20.9
8	7.6	35.7
9	4.9	8.7
10	6.4	9.6
11	6.0	14.5
12	7.4	26.9
13	5.8	15.7
14	8.6	36.2
15	6.5	18.1
16	8.3	28.9
17	6.7	14.9
18	8.6	25.8
19	8.4	21.7
20	6.7	25.7

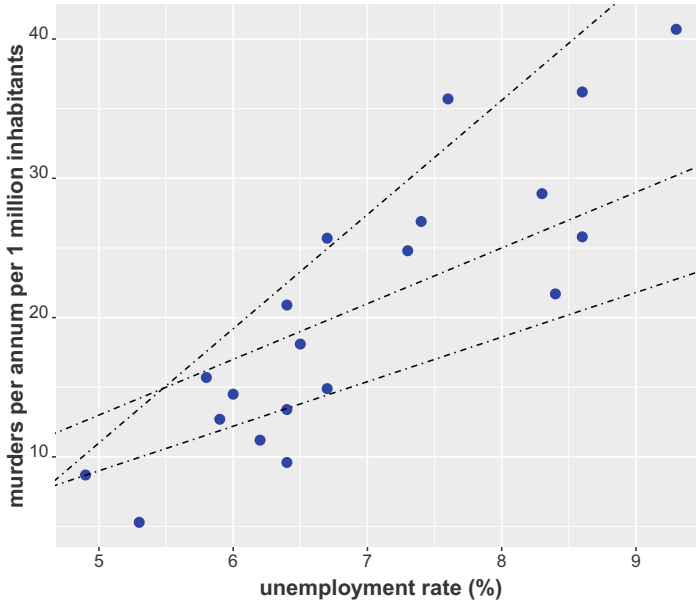


Fig. 11.3 Scatterplot of the number of murders per annum per 1 million inhabitants (y-axis) against the unemployment rate (x-axis), including three guessed linear approximation lines

where β_0 and β_1 are two unknown parameters. In Fig. 11.3, we added three specific linear functions that have different values for β_0 and β_1 . For each case, we just guessed the values of the parameters. Of course, this approach is unsatisfactory because it is arbitrary and unsystematic. For this reason, in the next section, we will discuss a systematic method that allows the estimation of the values β_0 and β_1 from our data set $\{(x_1, y_1), \dots, (x_n, y_n)\}$.

11.2 Simple Linear Regression

In this section, we will formalize the above problem by introducing a model for the linear relationship between two variables. Such a model is called *simple linear regression*. In later sections, we will extend this to more variables (see multiple linear regression in Sect. 11.4).

Suppose that we have n data points of the form $\{(x_1, y_1), \dots, (x_n, y_n)\}$. The X variable is called *covariate*, *explanatory variable*, *regressor*, *input*, or *predictor*, and the Y variable is called *response*, *explained variable*, or *output*. A simple linear regression model is defined by

$$y = \mathbb{E}[Y|X = x] = \beta_0 + \beta_1 x, \quad (11.2)$$

where β_0 and β_1 are the model parameters. Equation 11.2 defines the expectation value of Y given that X assumes the specific value x . The reason why this model is formulated for an expectation value is that both variables, i.e., Y and X , are random variables that are related via

$$Y \approx \beta_0 + \beta_1 X. \quad (11.3)$$

We used in Eq. 11.3 the approximation symbol ($\approx$) to indicate that this mapping is not exact but there is a *random error*. If we denote these errors by ε_i , we can add them to Eq. 11.3 to get the following exact equation:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \quad (11.4)$$

For the error term, we assume that $\mathbb{E}[\varepsilon_i | X = x] = 0$ and $\text{Var}(\varepsilon_i) = \sigma^2$ for all i .

11.2.1 Ordinary Least Squares Estimation of Coefficients

After formulating the problem, we can now present a solution for finding “the best” values for the parameters β_0 and β_1 in our linear regression model in Eq. 11.2 (or 11.3) using the *least squares* error measure. To define this error measure, we denote by

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i \quad (11.5)$$

a predicted value of y_i based on the estimated values of the parameters $\hat{\beta}_0$ and $\hat{\beta}_1$. Here, the symbol $\hat{\cdot}$ (called hat) indicates predicted values. We can measure the quality of this prediction by taking the difference of the true value from the predicted value given by

$$e_i = y_i - \hat{y}_i. \quad (11.6)$$

The entity e_i is the prediction error for the i th data point, and it is called the *residual*. It is clear that smaller values of e_i indicate a better prediction, and in the case of $e_i = 0$, the prediction is perfect. In a similar way, we can estimate these errors for all n data points, and as an aggregate measure, we define

$$RSS = \sum_{i=1}^n e_i^2, \quad (11.7)$$

called the *residual sum of squares (RSS)*. Due to the squares of e_i , we ensure that each term cannot be negative. Hence, the RSS is a quantitative error measure that assesses the quality of the estimated values of $\hat{\beta}_0$ and $\hat{\beta}_1$, and the best values are those which minimize RSS.

In order to find the extremum of a function, we need to calculate its first derivative and then check with the second derivation if the extremum is a minimum or a maximum [149]. As a result from this calculation (exercises), we find

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} = \frac{Cov(X, Y)}{Var(X)} \quad (11.8)$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \quad (11.9)$$

as the optimal parameter values. Here, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ and $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$ are the sample means. This is the *ordinary least squares (OLS)* solution for a simple linear regression model.

In Fig. 11.4, we show the resulting OLS regression line for our murderers data set. The result was obtained using the function `lm()` (linear model) available in R, and corresponding script is provided in Listing 11.1.

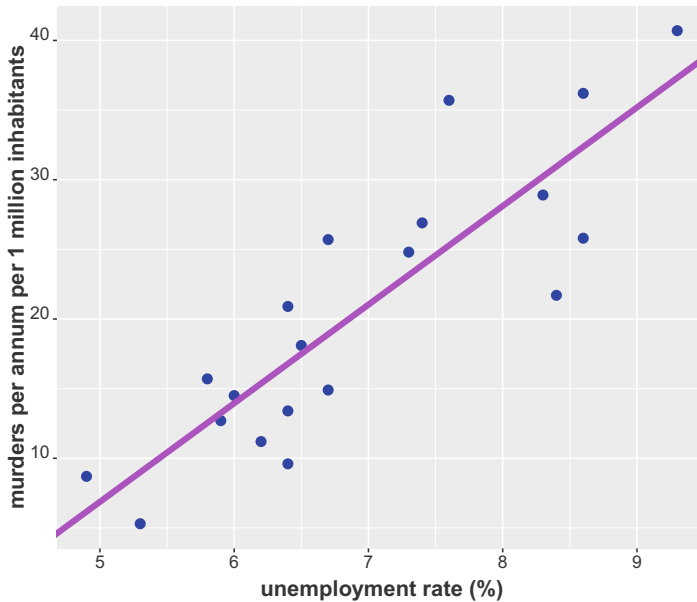


Fig. 11.4 Scatterplot of the number of murders per annum per 1 million inhabitants (y-axis) against the unemployment rate (x-axis), including the least squares solution

Listing 11.1: Simple linear regression with R

```

> res <- lm(Y ~ X)
>
> summary(res)

Call:
lm(formula = Y ~ X)

Residuals:
    Min       1Q   Median       3Q      Max
-9.2415 -3.7728  0.5795  3.2207 10.4221

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -28.5267      6.8137   -4.187 0.000554 ***
X              7.0796      0.9687    7.309 8.66e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.097 on 18 degrees of freedom
Multiple R-squared:  0.748,    Adjusted R-squared:  0.7339
F-statistic: 53.41 on 1 and 18 DF,  p-value: 8.663e-07

```

Overall, the result from the OLS regression, obtained using the function *summary()*, contains not only information about the value of the (regression) coefficients but also their standard errors, t-scores, and *p*-values (see Listing 11.1). The latter are discussed in Sects. 11.2.2 and 11.2.3. Information about the distribution of the residuals is shown at the beginning of the outputs of the function, providing some summary statistics. Furthermore, the residual standard error (Sect. 11.2.2), multiple R-squared (Sect. 11.2.4), adjusted R-squared, and the F-statistic are provided.

More information about the result from the OLS regression can be obtained using the function *confint()*, providing information about the 95% confidence intervals of the regression coefficients.

Listing 11.2: Confidence intervals for a linear model

```

> confint(res)

              2.5 %      97.5 %
(Intercept) -42.841799 -14.211619
X              5.044454   9.114655

```

Finally, using the function *coef()*, we obtain a vector of the estimated regression coefficients.

Listing 11.3: Vector of regression coefficients

```

> coef(res)
(Intercept)      X
-28.526709    7.079554

```

At this point, it should be clear why linear regression models are discussed after statistical inference (Chap. 6) and hypothesis testing (Chap. 10) because there are various concepts discussed in those chapters, e.g., confidence intervals or hypothesis tests, that are needed for regression models.

11.2.2 Variability of the Coefficients

We would like to remind that our estimation of the regression coefficients of the linear model, in Eqs. 11.8 and 11.9, is based on a data sample $\{(x_1, y_1), \dots, (x_n, y_n)\}$. That means, if we would be given another data sample of the same size and we would repeat the above procedure, we would get other estimates for the model parameters, which may be different from our first ones. For this reason, a natural question is how variable are our estimates in Eqs. 11.8 and 11.9.

To derive the following results, we need to make some further assumptions. Namely, the errors ε_i are independent from each other and

- $Var(\varepsilon_i) = \sigma^2$ for all $i \in \{1, \dots, n\}$;
- $\varepsilon_i \sim N(0, \sigma^2)$ or all $i \in \{1, \dots, n\}$.

Based on these assumptions, one can show that the standard errors of the estimated model parameters $\hat{\beta}_0$ and $\hat{\beta}_1$ are given by

$$SE(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2}, \quad (11.10)$$

$$SE(\hat{\beta}_0)^2 = \sigma^2 \left(\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right). \quad (11.11)$$

Furthermore, one can estimate the variance of the noise, i.e., $Var(\varepsilon)$, by

$$\hat{\sigma} = \frac{1}{n-2} \sum_{i=1}^n e_i^2. \quad (11.12)$$

11.2.3 Testing the Necessity of Coefficients

An important question one should always ask after estimating the model parameters is whether these parameters are necessary or can they be removed. The interesting point is that Eqs. 11.8 and 11.9 will always give some numerical values for $\hat{\beta}_0$ and $\hat{\beta}_1$ but what is not clear from these numbers is if one can remove the parameters from the model (by setting them to zero) without affecting the performance of the model. In statistical terms, we can formulate this question in the form of a hypothesis test.

Specifically, one can formulate a hypothesis test for the model parameter β_1 as follows:

Null hypothesis $H_0 : \beta_1 = 0$

Alternative hypothesis $H_1 : \beta_1 \neq 0$

Practically, one can use a one-sample t-test to perform the hypothesis test for the t-score

$$t = \frac{\hat{\beta}_1 - 0}{\text{SE}(\hat{\beta}_1)}. \quad (11.13)$$

Here, the standard error from Eq. 11.10 is used.

The resulting values are shown in the fourth and fifth column of the coefficients table, see, e.g., Listing 11.1.

11.2.4 Assessing the Quality of a Fit

In order to assess the quality of a fit, two entities are used, the *residual standard error (RSE)* and the *R² statistic*.

The *residual standard error* (RSE) assesses the standard deviation of all errors $e_i = y_i - \hat{y}_i$ made in predicting the (true) y-values. This is accomplished by normalizing RSS in the following way:

$$RSE = \sqrt{\frac{1}{n-2} \sum_{i=1}^n (y_i - \hat{y}_i)^2}; \quad (11.14)$$

$$= \sqrt{\frac{1}{n-2} RSS}. \quad (11.15)$$

The information RSE provides about the quality of a fit is relative, where high values indicate a worse and low values a better fit. For this reason, RSE measures the *lack-of-fit*. Here, “relative” means that one cannot provide absolute values of RSE that can be used as a strict guideline, but one needs to compare the RSE values of different fits with each other in a case by case basis to assess the meaning of a particular value.

The R^2 statistic, also called *coefficient of determination*, measures the proportion of explained variance by the model. It is defined by

$$R^2 = \frac{TSS - RSS}{TSS}, \quad (11.16)$$

where

$$TSS = \sum_{i=1}^n (y_i - \bar{y})^2, \quad (11.17)$$

$$RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2. \quad (11.18)$$

The interpretation of TSS is that it measures the variance in the response variables y_i relative to $\bar{y}$ and RSS measures the variance of the predicted and true response variables. Hence, TSS can be seen as a measure of the variance in the data (without considering a model fit) and RSS is the variance left after we fit a model. Overall, this means that R^2 is a measure for the explained variance in the data. The normalization by TSS ensures that R^2 can only assume values in $[0, 1]$.

For a simple linear regression, it can be shown that

$$\sqrt{R^2} = r. \quad (11.19)$$

That means, the square root of R^2 corresponds to the correlation coefficient between the predictor variable X and the response variable Y , i.e., $r = \text{Cor}(X, Y)$.

It is important to note that the function `summary()` in R (see e.g., Listing 11.1) denotes R^2 as multiple R-squared.

11.3 Preprocessing

Let's assume we have data of the form $(\mathbf{x}_i, y_i)$ with $i \in \{1, \dots, n\}$ where n is the number of samples, $\mathbf{x}_i = (X_{i1}, \dots, X_{ip})^T$ corresponds to vector of p predictors, and y_i is the response variable. We denote by $\mathbf{y} \in \mathbb{R}^n$ the vector of response variables and by $\mathbf{X} \in \mathbb{R}^{n \times p}$ the predictor matrix. The vector $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^T$ gives the regression coefficients.

The predictors and the response variable shall be standardized; that means

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^n X_{ij} = 0 \quad \text{for all } j, \quad (11.20)$$

$$\bar{s}_j^2 = \frac{1}{n} \sum_{i=1}^n X_{ij}^2 = 1 \quad \text{for all } j, \quad (11.21)$$

$$\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i = 0. \quad (11.22)$$

Here, $\bar{x}_j$ and $\bar{s}_j^2$ are the mean and variance of the predictor variables, and $\bar{y}$ is the mean of the response variables.

To study the regularization of regression models, we need to solve optimization problems, which are formulated in terms of norms. For this reason, we review, in the following, the norms needed for the subsequent sections. For a real vector $\mathbf{x} \in \mathbb{R}^n$ and $q \geq 1$, the L $_q$ -norm is defined by

$$\|\mathbf{x}\|_q = \left(\sum_{i=1}^n |x_i|^q \right)^{\frac{1}{q}}. \quad (11.23)$$

For the special case $q = 2$, one obtains the L2-norm (also known as Euclidean norm) and for $q = 1$ the L1-norm. Interestingly, for $q < 1$, Eq. 11.23 is defined, but it is no longer a norm in the mathematical sense.

11.4 Multiple Linear Regression

In this section, we extend the results for a simple linear regression model, discussed in the previous section, to a *multiple linear regression model*.

We begin our discussion by formulating a multiple regression problem:

$$Y_i = \sum_{j=1}^p X_{ij}\beta_j + \varepsilon_i \quad (11.24)$$

Here, $X_{ij} = (X_{i1}, \dots, X_{ip})^T \in \mathbb{R}^p$ are p predictor variables that are linearly mapped onto the response variable $Y_i \in \mathbb{R}$ for sample i . The mapping is defined by the p regression coefficients β_j . For the noise term, ε_i , we assume again that $\varepsilon_i \sim N(0, \sigma^2)$, which summarizes all kinds of uncertainties including measurement errors.

In order to see the similarity between a multiple linear regression, which has p predictor variables, and a simple linear regression, having one predictor variable, one can write Eq. 11.24 in the form

$$y_i = \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i \quad (11.25)$$

Here, $\mathbf{x}_i^T \boldsymbol{\beta}$ is the inner product (scalar product) between the two p -dimensional vectors $\mathbf{x}_i = (X_{i1}, \dots, X_{ip})^T$ and $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^T$. One can further summarize Eq. 11.25 for all samples $i \in \{1, \dots, n\}$ by

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \quad (11.26)$$

Here, the noise terms assumes the form $\boldsymbol{\varepsilon} \sim N(0, \sigma^2 \mathbf{I}_n)$ where $\mathbf{I}_n$ is the $\mathbb{R}^{n \times n}$ identity matrix. The matrix $\mathbf{X}$ is called the model matrix.

The solution of Eq. 11.26 can be formulated as an optimization problem given by

$$\hat{\boldsymbol{\beta}}^{OLS} = \arg \min \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2. \quad (11.27)$$

The ordinary least squares (OLS) solution of Eq. 11.27 can be analytically calculated, assuming that X has a full column rank, which implies that $X^T X$ is positive definite. The estimation of the model parameters is given by

$$\hat{\beta}^{OLS} = (X^T X)^{-1} X^T y. \quad (11.28)$$

If X has not a full column rank, the solution cannot be uniquely determined.

Listing 11.4: Multiple linear regression with R

```
> head(D)
      x1  x2  x3  y
1 587000 16.5 6.2 11.2
2 643000 20.5 6.4 13.4
3 635000 26.3 9.3 40.7
4 692000 16.5 5.3  5.3
5 1248000 19.2 7.3 24.8
6 643000 16.5 5.9 12.7

> res <- lm(data=D, y ~ x1 + x2 + x3)
> summary(res)

Call:
lm(formula = y ~ x1 + x2 + x3, data = D)

Residuals:
    Min       1Q   Median       3Q      Max
-5.7174 -3.3233  0.4031  1.7684 10.0329

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.676e+01  7.011e+00  -5.244 8.03e-05 ***
x1           7.629e-07  6.363e-07   1.199  0.24798
x2           1.192e+00  5.617e-01   2.123  0.04974 *
x3           4.720e+00  1.530e+00   3.084  0.00712 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.59 on 16 degrees of freedom
Multiple R-squared:  0.8183, Adjusted R-squared:  0.7843
F-statistic: 24.02 on 3 and 16 DF, p-value: 3.629e-06
```

11.4.1 Testing the Necessity of Coefficients

Similar to simple linear regression, where we tested the necessity of coefficients in Sect. 11.2.3, we also need to assess the coefficients for multiple linear regression models. However, instead of conducting a test for only one coefficient, for multiple linear regression, one tests the following:

null hypothesis $H_0 : \beta_1 = \beta_2 = \dots = \beta_p = 0$;
 alternative hypothesis $H_1 : \text{at least one } \beta_j \neq 0$.

As a test score, the F-statistic is used

$$F = \frac{(TSS - RSS)/p}{RSS/(n - p - 1)}. \quad (11.29)$$

The resulting value is shown in Listing 11.4.

11.4.2 Assessing the Quality of a Fit

The coefficient of determination defined in Sect. 11.2.4 is only valid for simple linear regression models. For a multiple linear regression model, this definition needs to be *adjusted*, considering the number of parameters in the model. The resulting measure is called *adjusted R^2* , and it is given by

$$R_{adj}^2 = 1 - \frac{RSS/(n - p - 1)}{TSS/(n - 1)}. \quad (11.30)$$

11.5 Diagnosing Linear Models

In order to fit linear models, we made several theoretical assumptions, e.g., about the error and the model itself. In this section, we will discuss the following topics, which are important for model diagnoses:

1. Error assumptions
 - Independence
 - Homoscedasticity (constant variance)
 - Normality
2. Linearity assumption
3. Leverage points
4. Outliers
5. Collinearity

In the following, we discuss these topics one-by-one.

11.5.1 Error Assumptions

For the errors of a regression model, we made the following assumptions:

- independence
- constant variance
- normality

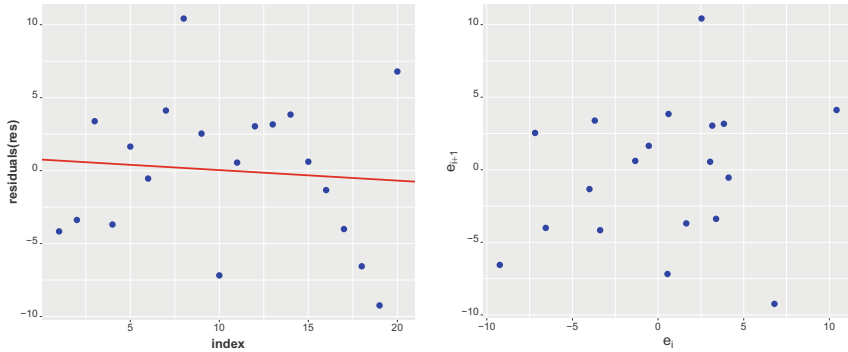


Fig. 11.5 Diagnostic plots to check independence of residuals. Left: Index plot of the residuals. Right: e_i vs e_{i+1}

Independence The first assumption we check is the independence of errors. Graphically, one can diagnose this by plotting e_i vs the index or by plotting e_i vs e_{i+1} .

Listing 11.5: Plotting e_i vs e_{i+1} , see Fig. 11.5 (right)

```
L <- length(residuals(res))
plot(residuals(res)[-L], residuals(res)[-L])
```

The left plot in Fig. 11.5 looks a bit suspicious because there are several residuals in a row, which assume increasing negative values; however, the plot for e_i vs e_{i+1} on the right-hand side looks fine. In order to confirm this quantitatively, one can use the Durbin-Watson test for e_i vs e_{i+1} . It uses the following test statistic:

$$DW = \frac{\sum_{i=1}^n (e_i - e_{i-1})^2}{\sum_{i=1}^n e_i^2}. \quad (11.31)$$

Listing 11.6: Durbin-Watson test for the correlation of e_i vs e_{i+1}

```
> library("lmtest")
> dwtest(res, alternative="two.sided")

Durbin-Watson test

data:  res
DW = 1.5113, p-value = 0.2796
alternative hypothesis: true autocorrelation is not 0
```

From the resulting p -value in Listing 11.6, one can see that the null hypothesis stating a zero correlation cannot be rejected; hence, the residuals are independent from each other.

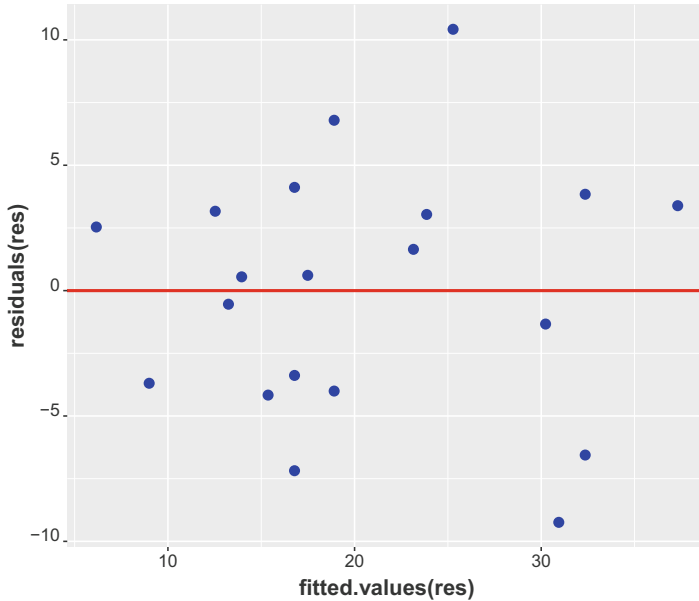


Fig. 11.6 Scatterplot showing e_i as a function of $\hat{y}_i$ (blue points). The red line is a linear regression of these values

Homoscedasticity The second assumption we check is the homoscedasticity (constant variance). For this, we plot the residuals (r_i) as a function of $\hat{y}_i$.

Listing 11.7: Detecting heteroscedasticity, see Fig. 11.6

```
res <- lm(data=D, y ~ x3)
summary(res)

plot(fitted(res), residuals(res))
```

Figure 11.6 shows the result from Listing 11.7. In addition, we added a red line corresponding to the linear regression of these values obtained using the command `lm(residuals(res), fitted.values(res))`. Homoscedasticity means that the variance does not change along the fitted values; hence, it is approximated by constant residuals. In such a case, a linear regression has a slope of zero, as shown by the results in Fig. 11.6. Any non-zero slope is an indicator of heteroscedasticity (the absence of homoscedasticity).

Normality The third assumption we check is the normality of the errors. Graphically, this can be done with a Q-Q plot.

Listing 11.8: Q-Q plot checking normality, see Fig. 11.7

```
qqnorm(residuals(res))
qqline(residuals(res))
```

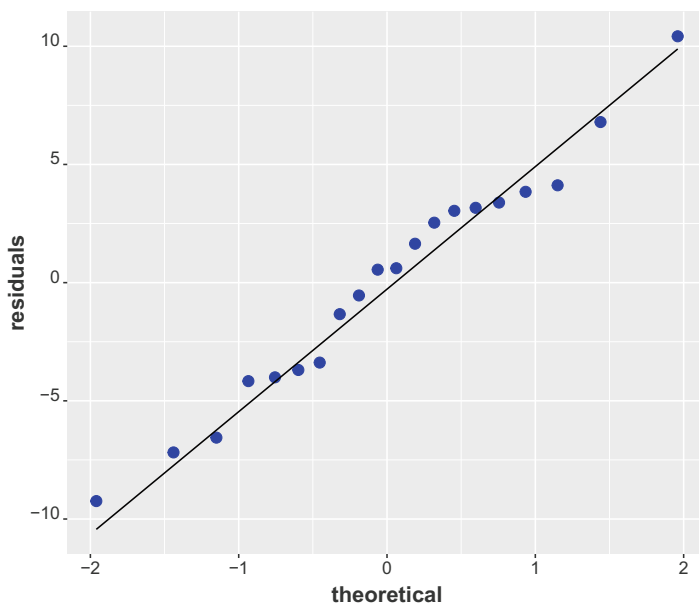



Fig. 11.7 Q-Q plot for checking the normality of the residual distribution

Listing 11.8 shows how to do this using R, and the result is displayed in Fig. 11.7. This assumption can also be checked quantitatively using the Shapiro-Wilk test. Listing 11.9 shows the result of this test.

Listing 11.9: Shapiro-Wilk normality test for Fig. 11.7

```
> shapiro.test(residuals(res))

Shapiro-Wilk normality test

data:  residuals(res)
W = 0.97814, p-value = 0.9079
```

The null hypothesis of the Shapiro-Wilk states the normality of a distribution. As one can see, the p -value indicates that the null hypothesis cannot be rejected.

11.5.2 Linearity Assumption of the Model

Since we want to fit a linear model, i.e., either a simple linear regression or a multiple linear regression model, the underlying relationship between the data of the predictors and the response variable need to be linear. For a simple linear regression

model, this can be graphically checked by plotting x_i vs y_i . However, for (high-dimensional) multiple linear regression models, this is not straightforward because only two-dimensional projections can be plotted easily. For this reason, alternatively, one can use a score quantifying the quality of a fit (see Sect. 11.4.2).

11.5.3 Leverage Points

Another entity to characterize a regression model is a leverage point. The idea behind a leverage point is to realize that not all data points exhibit the same influence on the model. In order to identify influential points, one can use the *leverage* h_i . For a simple linear regression model, one can show that

$$\text{Var}(e_i) = \sigma^2(1 - h_i), \quad (11.32)$$

$$\text{Var}(\hat{y}_i) = \sigma^2 h_i, \quad (11.33)$$

where σ^2 is the variance of the noise and the h_i are estimated by

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{i=1}^n (x_i - \bar{x})^2}. \quad (11.34)$$

From Eq. 11.34, it follows that $0 \leq h_i \leq 1$ and $\sum_{i=1}^n h_i = p$. Hence, the effect of a large h_i is to reduce the variance of a residual. Here “large” means a deviation of the average value by $2/n$, where 2 is the number of parameters of the model and n is the sample size.

We would like to note that the leverage points given by Eq. 11.34 depend only on the x_i values. Hence, they assess the influence of extremal x-points.

Listing 11.10: Leverages h_i , see Fig. 11.8

```
library("faraway")
halfnorm(influence(res)$hat)
```

From Fig. 11.8, one can see that the data points 3 and 9 have larger leverages than all the other points.

11.5.4 Outliers

In contrast with the leverage points, which focus only on the x-points, outliers are indicated by “unusual y-points”. For identifying outliers, studentized residuals, given by

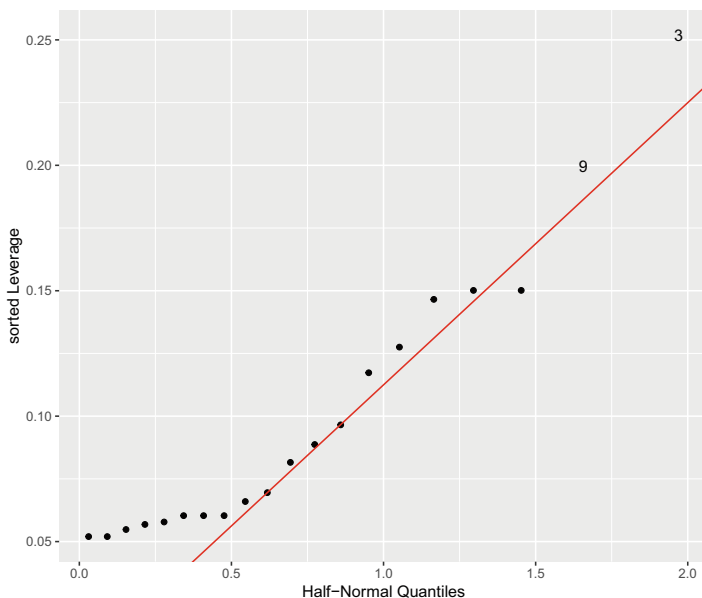


Fig. 11.8 Half-normal plot for the leverages h_i . The two labeled data points are twice as large as the average leverage value of p/n , which is here 0.1

$$\tilde{e}_i = \frac{e_i}{SE(e_i)}, \quad (11.35)$$

can be utilized. Observations with studentized residuals, $\tilde{e}_i$, larger than an absolute value of 3 are generally considered as outliers.

As a warning, we would like to remark that an unusual y-point could also indicate a deficiency of the model. Hence, one needs to be careful in judging whether a y-point is an outlier.

11.5.5 Collinearity

The collinearity between two predictor variables means that these two variables are closely related. Often, this is indicated by a high correlation coefficient. The problem is that this collinearity can lead to a reduction in the quality of the fit for the model due to a counterproductive competitive effect between these variables. A related problem is that in such a case, it is also difficult to separate the individual influences of the predictor variables on the response. More generally, this can also involve more than two predictor variables. In this case, it is called *multicollinearity*.

A simple way to check for pairwise collinearity is by looking at the correlation matrix for the predictor variables. However, this does not always reveal an existing collinearity and has even more severe limitations for multicollinearity. For this reason, the *variance inflation factor (VIF)*, defined by

$$\text{VIF}(\hat{\beta}_i) = \frac{1}{1 - R_{X_i|X_{-i}}^2}, \quad (11.36)$$

can be used [428]. Here, $R_{X_i|X_{-i}}^2$ is for a regression model of X_i on all remaining variables, indicated by X_{-i} . For a value of $R_{X_i|X_{-i}}^2$ close to one, multicollinearity is detected, and the resulting value of $\text{VIF}(\hat{\beta}_i)$ will be large.

11.5.6 Discussion

Least squares regression models can perform very badly when there are outliers in the data. For this reason, it can be very helpful to perform outlier detection on the data and remove these outliers from the data before building the regression model. A reason why least squares regression is so sensitive to outliers is that the model does not perform any form of coefficient shrinkage of the regression coefficients as, e.g., the LASSO (see Chap. 13). However, without such a restrictive mechanism, built directly into the model, outliers can cause the coefficients in the model to become very large.

Another factor, which can lead to a bad performance of a model, is the correlation between predictor variables. The disadvantage of the regression model is that it does not perform any form of variable selection to reduce the numbers of predictor variables as, e.g., ridge regression or LASSO. Instead, it uses the variables specified as input to the model.

The third factor that can reduce the performance of a model is called heteroskedasticity or heteroscedasticity. It refers to varying (i.e., non-constant) variances of the error term depending on the sampling region. One particular problem caused by heteroskedasticity is the obtention of inefficient and biased estimates of the OLS standard errors, which can result in biased statistical tests of the regression coefficients [274].

In summary, ordinary least squares regression performs neither a shrinkage nor a variable selection, potentially leading to the aforementioned problems. For this reason, advanced regression models have been introduced to guard against such problems. We will discuss such extensions in Chap. 13, where we introduce *regularization* methods.

11.6 Advanced Topics

In the next section, we address some advanced topics of multiple linear regression that further extend the framework toward more complex models. Specifically, we will discuss interactions, nonlinearities, and categorical predictors.

11.6.1 Interactions

The first extension of multiple linear regression we discuss includes interaction terms in the regression model. In general, an interaction is a multiplicative effect of two (or more) predictors on the response variable. An example for such an interaction term is given in the following model:

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \varepsilon_i. \quad (11.37)$$

The term $x_1 x_2$ introduces a nonlinearity in the model, which means that the model is no longer linear in x_1 and x_2 .

However, by introducing the new auxiliary variable $x_3 = x_1 x_2$, one can create a new model, which is linear in x_1 , x_2 and x_3 . In $\mathbb{R}$, such a model can be defined via $x_1 * x_2$. This creates the exact same model as in Eq. 11.37. In Listing 11.11, we present an example using simulated data.

If one just wants to include the interaction term but not the individual predictors, one needs to use the syntax ' $x_1 : x_2$ '. This creates a model of the form

$$y_i = \beta_0 + \beta_1 x_1 x_2 + \varepsilon_i. \quad (11.38)$$

which is obviously completely different from the model in Eq. 11.37

11.6.2 Nonlinearities

The next extension allows nonlinear terms, e.g., polynomials, in a model. For instance, a model of the form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^3 + \varepsilon. \quad (11.39)$$

is nonlinear in x_1 . However, by defining a new variable $x_2 = x_1^3$, the above model becomes

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon, \quad (11.40)$$

Listing 11.11: Interactions in multiple linear regression

```
# Simulated data
set.seed(123456) # Set seed
n <- 20 # Number of observations
p <- 2 # Number of predictors included in model
x1 <- rnorm(n)
x2 <- rnorm(n)
xI <- x1*x2 # interaction term
x <- cbind(x1, x2, xI)

beta <- c(1, 2, 3) # true model coefficients
y <- x %*% beta + rnorm(n)

# Regression with interaction
res <- lm(y ~ x1*x2)
> summary(res)

Call:
lm(formula = y ~ x1 * x2)

Residuals:
    Min       1Q   Median       3Q      Max
-1.20037 -0.53684  0.04399  0.45287  1.47095

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.4333      0.2224  -1.948   0.0692 .
x1             1.4041      0.2037   6.893 3.62e-06 ***
x2             2.0427      0.2411   8.471 2.62e-07 ***
x1:x2          2.5902      0.2329  11.121 6.15e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.873 on 16 degrees of freedom
Multiple R-squared:  0.9757, Adjusted R-squared:  0.9711
F-statistic: 214.1 on 3 and 16 DF, p-value: 4.023e-13
```

which is linear in x_1 and x_2 .

Listing 11.12 presents an example how a polynomial nonlinearity is analyzed using R. In order to indicate that a covariate should be used as a polynomial, the argument $I()$ needs to be used for the function $lm()$.

It is important to note that this works not only for polynomials but also for general nonlinear transformations, e.g., using $I(\log(x))$ makes this trick applicable to deal with general nonlinearities.

In Fig. 11.9, we show a visualization of the results in Listing 11.12. Here, the purple line corresponds to the fitted coefficients of the cubic model in Eq. 11.39. For comparison purpose, we repeated a similar analysis for a quadratic model given by

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2. \quad (11.41)$$

The result for this model is shown by the red line in Fig. 11.9.

Listing 11.12: Nonlinearities in multiple linear regression

```
# Simulating data
set.seed(123456) # Set seed for reproducibility
n <- 20 # Number of observations
p <- 2 # Number of predictors included in model
x1 <- rnorm(n)
x <- cbind(x1, x1^3)

beta <- c(1, 2)
y <- x %*% beta + rnorm(n)

# Regression with nonlinearities
res <- lm(y ~ x1 + I(x1^3))
> summary(res)

Call:
lm(formula = y ~ x1 + I(x1^3))

Residuals:
    Min       1Q   Median       3Q      Max
-2.3628 -0.6867 -0.0414  0.5949  1.5093

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.1677     0.2676  -0.627   0.539
x1             0.6311     0.3894   1.621   0.123
I(x1^3)       2.1326     0.0939  22.711 3.71e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.058 on 17 degrees of freedom
Multiple R-squared:  0.9892, Adjusted R-squared:  0.9879
F-statistic:  779 on 2 and 17 DF, p-value: < 2.2e-16
```

11.6.3 Categorical Predictors

Linear regression models can also handle categorical predictors. A categorical predictor does not assume numerical values, e.g., real numbers, but categories, which are also called levels. The categorical predictor itself is called a factor.

Generally, one categorical variable with n levels can be substituted by $n - 1$ dummy variables (or indicators) each with two levels. These $n - 1$ new dummy variables contain the same information as the single variable.

As an example, let's use the 'mtcars' data set to perform a regression of the variable "mpg" (miles per gallon) on the variables weight ("wt") of a car and its number of cylinders ("cyl"). Here, we consider "wt" as a numerical predictor and "cyl" as factor. For this example, "cyl" has three levels: 4 cylinder, 6 cylinder, and 8 cylinder. Hence, we need two indicator variables x_{I1} and x_{I2} to code the factor "cyl." That means the regression model will assume the following form:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_{I1} + \beta_3 x_{I2} + \varepsilon_i. \quad (11.42)$$

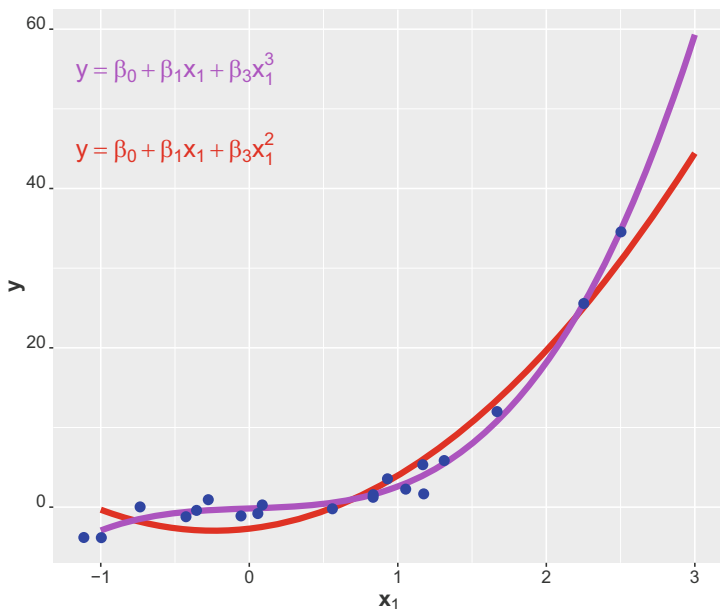


Fig. 11.9 The figure shows results of two nonlinear regression models. The purple line corresponds to a cubic model and the red line to a quadratic model

Here, $x_1 \in \mathbb{R}$ (as usual) but $x_{I_1}, x_{I_2} \in [0, 1]$. In total, this allows to code three different configurations given by

$$y_i = \beta_0 + \beta_1 x_1 + \varepsilon_i, \quad \text{for } x_{I_1} = 0, x_{I_2} = 0 \quad (11.43)$$

$$y_i = (\beta_0 + \beta_2) + \beta_1 x_1 + \varepsilon_i, \quad \text{for } x_{I_1} = 1, x_{I_2} = 0 \quad (11.44)$$

$$y_i = (\beta_0 + \beta_3) + \beta_1 x_1 + \varepsilon_i, \quad \text{for } x_{I_1} = 0, x_{I_2} = 1. \quad (11.45)$$

As one can see, all three models have the same slope but a different intercept due to the influence of the different levels of the factorial variable. Importantly, an indicator variable can only assume one level because both indicators code together for the number of cylinders (“cyl”), which is in our case a factor. For this reason, e.g., $x_{I_1} = 1, x_{I_2} = 1$ is not possible (see below the discussion using the function *contrasts()*). The numerical result of this model is shown in Listing 11.13.

To clarify the meaning of a factor, one can use the function *contrasts()* available in R. The output of this function is a table showing the coding of two indicator variables (columns) for the number of cylinders (rows). This means that “4 cylinders” (cyl4) is the reference point because it corresponds to the absence of the indicator variables, i.e., $x_{I_1} = 0, x_{I_2} = 0$. It also shows that cyl6 and cyl8, corresponding to 6 and 8 cylinder, are coded using $x_{I_1} = 1, x_{I_2} = 0$ and $x_{I_1} = 0, x_{I_2} = 1$, respectively. In summary, this gives the following meaning for the

Listing 11.13: A factor in a multiple linear regression

```

data("mtcars")
y <- mtcars$mpg
x1 <- mtcars$wt
x2 <- as.factor(mtcars$cyl)
dat <- data.frame(y = y, wt = x1, cyl = x2)

res <- lm(y ~ wt + cyl, data = dat)
summary(res)
# output
lm(formula = y ~ wt + cyl, data = dat)

Residuals:
    Min       1Q   Median       3Q      Max
-4.5890 -1.2357 -0.5159  1.3845  5.7915

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   33.9908     1.8878   18.006 < 2e-16 ***
wt            -3.2056     0.7539   -4.252 0.000213 ***
cyl6          -4.2556     1.3861   -3.070 0.004718 **
cyl8          -6.0709     1.6523   -3.674 0.000999 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.557 on 28 degrees of freedom
Multiple R-squared:  0.8374, Adjusted R-squared:  0.82
F-statistic: 48.08 on 3 and 28 DF, p-value: 3.594e-11

contrasts(dat$cyl)
# output
      6 8
4 0 0
6 1 0
8 0 1

```

different models:

$$y_i = \beta_0 + \beta_1 x_1 + \varepsilon_i, \quad \text{for 4 cylinder} \quad (11.46)$$

$$y_i = (\beta_0 + \beta_2) + \beta_1 x_1 + \varepsilon_i, \quad \text{for 6 cylinder} \quad (11.47)$$

$$y_i = (\beta_0 + \beta_3) + \beta_1 x_1 + \varepsilon_i, \quad \text{for 8 cylinder.} \quad (11.48)$$

Hence, the results for 6 and 8 cylinder are shown with reference to 4 cylinder (see the change in the intercept in Eqs. 11.47 and 11.48).

If one wants to use a different reference, one needs to indicate this explicitly. For instance, including the command `dat$cyl <- relevel(dat$cyl, ref="6")` will use 6 cylinders as the reference. It is worth noting that in this case, the remaining indicators are arranged automatically. To illustrate this, an example is shown in Listing 11.14.

Listing 11.14: Set a new baseline category

```

dat$cyl <- relevel(dat$cyl, ref = "6")
res <- lm(y ~ wt + cyl, data = dat)
summary(res)
# output
Call:
lm(formula = y ~ wt + cyl, data = dat)

Residuals:
    Min       1Q   Median       3Q      Max
-4.5890 -1.2357 -0.5159  1.3845  5.7915

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   29.7352     2.5410   11.702 2.69e-12 ***
wt            -3.2056     0.7539   -4.252 0.000213 ***
cyl4           4.2556     1.3861    3.070 0.004718 **
cyl8          -1.8153     1.3576   -1.337 0.191956
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.557 on 28 degrees of freedom
Multiple R-squared:  0.8374, Adjusted R-squared:  0.82
F-statistic: 48.08 on 3 and 28 DF, p-value: 3.594e-11

contrasts(dat$cyl)
# output
  4 8
6 0 0
4 1 0
8 0 1

```

If one does not want an automatic arrangement, one can also specify the order of all levels. This can be done using the command `dat$cyl <- factor(dat$cyl, levels = c("8", "6", "4"))` (i.e., we change the first line in Listing 11.14).

11.6.4 Generalized Linear Models

The last extension we present, in this chapter, is a model family called *generalized linear models* (GLM). GLMs were popularized by McCullagh and Nelder in the 1980s.

In contrast with the models discussed so far, a GLM is different (more general) in two aspects. First, a GLM allows the response variable to have a distribution other than the normal distribution, and, second, we can model the connection between predictors and (the mean of) response by a link function. For the distribution, the only restriction is that the distribution of the response needs to be a representative of the exponential family. This family class of probability distributions includes normal, binomial, Poisson, Dirichlet, or gamma distributions; hence, a such model family is very flexible.

Table 11.2 An overview of common link functions, $\eta = g(\mu)$, and their corresponding mean functions, $\mu = g^{-1}(\eta)$

Link function name	Link function equation: $\eta = g(\mu)$	Mean function equation: $\mu = g^{-1}(\eta)$	Mean function name
Identity	μ	η	Identity
Inverse	μ^{-1}	η^{-1}	Inverse
Inverse-square	μ^{-2}	$\eta^{-1/2}$	Inverse-square-root
Log	$\ln \mu$	$\exp \eta$	Exponential
Logit	$\ln \frac{\mu}{1-\mu}$	$\frac{1}{1+\exp(-\eta)}$	Logistic
Probit	$\Phi^{-1}(\mu)$	$\Phi(\eta)$	Normal quantile

A GLM is defined by the following three components:

1. A *conditional probability distribution*, $f()$, for the response variable Y from the exponential family, $y \sim f(y|\mu)$.
2. A *linear predictor*, η , given by $\eta = \beta_0 + \beta_1 x$.
3. A *link function*, g , given by $\mu = g^{-1}(\eta) = \mathbb{E}[Y|X]$.

The linear predictor includes the independent variables X in the model. Above we formulated a GLM for one covariate for simplicity; however, an extension to the multivariate case is straightforward by defining

$$\eta = \beta_0 + \sum_{i=1}^p \beta_i x_i. \quad (11.49)$$

The link function models the relationship between the linear predictor and the mean of the distribution function given by $\eta = g(\mu)$. The definition of the link function assumed that g is a differentiable function for which an inverse exists. The inverse g^{-1} is called mean function because it gives the mean of the response variable, i.e., $\mu = g^{-1}(\eta)$.

In Table 11.2, we show some examples of common link functions and the corresponding mean functions. Specifically, the first two columns show the name of link functions and their equations, i.e., $\eta = g(\mu)$, and the remaining column show the corresponding mean functions, $\mu = g^{-1}(\eta)$, and their common names.

For the probit function, $\Phi()$ is the cumulative distribution function of the standard-normal distribution.

At this point the importance of the link function can be discussed in more detail. Let's consider two link functions: Identity and Inverse. The corresponding linear predictors are given by

$$\eta = \mu = \beta_0 + \beta_1 x = \mathbb{E}[Y|X]; \quad (11.50)$$

$$\eta = \mu^{-1} = \beta_0 + \beta_1 x = \mathbb{E}[Y|X]^{-1}. \quad (11.51)$$

The linear predictor in Eq. 11.50 corresponds to a linear regression model, whereas Eq. 11.51 requires a transformation (the inverse) to obtain

$$\mathbb{E}[Y|X] = (\beta_0 + \beta_1 x)^{-1}. \quad (11.52)$$

Hence, from Eq. 11.52, it is clear that the expectation value of the response variable is nonlinear in the predictor variable. As one can see, using different link functions, one can obtain different forms of nonlinearity to model the relationship between predictor and response variables.

When specifying a GLM, the link function needs to suit the probability distribution of the response variable. This means that it cannot be arbitrarily chosen. Information about possible link functions can be obtained via the help function in R for the function *family()*. From this, one finds, e.g., the “Gaussian” (aka normal) family accepts the link functions “Identity,” “Log,” and “Inverse,” and the “Poisson” family accept the links “Log,” “Identity,” and “Square-root.” This information can be used to fit a GLM via the function *glm*.

From these examples, one can see that the link function is not unique to a family, but multiple families can have the same link function. For instance, “Log” can be used as a link function for a Gaussian and a Poisson family. Overall, R provides information for six families (Gaussian, Binomial, Poisson, Quasi, Gamma, and Inverse-Gaussian) each with a number of link functions.

After specifying the probability family, the linear predictor, and the link function, the parameters of the model $\{\beta_i\}_{i=0}^p$ need to be estimated. For this optimization, the function *glm* uses the maximum likelihood estimation.

11.6.4.1 How to Determine Which Family to Use When Fitting a GLM?

An important questions is how to decide which family to use when fitting a GLM. This depends on the type of the response variable. Specifically, one can select a family based on the following distinctions:

- Continuous and unbounded response: Gaussian distribution.
- Continuous and non-negative response: Gamma distribution or Inverse-Gaussian distribution.
- Count response: Poisson distribution, quasi-Poisson distribution, or negative binomial distribution.
- Binary response: Binomial distribution.
- Multi-category response: Multinomial distribution.

11.6.4.2 Advantages of GLMs Over Traditional OLS Regression

Some advantages of GLMs over traditional (OLS) regression include:

- The choice of link function is separate from the choice of the probability of the response variable. This gives more flexibility for the modeling.
- If the link produces additive effects, then we do not need constant variance.
- The coefficients of the model are fitted via maximum likelihood estimation.
- R provides the function *glm*, which is very flexible in specifying the three components needed to define a GLM. This allows an easy comparison of different GLMs.

11.6.4.3 Example: Poisson Regression

In the following, we discuss a problem involving count data. For such data the response variable assumes only discrete values, e.g., $y \in \{0, 1, \dots, m\}$. Importantly, the number of values is larger than two, i.e., $m > 1$; otherwise we would have binary data. A probability distribution for count data is the Poisson distribution. Hence, in the following, we establish a GLM for a Poisson regression.

We use data from [315] on the number of publications produced by Ph.D. biochemists to illustrate the application of a Poisson regression. The data can be loaded from the Stata website as shown in Listing 11.15.

Listing 11.15: Importing data from Stata

```
library(foreign)
dat <- read.dta("http://www.stata-press.com/data/lf2/couart2.dta")
```

In Fig. 11.10, we show the number of publications by biochemists produced in the last 3 years of Ph.D. This corresponds to the response variable of our model.

In Listing 11.16, we show a Poisson regression with Log as link function for the covariate “ment” corresponding to the articles published by the mentor in the last 3 years.

The coefficients in Listing 11.16 can be accessed using the command *exp(coef(res.poisson))*.

In the output of Listing 11.16, one can see information about the deviance of the model. In general, deviance is a measure of goodness of fit of a generalized linear model, and higher values indicate a worse fit.

There are two versions of the deviance provided in Listing 11.16: *Null deviance* and *Residual deviance*. The Null deviance shows how well the response variable is predicted by a model that includes only the intercept but no other covariates. In contrast, the Residual deviance is for the full model as specified by the function *glm()*. For this reason, the Null deviance is always larger than the Residual deviance when the data are not explained by a model containing only an intercept. In Listing 11.16, it is indicated that “Dispersion parameter for poisson family taken to be 1” because this is the assumption of the Poisson model. However, as a rule of thumb, the ratio of the Residual deviance and the degrees of freedom (df) gives an estimate for the actual dispersion:

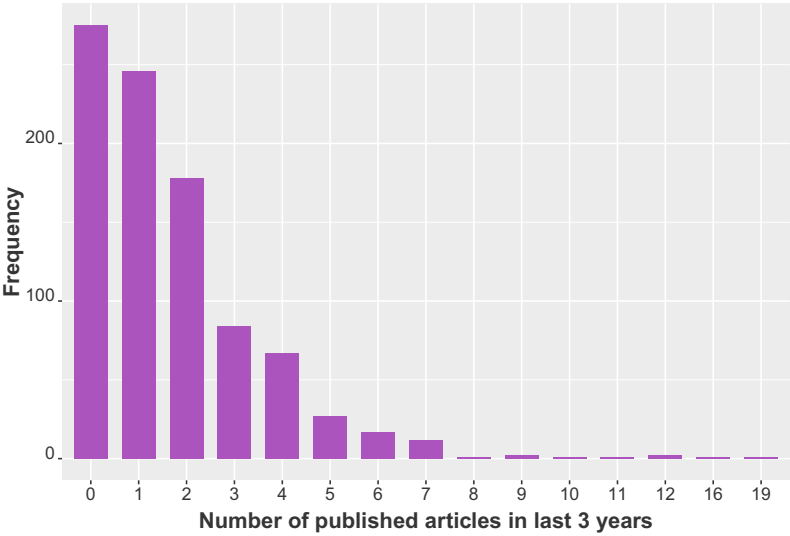


Fig. 11.10 The figure shows the number of published articles in the last 3 years of Ph.D

Listing 11.16: Poisson regression with Log as link function

```
res.poisson <- glm(art~ment, family=poisson(link="log"), data=dat)
summary(res.poisson)
# output
Call:
glm(formula = art ~ ment, family = poisson(link = "log"), data =
    dat)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.5700  -1.6316  -0.3598   0.5068   5.9483

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.259906    0.034361   7.564 3.91e-14 ***
ment         0.026050    0.001917  13.586 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 1817.4  on 914  degrees of freedom
Residual deviance: 1669.5  on 913  degrees of freedom
AIC: 3341.3

Number of Fisher Scoring iterations: 5
```

$$\hat{\phi} = \frac{\text{Residual deviance}}{\text{df}}. \tag{11.53}$$

If the fit is good $\hat{\phi} \approx 1$. From the value in Listing 11.16, we find that $\hat{\phi} = 1.82$ which means that there is about 80% overdispersion. We will continue this discussion at the end of this section.

Information about “Fisher Scoring” refers to the optimization method used to estimate the coefficients. Since the maximum likelihood estimation cannot be performed analytically, numerical procedures need to be used, e.g., the Newton-Raphson method. The Fisher Scoring improves upon the Newton-Raphson method by replacing the Hessian matrix with its expected value which is the (negative) Fisher information.

The meaning of AIC (Akaike information criterion) will be discussed in Sect. 12.5.

Using the coefficients in Listing 11.16, one can connect to the Poisson distribution, which is given by

$$P(Y = y|\lambda) = \frac{\exp(-\lambda)\lambda^y}{y!}, \quad (11.54)$$

and characterized by the single parameter λ .

For a Poisson distribution, the mean and variance are given by

$$\mathbb{E}[Y|x] = \lambda, \quad (11.55)$$

$$\text{Var}[Y|x] = \lambda. \quad (11.56)$$

Using the connection between the link function and the linear predictor, one obtains

$$\ln(\lambda) = \beta_0 + \beta_1 x. \quad (11.57)$$

Here, we just used $\mu = \lambda$ since the parameter for a Poisson distribution is commonly denoted λ . From Eq. 11.57, it follows that

$$\lambda = \exp(\beta_0 + \beta_1 x). \quad (11.58)$$

Overall, this results in the Poisson distribution for our GLM given by

$$P(Y = y|x, \beta_0, \beta_1) = \frac{\exp(-\exp(\beta_0 + \beta_1 x))\exp(\beta_0 + \beta_1 x)^y}{y!}. \quad (11.59)$$

Here, (x, y) is an observation point. It is important to note that Eq. 11.59 provides the probability for different responses given by y . However, the predictions of the GLM are with respect to the expectation value, i.e.,

$$\mathbb{E}[Y|x] = \exp(\beta_0 + \beta_1 x). \quad (11.60)$$

Equation 11.60 is also important in interpreting the regression coefficients. In order to explore this, let's rewrite Eq. 11.60 as

$$\mathbb{E}[Y|x] = \exp(\beta_0)\exp(\beta_1x). \quad (11.61)$$

From this, it follows that:

- $\beta_1 > 0$: Positive association with x ;
- $\beta_1 < 0$: Negative association with x ;
- $\beta_1 = 0$: No association with x .

Furthermore, for $x = 0$ we obtain, from Eq. 11.61, that

$$\mathbb{E}[Y|x = 0] = \exp(\beta_0). \quad (11.62)$$

In order to interpret the change in the response, going from x to $x + 1$, i.e., changing the value of x by one unit, one needs to assess the following expressions:

$$\mathbb{E}[Y|x] = \exp(\beta_0)\exp(\beta_1x); \quad (11.63)$$

$$\mathbb{E}[Y|x + 1] = \exp(\beta_0)\exp(\beta_1(x + 1)) = \exp(\beta_0)\exp(\beta_1x)\exp(\beta_1). \quad (11.64)$$

Combining the above two equations leads to

$$\mathbb{E}[Y|x + 1] = \mathbb{E}[Y|x]\exp(\beta_1). \quad (11.65)$$

Hence, the change of the response for one unit change in x is given by the factor $\exp(\beta_1)$. This implies that the exponent of the regression coefficient has an intuitive interpretation for the Poisson model but not the regression coefficient itself. Let's use this model to make some predictions.

Listing 11.17: Predictions on new data for the fitted Poisson model

```
new.dat <- data.frame(ment=seq(0, 80, by=1))
pred.new.dat <- predict(res.poisson, newdata=new.dat,
  type="response")
```

The results of these predictions given by Listing 11.17, together with the data are shown in Fig. 11.11.

To assess the goodness of fit, we use the deviance (see Listing 11.16). The deviance is a measure of how well the model fits the data. Since the deviance can be derived as the profile likelihood ratio test comparing the current model to the saturated model, likelihood theory would predict that (assuming the model is correctly specified) the deviance follows a chi-squared distribution, with degrees of freedom equal to the difference in the number of parameters.

Using R, the p -value for the deviance goodness of fit test is estimated as shown in Listing 11.18.

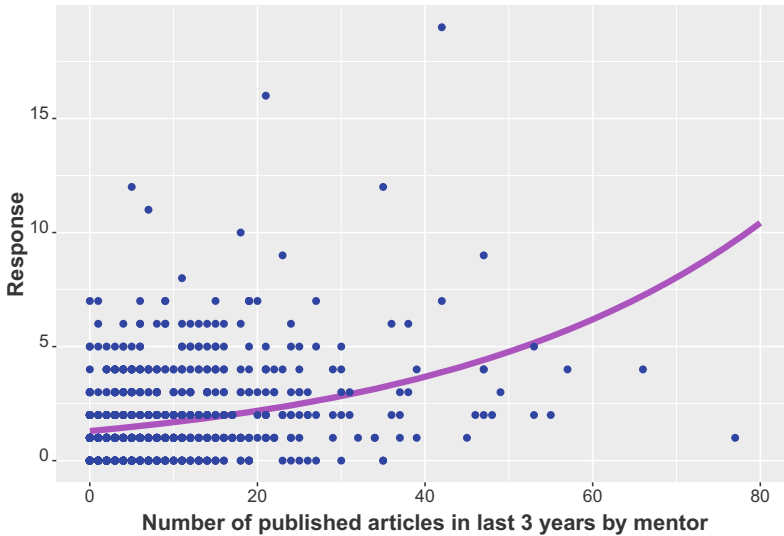


Fig. 11.11 Results for predictions of the fitted Poisson model (purple line). The blue points correspond to the data

Listing 11.18: Deviance goodness of fit test

```
pchisq(res.poisson$deviance, df=res.poisson$df.residual,
       lower.tail=FALSE)
[1] 5.36703e-47
```

The null hypothesis for this test is: “our model is correctly specified.” For a significance level of $\alpha = 0.05$, we have strong evidence to reject the null hypothesis because the p -value is much smaller than α . In general, a high p -value indicates no evidence of lack-of-fit, and a low p -value indicates evidence of lack-of-fit.

After having identified a problem with the quality of the fit (see Listing 11.18), the next step is to find the reason behind this. One problem could be dispersion. In general, dispersion quantifies how much higher (or lower) the observed variance is, relative to the expected variance of the model. For a Poisson distribution, the expected value is equal to the variance; hence, the dispersion should be 1. In other words, when we fit a Poisson model, we expect the variance to increase with the mean value by a factor of 1. If the variance decreases at a lower rate, then the data are underdispersed, and if the variance decreases at a higher rate, then the data are overdispersed. This is tested in Listing 11.19.

The hypothesis of the overdispersion test can be formulated as follows:

Null hypothesis: “The true dispersion is not greater than 1”.

Alternative hypothesis: “The true dispersion is greater than 1”.

Given the very low p -value of $9.289e - 09$, we need to reject the null hypothesis. This means that we found a reason for the bad quality of the fit.

Listing 11.19: Overdispersion test

```

library(AER)
dispersiontest(res.poisson)
# output
Overdispersion test

data:  res.poisson
z = 5.6247, p-value = 9.289e-09
alternative hypothesis: true dispersion is greater than 1
sample estimates:
dispersion
1.889713

```

The above problem can be placed in a wider context. Instead of thinking of testing the quality of a model, this can be seen as testing several models among which we try to find the best one. This setting is formalized by model selection. For this reason, we stop our discussion here and continue in Sect. 12.5, where we introduce formally model selection for GLMs.

11.6.4.4 Example: Logistic Regression

The next GLM we discuss is the logistic regression. This model is for binary data, i.e., the response can only assume two values, e.g., “Yes” or “No.” The model assumes that the response is sampled from a binomial distribution:

$$Y|x \sim \text{Binom}(p(x)). \quad (11.66)$$

The linear predictor is given by

$$\eta = \beta_0 + \beta_1 x. \quad (11.67)$$

The link function is given by the logit function

$$g(p(x)) = \text{logit}(p(x)) = \eta \quad (11.68)$$

with

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right). \quad (11.69)$$

In Sect. 9.6, we showed that Eq. 11.68 can be solved for $p(x)$, giving

$$p(x) = \frac{\exp(\beta_0 + \beta_1 x)}{1 + \exp(\beta_0 + \beta_1 x)} = \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x))}, \quad (11.70)$$

which is the parameter of the binomial distribution (see Eq. 11.66).

In Listing 11.20, we present an example to illustrate how to specify a logistic regression model using R.

Listing 11.20: Logistic regression with Logit as link function

```
res.logistic <- glm(InMichelin ~ Food, data=df,  
  family=binomial(link=logit))
```

A numerical example of a logistic regression model was already presented and discussed in Sect. 9.6 because the model performs a classification.

The logistic regression model is a good example to show that regression and classification are not that different. In fact, it depends on the type of output variable, and for a binomial distribution, a GLM can serve as a classifier.

11.7 Summary

In this chapter, we introduced OLS regression models and GLMs. Both frameworks provide linear regression models. Despite the fact that the underlying idea of an OLS regression model is easy to understand, we have seen that the diagnostic of such models is not that simple, and it requires attention because the analyst needs to make decisions. Furthermore, advanced topics of OLS regression, e.g., interaction terms, nonlinearities, or categorical predictors, make the models quickly more involved and non-trivial. This is even more true for GLMs, providing an elegant extension of OLS regression models by allowing the response variable to have an error distribution other than a normal distribution. Also, the link function permits to model different forms of nonlinearity.

Learning Outcome 11: Linear Regression Models

Linear regression models are supervised learning approaches that require labeled data for the training. The labels provide numerical information of the output variable(s). Linear regression models are a family of methods that comprise many different regression models, including OLS and GLM.

Finally, we would like to highlight that GLMs have an elegant underlying framework, allowing to derive a multitude of different models with different statistical properties. This observation is worth noting because it is different from all the methods we discussed so far, in this book.

11.8 Exercises

1. For the data in Listing 11.12, estimate a nonlinear regression model. Specifically, fit a quadratic regression model given by

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2. \quad (11.71)$$

Compare the results with the cubic regression model as discussed in Sect. 11.6.2.

2. Repeat the analysis but with more terms in the regression model. Specifically, fit the model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3. \quad (11.72)$$

Discuss the differences between the quadratic model in Eq. 11.71 and the cubic model in Eq. 11.39.

3. Repeat the analysis but using a polynomial regression model of 6-th order given by

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3 + \beta_4 x_1^4 + \beta_5 x_1^5 + \beta_6 x_1^6. \quad (11.73)$$

Is there an advantage or disadvantage to go to higher orders? A formal approach to the problem is provided in Chap. 12 because this problem is related to model selection.

4. Repeat the analysis shown in Listing 11.14 for the categorical variable “cyl.” But this time, use 8 cylinder as the reference, and order the remaining levels in the order 6 cylinder and 4 cylinder. How does one check that the order of the levels is correct?

Chapter 12

Model Selection



12.1 Introduction

In this chapter, we discuss approaches for a problem called model selection. Model selection is always needed when we have a number of candidate models that could be used for a prediction task, but we want to choose the best one among them. For instance, for a classification problem, we may consider an SVM or a decision tree. Similarly, for a regression analysis, there may be different options for the number of predictors of the model. In either case, one needs to decide which statistical model to select from the available candidates. This is the topic of model selection.

There is a related topic to model selection called model assessment. Model selection and model assessment are frequently confused despite the fact that both topics focus on different goals. For this reason, we start our discussion about model selection by clarifying the difference to model assessment.

12.2 Difference Between Model Selection and Model Assessment

If we would live in an ideal world with infinitely large data sets, the problem of model selection would be an easy one. For each candidate model, one could estimate the test error for infinitely large test data. Based on this, one could select the model with the smallest error. Importantly, in this case, the ideal test error would correspond to the so-called *expected generalization error* [42, 363, 488], providing the most complete information about the generalization abilities of a prediction model. We discuss the important (but difficult) expected generalization error formally in Chap. 18.

Overall, this means that in an ideal world with infinitely large data sets, the expected generalization error could be used to assess and select models. However,

we do not live in such an ideal world. For this reason, the finite size of the data enforces a trade-off between model selection and model assessment, which makes both tasks non-trivial.

In reality, model selection and model assessment aim to deal with finite data, requiring us to be more economic about the data usage. While model selection has its focus on selecting one model, model assessment aims to approximate the expected generalization error. The underlying idea of model selection is to estimate an auxiliary function that is different to the expected generalization error but suffices to rank different models, in a similar way, as we would have done using the expected generalization error if known. This means that the measure used for model selection just needs to result in the same ordering of the models as if the generalization errors of the models would have been used for the ranking. Hence, model selection is actually a model ordering problem, and the best model is selected without necessarily estimating the expected generalization error. This explains why model assessment and model selection are in general two different approaches.

Briefly, we can state the goals of model selection and model assessment as follows:

Model selection: Estimate a performance criterion of different models in order to choose the best model.

Model assessment: For the best model resulting from model selection, estimate its expected generalization error.

In the following, we outline the underlying idea of model selection in more detail.

12.3 General Approach to Model Selection

There are two schools of thought for model selection, and they differ in the way how the “best model” is defined. The first defines best model as the “best prediction model” and the second as the “true model” that generated the data [108, 168, 169]. For this reason, the latter is also referred to as *model identification*. The first definition fits seamlessly to our above discussion about the expected generalization error. In contrast, the second one is based on the assumption that the true model has also the best expected generalization error. For very large sample sizes, i.e., $n_{\text{train}} \rightarrow \infty$, this is uncontroversial. However, for finite sample sizes, as is always the case in practice, this is not necessarily the case.

In Fig. 12.1, we visualize the theoretical problem of model selection. In Fig. 12.1a, we show three model families, g_1 , g_2 , and g_3 , indicated by the three curves colored in blue, red, and green. Each of these model families corresponds to a statistical model, e.g., a linear regression model with p_1 , p_2 , and p_3 covariates. Each point, along these curves, corresponds to a particular model obtained by estimating the parameters of the models using a training data set D . Here, $g_1(x, \hat{\beta}_1(D))$, $g_2(x, \hat{\beta}_2(D))$, and $g_3(x, \hat{\beta}_3(D))$ are three such models that have been specified by

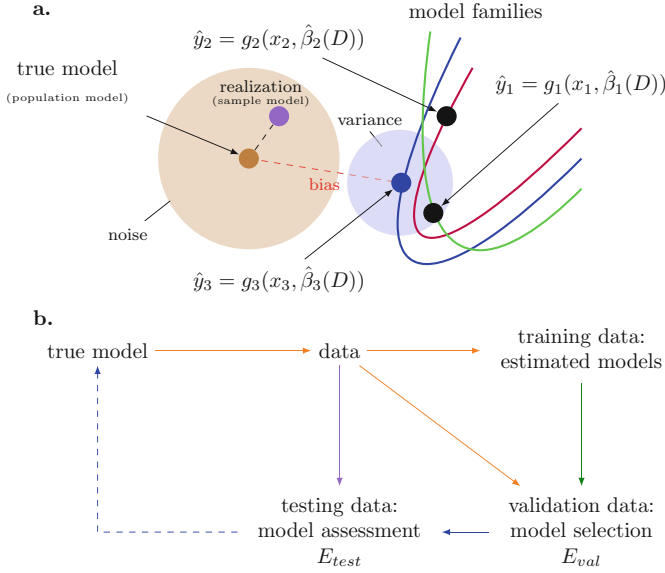


Fig. 12.1 Visualization of the theoretical problem of model selection. (a): Three model families, g_1 , g_2 , and g_3 , are shown as well as the estimates of three specific models obtained using the training data. (b): A summary combining model selection and model assessment, emphasizing that different data sets are used for the different analysis steps, is shown

the training data. For a given instance, x , each of these models makes a prediction $\hat{y}_i = g_i(x, \hat{\beta}_i(D))$ (with $i \in \{1, \dots, 3\}$).

After the parameters of the three models have been estimated, one performs a model selection for identifying the best model according to a chosen criterion. For this, a validation data set is used. The best model should be “closest” to the true model (shown in brown). An additional problem is that the true model as well as the estimated best model has a variability due to the finite size of the data and noise, e.g., due to measurement errors. This is indicated by the light brown and light blue circles in Fig. 12.1a. As a consequence, the true model may appear differently as indicated by the “realization” of the true model. Similarly, there is not just one best model but different realizations, all of which fall within the light blue circle in Fig. 12.1a. Formally, the models at the center of these circles correspond to population models, whereas each model within the large circles correspond to sample models. Finally, one performs a model assessment of the best model, using a test data set.

In Fig. 12.1b, a summary of the entire model selection process is shown. Here we emphasize that different data (training data, validation data, and testing data) are used for the different analysis step. Assuming an ideal (very large) data set D , there are no problems with the practical realization of these steps. However, practically, we have no ideal data set but one with a finite sample size. For this reason, there are various approximations to realize model selection practically.

In the following, we discuss three different model selection methods. The first one is for multiple linear regression models and the second one for Bayesian models, and the third one can be used for general models.

12.4 Model Selection for Multiple Linear Regression Models

The methods we discuss in this section, except the AIC and BIC, are specifically for multiple linear regression models. This means that for other types of models, they cannot be used.

12.4.1 R^2 and Adjusted R^2

The first measure we discuss is called the *coefficient of determination* (COD) [117, 504]. The COD is defined as

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}. \quad (12.1)$$

This definition is based on SSR, SST and SSE given as follows:

$$SST = TSS = \sum_{i=1}^n (y_i - \bar{y})^2 = \|Y - \bar{Y}\|^2; \quad (12.2)$$

$$SSR = ESS = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2 = \|\hat{Y} - \bar{Y}\|^2; \quad (12.3)$$

$$SSE = RSS = \sum_{i=1}^n (\hat{y}_i - y_i)^2 = \sum_{i=1}^n e_i^2 = \|\hat{Y} - Y\|^2. \quad (12.4)$$

Here, $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$ is the mean value of the predictor variable, and $e_i = \hat{y}_i - y_i$ are the residuals.

The COD is a measure of how well the model explains the variance of the response variables. A disadvantage of R^2 is that a submodel of a full model has always a smaller value regardless of its quality. For this reason, a modified version of R^2 has been introduced called the *adjusted coefficient of determination* (ACOD). The ACOD is defined as

$$R_{adj}^2 = 1 - \frac{SSE(n-1)}{SST(n-p)}. \quad (12.5)$$

It can also be written as a function of R^2 as follows:

$$R_{adj}^2 = 1 - \frac{(n-1)}{(n-p-1)}(1-R^2). \quad (12.6)$$

The ACOD adjusts for sample size n of the training data and the model complexity, defined by the number of covariates p .

12.4.2 Mallow's Cp Statistic

For a general model, using in-sample data $\{(x_i, y_i)\}$ for training and out-sample data $\{(x_i, y'_i)\}$ for testing, one can show that

$$\mathbb{E} \left[\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \right] < \mathbb{E} \left[\frac{1}{n} \sum_{i=1}^n (y'_i - \hat{y}_i)^2 \right]. \quad (12.7)$$

Furthermore, if the model is linear, having p predictors and an intercept, one can show that

$$\mathbb{E} \left[\frac{1}{n} \sum_{i=1}^n (y'_i - \hat{y}_i)^2 \right] = \mathbb{E} \left[\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \right] + \underbrace{\frac{2}{n} \sigma^2 (p+1)}_{\text{optimism}}. \quad (12.8)$$

The last term in Eq. 12.8 is called the *optimism* because it is the amount by which the in-sample error underestimates the out-sample error. Hence, a large value of the optimism indicates a large discrepancy between both errors. It is interesting to note that:

1. The optimism increases with σ^2 .
2. The optimism increases with p .
3. The optimism decreases with n .

The above relationship can be explained as follows:

1. Adding more noise (indicated by increasing σ^2) and leaving n and p fixed makes it harder for a model to be learned.
2. Increasing the complexity of the model (indicated by increasing p) and leaving σ^2 and n fixed makes it easier for a model to fit the test data but makes it prone to overfitting.
3. Increasing the test data set (indicated by increasing n) and leaving σ^2 and p fixed reduces the chances of overfitting.

The problem with Eq. 12.8 is that σ^2 corresponds to the true value of the noise, which is unknown. For this reason, one needs to use an estimator to obtain a

reasonable approximation. One can show that by estimating $\hat{\sigma}^2$ from the largest (i.e., the least parsimonious) model, then this will be an unbiased estimator of σ^2 , if the true model is smaller (i.e., the most parsimonious).

Using this estimate of σ^2 leads to Mallows's C_p statistic [192, 530] given by

$$C_p = \mathbb{E} \left[\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \right] + \frac{2}{n} \hat{\sigma}^2 (p + 1). \quad (12.9)$$

Alternatively, we can write Eq. 12.9 as follows:

$$C_p = \text{MSE} + \frac{2}{n} \hat{\sigma}^2 (p + 1). \quad (12.10)$$

For model selection, one chooses the model that minimizes C_p . Mallows's C_p is only used for linear regression models that are evaluated with the squared error.

12.4.3 Akaike's Information Criterion (AIC) and Schwarz's BIC

The next two model selection criteria have similarity to Eq. 12.10.

Akaike's Information Criterion (AIC) The Akaike's information criterion (AIC) [5, 62, 453] for a model $\mathcal{M}$ is defined by

$$\text{AIC}(\mathcal{M}) = -2 \log(L_{\mathcal{M}}) + 2 \dim(\mathcal{M}). \quad (12.11)$$

Here, $L_{\mathcal{M}}$ is the likelihood of the model $\mathcal{M}$, evaluated at the maximum likelihood estimate, and $\dim(\mathcal{M})$ is the dimension of the model corresponding to the number of free parameters. In contrast with Mallows's C_p , the Akaike's information criterion selects the model that maximizes $\text{AIC}(\mathcal{M})$.

For a linear model, one can show that the log likelihood is given by

$$\log(L_{\mathcal{M}}) = -\frac{n}{2} \log(\text{MSE}) + C', \quad (12.12)$$

where C' is a model independent constant and the dimension of the model is

$$\dim(\mathcal{M}) = p + 2. \quad (12.13)$$

Taken together, this gives

$$\text{AIC}(\mathcal{M}) = n \log(\text{MSE}) + 2p + C, \quad (12.14)$$

with $C = -2C' + 4$. For model comparisons, the parameter C is irrelevant.

The BIC (Bayesian Information Criterion) BIC [354, 423], also called the Schwarz criterion, has a similar form as the AIC. The BIC is defined by

$$\text{BIC}(\mathcal{M}) = -2 \log(L_{\mathcal{M}}) + p \log(n). \quad (12.15)$$

For a linear model with normal distributed errors, this simplifies to

$$\text{BIC}(\mathcal{M}) = n \log(\text{MSE}) + p \log(n). \quad (12.16)$$

The BIC selects the model that maximizes $\text{BIC}(\mathcal{M})$.

The common idea of AIC and BIC is to penalize larger models. Since $\log(n = 8) > 2$, then the BIC penalizes stronger than AIC (usually data sets have more than 8 samples). Hence, the BIC selects smaller models (i.e., more parsimonious) compared to the AIC. The BIC has a consistency property, meaning that when the true unknown model is one of the models under consideration and the sample size, n , tends to infinity, the BIC selects the correct model. In contrast, the AIC does not have this consistency property.

In general, the AIC and the BIC are considered to have a different view on model selection [4]. The BIC assumes that the true model is among the studied ones, and its goal is to identify it. In contrast, the AIC does not assume this; instead, the goal of the AIC is to find the model that maximizes the predictive accuracy. In practice, the true model is rarely among the model families studied, and for this reason, the BIC cannot select the true model. For such a case, the AIC is the appropriate approach for finding the best approximating model. Several studies preferred the AIC over the BIC for practical applications [4, 62, 354]. For instance, in [484], it was found that the AIC can select a better model compared to the BIC, even when the true model is among the studied models. Specifically, for regression models, in [508] it has been demonstrated that the AIC is asymptotically efficient for selecting the model with the least MSE, while the BIC does not when the true model is not among the studied models.

In summary, the AIC and the BIC have the following characteristics:

- The BIC selects smaller models (more parsimonious) compared to the AIC and therefore more prone to underfitting.
- The AIC selects larger models (less parsimonious) compared to the BIC and therefore more prone to overfitting.
- The AIC represents a frequentist point of view.
- The BIC represents a Bayesian point of view.
- The AIC is asymptotically efficient but not consistent.
- The BIC is consistent but not asymptotically efficient.
- The AIC should be used when the objective is the prediction accuracy of a model.
- The BIC should be used when the objective is the model interpretability.

The AIC and the BIC are generic in their applications not limited to linear models, and they can be applied whenever we have a likelihood of a model [292].

12.4.4 Best Subset Selection

So far, we discussed evaluation criteria that can be used for model selection. However, we did not discuss how these criteria are actually used. In the following, we provide this information, discussing best subset selection (Algorithm 7), forward stepwise selection (Algorithm 8), and backward stepwise selection (Algorithm 9) [27, 101, 117]. All of these approaches are computational.

The most brute force model selection strategy is to evaluate each possible model. This is the idea of best subset selection (Best).

Algorithm 7: Best subset selection.

Input: A model family $\mathcal{M}$ to be fitted.

Let $\hat{\mathcal{M}}_0$ denote the fitted model with zero parameters.

for $k = 1, \dots, p$ **do**

Fit all $\binom{p}{k}$ models having k parameters.

Select the best model with k parameters and call it $\hat{\mathcal{M}}_k$. This evaluation is based on minimizing the MSE or on maximizing R^2 .

Select the best model from $\{\hat{\mathcal{M}}_0, \dots, \hat{\mathcal{M}}_p\}$ using C_p , the AIC or the BIC as evaluation criterion or cross validation.

Best subset selection evaluates each model with k parameters by the MSE or R^2 . Due to the fact that each of these models has the same complexity (a model with k parameters), measures based on the model complexity are not needed. However, when comparing the $p + 1$ different models, having different parameters (see line 5 in Algorithm 7), a complexity penalizing measure such as the C_p , AIC, or BIC needs to be used.

For a linear regression model, one needs to fit all combinations with p predictors. A problem with the best subset selection is that in total, one needs to evaluate $\sum_{k=0}^p \binom{p}{k} = 2^p$ different models. For $p = 20$, this gives already over 10^6 models, leading to computational problems in practice. For this reason, approximations to the best subset selection are needed.

12.4.5 Stepwise Selection

Two such approximations are discussed in the following. Both of these follow a greedy approach, where the forward stepwise selection proceeds in a bottom-up manner, while the backward stepwise selection proceed in a top-down manner.

12.4.5.1 Forward Stepwise Selection

The idea of the forward stepwise selection (FSS) is to start with a null model, without parameters, and add successively one parameter at a time that is best according to a selection criterion.

For a linear regression model, with p predictors, this gives

$$1 + \sum_{k=0}^{p-1} (p - k) = 1 + \frac{p(p+1)}{2} \quad (12.17)$$

models. For $p = 20$, this gives only 211 different models that need to be evaluated. This is a great improvement compared to best subset selection.

Algorithm 8: Forward stepwise selection.

Input: A model family $\mathcal{M}$ to be fitted.

Let $\hat{\mathcal{M}}_0$ denote the fitted model with zero parameters.

for $k = 0, \dots, p - 1$ **do**

 Fit all $p - k$ models having $k + 1$ parameters.

 Select the best of these $p - k$ models and call it $\hat{\mathcal{M}}_{k+1}$. This evaluation is based on minimizing the MSE or on maximizing R^2 or cross validation.

Select the best model from $\{\hat{\mathcal{M}}_0, \dots, \hat{\mathcal{M}}_p\}$ using C_p , the AIC or the BIC as evaluation criterion.

12.4.5.2 Backward Stepwise Selection

The idea of the backward stepwise selection (BSS) is to start with a full model with p parameters and remove successively one parameter at a time that is worst according to a selection criterion.

Algorithm 9: Backward stepwise selection.

Input: A model M to be fitted.

Let $\hat{\mathcal{M}}_p$ denote the fitted model with p parameters.

for $k = p, \dots, 1$ **do**

 Fit all k models having $k - 1$ parameters from the parameters of model $\hat{\mathcal{M}}_k$.

 Select the best of these k models and call it $\hat{\mathcal{M}}_{k-1}$. The evaluation of this is based on minimizing the MSE or on maximizing R^2 .

Select the best model from $\{\hat{\mathcal{M}}_0, \dots, \hat{\mathcal{M}}_p\}$ by using C_p , the AIC or the BIC as evaluation criterion.

The number of models that need to be evaluated using the backward stepwise selection is exactly the same as for the forward stepwise selection.

Both stepwise selection strategies are not guaranteed to find the best model containing a subset of the p predictors. However, when p is large, both approaches may be the only ones practically feasible. Despite the apparent symmetry of the forward stepwise selection and the backward stepwise selection, there is a difference in situations where $p > n$, i.e., when we have more parameters than samples in our data. In this case, the forward stepwise selection approach can still be applied because the procedure may be systematically limited to n parameters.

12.5 Model Selection for Generalized Linear Models

In Sect. 11.6.4, we introduced generalized linear models as an extension of OLS regression models. Now, we continue our discussion of the Poisson model in the context of model selection. Specifically, the results in Sect. 11.6.4.3 showed that the Poisson regression model discussed there did not provide a good fit for the studied data. This was indicated by the deviance goodness-of-fit test. In the following section, we continue this discussion by investigating a number of possible GLMs that could be used instead of a Poisson regression model.

12.5.1 Negative Binomial Regression Model

A possible problem with a Poisson distribution is that it assumes that the mean and the variance are the same. However, sometimes more flexibility is needed when the variance is greater than the mean, which is called overdispersion. In such a situation, a negative binomial regression model can be used because the negative binomial distribution has one parameter more than the Poisson distribution that allows to adjust the variance independently from its mean.

Specifically, a negative binomial regression model assumes that the variance is a quadratic function of the mean:

$$\mathbb{E}[Y|x] = \mu, \quad (12.18)$$

$$\text{Var}[Y|x] = \mu + \mu^2 \phi. \quad (12.19)$$

Here ϕ is the dispersion parameter.

In Listing 12.1, we show a negative binomial regression with log as link function for the same data used as in Sect. 11.6.4.3. The covariate “ment” corresponds to the articles published by the mentor in the last 3 years, and the covariate “art” is the numbers of articles published during the Ph.D.

Listing 12.1: Negative Binomial regression model with Log as link function

```

library(foreign)
library(MASS)
dat <- read.dta("http://www.stata-press.com/data/lf2/couart2.dta")

res.nb <- glm.nb(art~ment, link = log, data=dat)
summary(res.nb)
# output
Call:
glm.nb(formula = art ~ ment, data = dat, link = log, init.theta =
      2.135947279)

Deviance Residuals:
      Min       1Q   Median       3Q      Max
-2.1656  -1.4160  -0.2836   0.3904   3.4626

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.216113    0.047114   4.587  4.5e-06 ***
ment         0.030301    0.003129   9.685  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Negative Binomial(2.1359) family taken
 to be 1)

Null deviance: 1087.0  on 914  degrees of freedom
Residual deviance: 1002.5  on 913  degrees of freedom
AIC: 3145.7

Number of Fisher Scoring iterations: 1

              Theta:  2.136
              Std. Err.:  0.247

2 x log-likelihood:  -3139.675

```

Let's perform some tests to check the quality of the above fit. In Listing 12.2, we show the deviance goodness-of-fit test and in Listing 12.3 the dispersion test.

Listing 12.2: Deviance goodness-of-fit test

```

pchisq(res.nb$deviance, df=res.nb$df.residual, lower.tail=FALSE)
[1] 0.02039674

```

The null hypothesis for deviance goodness-of-fit is: “our model is correctly specified.” Given that the resulting p -value is around 0.05 (see Listing 12.2), we do not have strong evidence to reject the null hypothesis. In general, a high p -value indicates no evidence of lack-of-fit, and a low p -value indicates evidence of lack-of-fit.

Listing 12.3: Dispersion test

```
library(DHARMA)
sim_res.nb <- simulateResiduals(res.nb, refit=T)
testDispersion(sim_res.nb, alternative = "two.sided")
# output
DHARMA nonparametric dispersion test via mean deviance
      residual fitted
vs. simulated-refitted

data:  sim_res.nb
dispersion = 1.0371, p-value = 0.12
alternative hypothesis: two.sided
```

The null hypothesis for the dispersion test is “There is no difference in dispersion.” By specifying the alternative hypothesis as “two.sided,” the dispersion can correspond to an over- or underdispersion, and the parameter “less” is for underdispersion, whereas the parameter “greater” is for overdispersion. As one can see from Listing 12.3, the p -value does not allow to reject the null hypothesis.

These results look better than for the Poisson model in Sect. 11.6.4.3.

12.5.2 Zero-Inflated Poisson Model

Another possible problem with a Poisson regression model, which can also lead to overdispersion, is the presence of too many zero-count observations. This can be modeled with a zero-inflated Poisson model, given by

$$f(y|x) = \begin{cases} \pi + (1 - \pi) \exp^{-\lambda} & \text{if } y = 0, \\ (1 - \pi) \frac{\lambda^y \exp^{-\lambda}}{y!} & \text{if } y > 0. \end{cases} \quad (12.20)$$

From this model, one can obtain the mean and the variance as follows:

$$\mathbb{E}[Y|x] = (1 - \pi)\lambda, \quad (12.21)$$

$$\text{Var}[Y|x] = (1 - \pi)(\lambda^2 + \lambda). \quad (12.22)$$

where π is the probability that an observation is zero.

From Fig. 11.10, one can see that the number of zeros in our data is indeed quite large (there are 275 zeros).

Listing 12.4: Zero-inflated Poisson model

```

res.zero.inflated <- zeroinfl(art ~ ment | ment, dist =
  'poisson', data = dat)
summary(res.zero.inflated)
# output
Call:
zeroinfl(formula = art ~ ment | ment, data = dat, dist =
  "poisson")

Pearson residuals:
      Min       1Q   Median       3Q      Max
-2.2503 -0.8754 -0.2820  0.4747  8.3542

Count model coefficients (poisson with log link):
      Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.54205     0.04214  12.862  <2e-16 ***
ment         0.01808     0.00219   8.258  <2e-16 ***

Zero-inflation model coefficients (binomial with logit link):
      Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.63383     0.19486  -3.253  0.00114 **
ment        -0.12730     0.03733  -3.410  0.00065 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Number of iterations in BFGS optimization: 16
Log-likelihood: -1618 on 4 Df

```

12.5.3 Quasi-Poisson Model

For a Quasi-Poisson regression, the variance is assumed to be a linear function of the mean. The model fits an extra dispersion parameter to account for that extra variance:

$$\mathbb{E}[Y|x] = \mu, \quad (12.23)$$

$$\text{Var}[Y|x] = \mu\phi. \quad (12.24)$$

Here, ϕ is the dispersion parameter.

From Listing 12.5, we can see that the Residual deviance does not show much improvement upon the Poisson regression model. Furthermore, we would like to note that the AIC does not return a value, because there exists no maximum likelihood for a Quasi-Poisson model.

On a technical note, we would like to add that the dispersion parameter, as shown in the output in Listing 12.5, can be obtained via the following command in Listing 12.6.

Listing 12.5: Quasi-Poisson model

```

es.quasi.poisson <- glm(art~ment, family = quasipoisson(link =
  "log"), data = dat)
summary(res.quasi.poisson)
# output
glm(formula = art ~ ment, family = quasipoisson(link = "log"),
  data = dat)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.5700  -1.6316  -0.3598   0.5068   5.9483

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.259906   0.047171   5.510 4.67e-08 ***
ment         0.026050   0.002632   9.896 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be
 1.884658)

Null deviance: 1817.4 on 914 degrees of freedom
Residual deviance: 1669.5 on 913 degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 5

```

Listing 12.6: Dispersion of the Quasi-Poisson model

```

summary(res.quasi.poisson)$dispersion
[1] 1.884658

```

Overall, one can see that the result is similar using the Poisson regression model (see Sect. 11.6.4.3), hence, this does not provide an alternative for the studied data.

12.5.4 Comparison of GLMs

While in the previous section, we discussed several GLMs individually, in this section we discuss a test allowing to compare pairs of GLMs.

To compare pairs of GLMs, one can use the Vuong test. The Vuong test is a likelihood ratio test, which can be used for nested and non-nested models [485]. Nested models just differ in the covariates used for a fit, and one model is fully embedded in the larger model. In contrast, non-nested models can assume general forms, and they cannot be reduced into each other [122].

A Vuong test is similar to a traditional likelihood ratio test (LRT), but differs in the sampling distribution. While a LRT uses a chi-square null distribution, the Vuong LRT uses a weighted sum of chi-square distributions. The latter distribution

converges to the traditional chi-square distribution when the full model is the true model (for nested models).

From Listing 12.7, one can see that there are actually two different hypothesis tests shown: variance test and non-nested likelihood ratio test. The reason for this is that, it is often difficult to see whether or not non-nested models are overlapping. The null hypothesis of these tests are:

- H_0 for the variance test: “Model 1 and model 2 are indistinguishable.”
- H_0 for the non-nested likelihood ratio test: “Model fit’s of both models are equal.”

Listing 12.7: Vuong test: Pairwise comparison of GLMs

```
library("nonnest2")
vuongtest(res.poisson, res.nb, nested=FALSE)
# output
Model 1
Class: glm
Call: glm(formula = art ~ ment, family = poisson(link = "log"),
  data = dat)

Model 2
Class: negbin
Call: glm.nb(formula = art ~ ment, data = dat, link = log,
  init.theta = 2.135947279)

Variance test
H0: Model 1 and Model 2 are indistinguishable
H1: Model 1 and Model 2 are distinguishable
w2 = 0.587,    p = 0.000104

Non-nested likelihood ratio test
H0: Model fits are equal for the focal population
H1A: Model 1 fits better than Model 2
z = -4.264,    p = 1
H1B: Model 2 fits better than Model 1
z = -4.264,    p = 1.004e-05
```

For a significance level $\alpha = 0.05$, both null hypothesis need to be rejected, although the p -value for the variance test is much larger than for the non-nested likelihood ratio test. This is understandable because the Poisson regression model can be considered as a special case of the negative binomial model. The comparison of the other pairs of GLMs is left as an exercise.

One can also use Vuong’s theory to obtain confidence intervals of the AIC and the BIC for non-nested models. The results of this analysis are shown in Listing 12.8.

Here, it is important to note that the 95% confidence intervals of both the AIC differences and the BIC differences do not overlap with zero. This means that AIC1 (for model 1) is always larger than AIC2 (for model 2) and similarly for the BICs. Hence, model 1 can be considered a better fit compared to model 2.

Finally, we can compare more than two GLMs with each other based on AIC or BIC, as shown in Listing 12.9. This example compares the Poisson regression model, the negative binomial model, and the zero-inflated Poisson model with each other.

Listing 12.8: Confidence intervals of AIC and BIC for non-nested models

```

icci(res.poisson, res.nb)
#output
Model 1
  Class: glm
  Call: glm(formula = art ~ ment, family = poisson(link = "log"),
            data = dat)
  AIC: 3341.286
  BIC: 3350.924

Model 2
  Class: negbin
  Call: glm.nb(formula = art ~ ment, data = dat, link = log,
               init.theta = 2.135947279)
  AIC: 3145.675
  BIC: 3160.132

95% Confidence Interval of AIC difference (AICdiff = AIC1 - AIC2)
104.780 < AICdiff < 286.443

95% Confidence Interval of BIC difference (BICdiff = BIC1 - BIC2)
99.961 < BICdiff < 281.624

```

Listing 12.9: AIC and BIC for GLMs

```

AIC(res.poisson, res.nb, res.zero.inflated)
# output
              df      AIC
res.poisson    2 3341.286
res.nb         3 3145.675
res.zero.inflated 4 3244.915

BIC(res.poisson, res.nb, res.zero.inflated)
# output
              df      BIC
res.poisson    2 3350.924
res.nb         3 3160.132
res.zero.inflated 4 3264.191

```

12.6 Model Selection for Bayesian Models

In this section, we discuss model selection yet for another specific class of models, namely, Bayesian models. This means that the method discussed below can be used for probabilistic models, which allow the estimation of a posterior distribution.

The model selection criterion we discuss in the following, is called the Bayes' factor [263, 272, 295, 343]. Suppose that we have a finite set of candidate models $\{\mathcal{M}_i\}_{i=1}^M$ with $i \in \{1 \dots M\}$, we can use for fitting the data D . In order to select the best model from a Bayesian perspective, we need to evaluate the posterior probability of each model, given by

$$P(\mathcal{M}_i|D), \quad (12.25)$$

for the available data. Using Bayes' theorem, one can write this probability as

$$P(\mathcal{M}_i|D) = \frac{p(D|\mathcal{M}_i)P(\mathcal{M}_i)}{\sum_{j=1}^M p(D|\mathcal{M}_j)P(\mathcal{M}_j)}. \quad (12.26)$$

Here, the term $p(D|\mathcal{M}_i)$ is called the **evidence for the model** $\mathcal{M}_i$ or simply evidence.

The ratio of the posterior probabilities for models $\mathcal{M}_A$ and $\mathcal{M}_B$, corresponding to the **posterior odds** of the models, is given by

$$\begin{aligned} \frac{P(\mathcal{M}_A|D)}{P(\mathcal{M}_B|D)} &= \frac{p(D|\mathcal{M}_A)P(\mathcal{M}_A)}{p(D|\mathcal{M}_B)Pr(\mathcal{M}_B)} = \frac{p(D|\mathcal{M}_A)}{p(D|\mathcal{M}_B)} \times \text{prior odds} \\ &= BF_{AB} \times \text{prior odds}. \end{aligned}$$

That means the *Bayes' factor* of the models is the ratio of the posterior probabilities and the prior probabilities:

$$BF_{AB} = \frac{\text{posterior odds}}{\text{prior odds}} = \frac{\frac{P(\mathcal{M}_A|D)}{P(\mathcal{M}_B|D)}}{\frac{P(\mathcal{M}_A)}{P(\mathcal{M}_B)}}. \quad (12.27)$$

For non-informative priors, i.e., $P(\mathcal{M}_A) = P(\mathcal{M}_B) = 0.5$, the *Bayes' factor* simplifies to

$$BF_{AB} = \text{posterior odds} = \frac{P(\mathcal{M}_A|D)}{P(\mathcal{M}_B|D)}. \quad (12.28)$$

In order to see practical problems when using the Bayes' factor, let's assume that a model $\mathcal{M}_i$ depends on the parameter θ . Then, the evidence can be written as

$$p(D|\mathcal{M}_i) = \int p(D|\theta, \mathcal{M}_i)p(\theta|\mathcal{M}_i)d\theta. \quad (12.29)$$

A serious problem with this expression is that it can be very hard to evaluate numerically. Especially in high dimensions, where no closed-form solution is available. This is a general problem of the Bayes' factor, which makes its application problematic.

It is interesting to note that there is a close connection between the BIC and the Bayes' factor. Specifically, in [272] it has been proven that for $n \rightarrow \infty$ the following holds:

$$2 \ln BF_{BA} \approx \text{BIC}_A - \text{BIC}_B. \quad (12.30)$$

Note that the relation $-\ln BF_{AB} = \ln BF_{BA}$ is negatively symmetric. Equation 12.30 implies that model comparison results for the BIC and the Bayes' factor can approximate each other.

Table 12.1 Interpretation of the comparison of models with the BIC and Bayes’ factor

Evidence	$\Delta \text{BIC} = \text{BIC}_k - \text{BIC}_{\min}$	$\text{BF}_{\min,k}$
Weak	0 – 2	1 – 3
Positive	2 – 6	3 – 20
Strong	6 – 10	20 – 150
Very strong	> 10	> 150

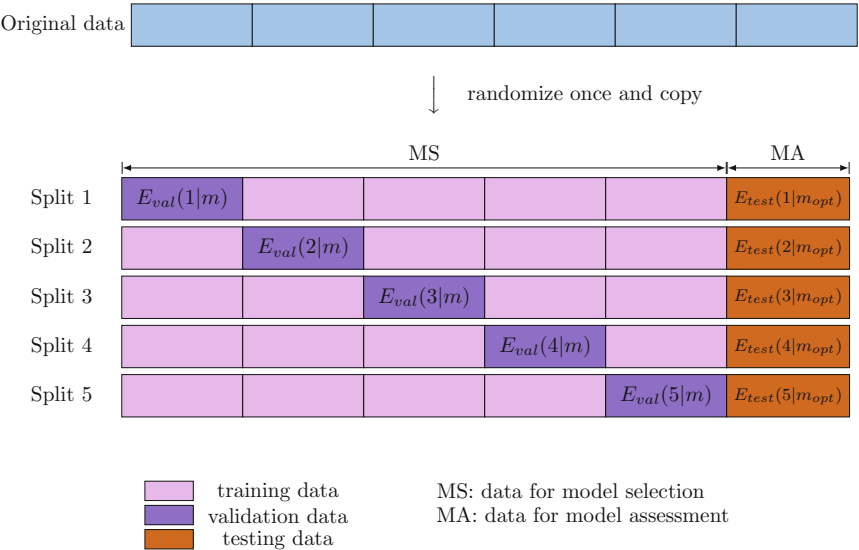


Fig. 12.2 Visualization of resampling for model selection (MS) and model assessment (MA). Each model, $m \in \{1, \dots, M\}$, is evaluated with the validation data. Finally, only the optimal model, m_{opt} , is evaluated using the testing data

For practical applications and for interpreting the BIC and Bayes’ factors, in [394] the following evaluation, for comparing two models (see Table 12.1), was suggested. Here, “min” indicates the model with the smaller BIC or posterior probability.

12.7 Nonparametric Model Selection for General Models with Resampling

In contrast with the previous sections, the nonparametric approach discussed below can be used for general statistical models, including multiple linear regression models and Bayesian models. This approach is based on the resampling of data, forming an extension of k-fold CV discussed in Sect. 4.2.3. Due to the data-based nature of the resampling method, this allows also to perform model assessment in addition to model selection.

The idea of the resampling method is shown in Fig. 12.2. To perform model selection (MS) and model assessment (MA), the data are split into three parts. One part of the data is used for training, one for validation and one for testing. The splits of the data are obtained by randomization of the original data. These data are then copied and split as shown in Fig. 12.2. Specifically, the separation is similar to a k -fold CV with one additional fold. In Fig. 12.2 the example shows a fivefold CV.

When we were discussing general resampling methods in Chap. 4, there was no model selection involved because we assumed that only one model is studied. Hence, only training data and testing data were needed. This corresponded to model assessment. However, to perform model selection and model assessment, one needs three different data sets for performing these two tasks. Importantly, in Fig. 12.2 one can see that the data for both stages, i.e., MS and MA, are non-overlapping. That means no data point in the testing data is ever used for training and validation and vice versa.

The example shown in Fig. 12.2 visualizes the usage of a fivefold CV; however, any other resampling method can be used for model selection and model assessment in a similar way. That means the MS part needs to be adopted as described for the resampling methods in Chap. 4, while the MA part uses just a separate proportion of the data (similar to the holdout set method; see Sect. 4.2.1).

Formally, model selection operates as follows. For each split i ($i \in \{1, \dots, k\}$), the parameters of model m ($m \in \{1, \dots, M\}$) are estimated using the training data, and the prediction error is evaluated using the validation data, i.e.,

$$E_{\text{val}}(i|m). \quad (12.31)$$

After the last split, the errors are summarized by

$$E_{\text{val}}(m) = \frac{1}{k} \sum_{i=1}^k E_{\text{val}}(i|m). \quad (12.32)$$

This gives estimates of the prediction error for each model m . The best model can then be selected as

$$m_{\text{opt}} = \underset{m}{\operatorname{argmin}} \left\{ E_{\text{val}}(m) \right\}. \quad (12.33)$$

Using the optimal model, given by m_{opt} , one uses now the test data to estimate the generalization error for each split as follows:

$$E_{\text{test}}(i|m_{\text{opt}}). \quad (12.34)$$

Here, it is important to note that this is only performed for the optimal model. These results can be again summarized as

$$E_{\text{test}}(m_{\text{opt}}) = \frac{1}{k} \sum_{i=1}^k E_{\text{test}}(i|m_{\text{opt}}). \quad (12.35)$$

In Chap. 18, we will formally define what we mean by the *expected generalization error* and discuss why this entity can only be approximated in reality, e.g., by means of resampling methods or model assessment.

Overall, a nonparametric model selection method is very flexible, but requires a sufficiently large data set in order to split the data into $k+1$ folds, to perform the MS and MA part. This is the only assumption this method is based on. Given that we are living in the *big data* era, for most contemporary data sets, this should not be a problem. However, a potential problem is the computational burden. Therefore, this assumes that computational resources are available to execute an analysis multiple times.

12.8 Summary

In this chapter, we discussed two types of model selection methods. The first type of methods were parametric and the second type nonparametric. The parametric methods, e.g., R^2 , Mallows's Cp statistic, or AIC, are elegant and simple in their application because models are selected based on an analytical expression. The price to be paid for this elegance comes in the form of assumptions that need to be made, which limit the application of these methods to specific statistical models. For instance, R^2 and Mallows's Cp statistic can only be used for multiple linear regression models and Bayes' factors for Bayesian models. In contrast, nonparametric resampling methods for model selection are non-elegant without analytical expressions. However, the numerical nature of this approach is very flexible and applicable to any kind of statistical model regardless of its nature.

In Fig. 12.3, we summarize different model selection approaches. In this figure, we highlight two important characteristics of such methods. The first characteristics distinguishes methods in terms of data splitting, and the second is in terms of model complexity. Neither best subset selection (Best) nor the forward stepwise selection (FSS) or the backward stepwise selection (BSS) applies data splitting, but they use the entire data for evaluation. Furthermore, each of these approaches is a two-step procedure that employs in its first step a measure, which does not consider the model complexity. For instance, in this step either the MSE or R^2 is used. Then, in the second step a measure considering model complexity is used, e.g., the AIC, the BIC, or C_p .

Data splitting is typically based on resampling of the data, and in this chapter, we used cross-validation. Interestingly, CV can be used with or without model complexity measures. For instance, regularized regression models, e.g., ridge regression, LASSO, or elastic net, consider the complexity by varying the value of λ (regularization parameter). Regularization is the topic of Chap. 13.

		model complexity		
		no	yes	
		MSE	AIC	regularization
data splitting	no	Best	1	2
		FSS	1	2
		BSS	1	2
	yes	CV	x	
		CV		x

Fig. 12.3 Summary of different model selection approaches. Here, the AIC stands for any criterion considering model complexity, e.g., the BIC or C_p , and regularization is any regularized regression model, e.g., LASSO or elastic net

For model selection, cross-validation (CV) is the most practical and flexible approach one can use [13, 178, 448]. The reason for this is that it is conceptually simple, it is intuitive, and it can be applied to any statistical model family regardless of its technical details (for instance, to parametric and nonparametric models). Compared with other approaches for model selection, cross-validation has the following advantages:

- Cross-validation is a computational method that is simple in its realization.
- Cross-validation makes few assumptions about the true underlying model.
- Compared with the AIC, the BIC, and the adjusted R^2 , cross-validation provides a direct estimate of the prediction error.
- Every data point is used for both training and testing.

Learning Outcome 12: Model Selection

For finite data, model selection aims to rank candidate models and obtain the same order as when using infinite data to estimate the expected generalization error.

In summary, cross-validation, the AIC, and C_p all have the same goal: try to find a model that predicts best, and practically they all tend to choose similar models. On the other hand, the BIC is quite different and tends to choose smaller models. The goal of BIC is also different because it tries to identify the true model. In general, smaller models are easier to interpret, providing a better understanding of

the underlying process. Overall, cross-validation is the most general approach, and it can be used for parametric as well as nonparametric models.

12.9 Exercises

1. What are the difference between the best subset selection and the stepwise selection?
2. What are the main difference between the AIC and the BIC, including their respective strengths and weaknesses?
3. Discuss the definition of the Bayes' factor for informative priors.
4. Why is the expected generalization error important in selecting the best model? Hint: See the discussion of the expected generalization error in Chap. 18.
5. Select a prediction model for an application that does not allow to use a parametric criterion, e.g., Mallows' C_p statistic or AIC and outline the nonparametric approach that can be used in this case.
6. Compare the results from a Poisson regression model with a Zero-inflated Poisson regression model using the Vuong test.

Part III

Advanced Topics

Chapter 13

Regularization



13.1 Introduction

In this chapter, we discuss extensions of regression models introduced in Chap. 11. Despite the success and importance of ordinary least squares (OLS) regression models, high-dimensional data require a modification of this framework to deal with them efficiently. For this reason, in recent years new regression models have been introduced that extend classical regression models significantly. These models are based on *regularization*, which is a concept introduced by Tikhonov to deal with ill-posed inverse problems [40, 91, 463]. We will see that depending on the mathematical formulation of the regularization, different regression models can be derived. Maybe the most prominent one of them is the *least absolute shrinkage and selection operator (LASSO)* model.

The idea underlying regularization is to modify the optimization functional of a (regression) model using an additional parameterized term. This framework is very general and not limited to regression models, and it can be applied to any optimization functional of models, e.g., for deep neural networks. In the context of regression models, regularization can lead to two effects. First, the absolute value of regression coefficients can be reduced. Second, some regression coefficients can even be shrunk to zero. While both effects will generally improve the prediction abilities and reduce overfitting of the model, only the later performs a type of model selection on the number of parameters of the regression model. Interestingly, the shrinkage of regression coefficients is performed automatically by the model, and does not require manual intervention. This makes it a very desirable feature, explaining part of the success of models like the LASSO.

In this chapter, we discuss a number of different regularized regression models including ridge regression, non-negative garrote, LASSO, Dantzig selector, adaptive LASSO, elastic net, and group LASSO [442, 462]. We will see that each one has its own merits and is not necessarily making other models obsolete. Instead, the selection needs to be guided by the application.

13.2 Preliminaries

We begin this chapter by briefly providing some statistical preliminaries needed for the regression models. First, we discuss some preprocessing steps used to standardize the data for all regression models. Second, we discuss the data we are using to demonstrate the differences between the various regression models.

13.2.1 Preprocessing and Norms

Let's assume that we have some data of the form $(\mathbf{x}_i, y_i)$ with $i \in \{1, \dots, n\}$ where n is the number of samples, $\mathbf{x}_i = (X_{i1}, \dots, X_{ip})^T$ corresponds to the vector of predictor variables for sample i , p is the number of predictors, and y_i is the response variable. We denote by $\mathbf{y} \in \mathbb{R}^n$ the vector of response variables and by $\mathbf{X} \in \mathbb{R}^{n \times p}$ the predictor matrix. The vector $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^T$ indicates the regression coefficients.

The predictors and the response variable shall be standardized as follows:

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^n X_{ij} = 0 \quad \text{for all } j; \quad (13.1)$$

$$\bar{s}_j^2 = \frac{1}{n} \sum_{i=1}^n X_{ij}^2 = 1 \quad \text{for all } j; \quad (13.2)$$

$$\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i = 0. \quad (13.3)$$

Here, $\bar{x}_j$ and $\bar{s}_j^2$ are the mean and variance of the predictor variables, and $\bar{y}$ is the mean of the response variable.

In order to study the regularization of regression models, we need to solve optimization problems, which are formulated in terms of norms. For this reason, we review in the following the norms needed for the subsequent sections. For a real vector $\mathbf{x} \in \mathbb{R}^n$ and $q \geq 1$, the L $_q$ -norm is defined by

$$\|\mathbf{x}\|_q = \left(\sum_{i=1}^n |x_i|^q \right)^{\frac{1}{q}}. \quad (13.4)$$

The special case $q = 2$ gives the **L2-norm** (also known as **Euclidean norm**), and the case $q = 1$ gives the **L1-norm**. Interestingly, for $q < 1$, Eq. 13.4 is still defined, but it is no longer a norm in the mathematical sense.

We will revisit the L2-norm when discussing ridge regression and the L1-norm for the LASSO. The infinity norm, also called **maximum norm**, is defined by

$$\|\mathbf{x}\|_{\infty} = \max_i |x_i|. \quad (13.5)$$

This norm is used by the Danzig selector.

For $q = 0$, one obtains the *L0-norm* (which is strictly not a norm as mentioned above) corresponding to the number of non-zero elements, i.e.,

$$\|\mathbf{x}\|_0 = \#\text{non-zero elements}. \quad (13.6)$$

13.2.2 Data

To illustrate the methods in this chapter, we use a simulated data set. Listing 13.1 presents the code used to generate the data. This allows us to know the true regression model.

The simulated data consist of 20 uncorrelated predictor variables from which only the first 5 influence the response variable. The true values of the regression coefficients are given by $\boldsymbol{\beta} = (1, 2, 3, -1, -2)^T$, whereas all the other coefficients are zero.

Listing 13.1: Simulated data

```
set.seed(123456) # Set seed
n <- 200 # Number of samples of the data
p <- 20 # Number of predictor variables
p.eff <- 5 # Number of effective (true) predictors

x <- matrix(rnorm(n*p), nrow=n, ncol=p)
beta <- c(1, 2, 3, -1, -2)
xm <- x[,1:p.eff]
y <- xm %*% beta + rnorm(n)

# Split data: train (2/3) and test (1/3)
train_rows <- sample(1:n, 0.66*n)
x.train <- x[train_rows, ]
x.test <- x[-train_rows, ]

y.train <- y[train_rows]
y.test <- y[-train_rows]
```

In Listing 13.1, we set a seed, which allows to reproduce the same “random” data each time we run this code, since the seed initializes the pseudorandom number generator in R. Hence, if one wants to generate a new random data set, one needs to set a different seed.

13.2.3 R Packages for Regularization

Many of the following regression models, e.g., the LASSO, adaptive LASSO, and elastic net, can be implemented using the package *glmnet*, available in R. This package provides an efficient implementation via the cyclical coordinate descent method [174], and it has been demonstrated that even with thousands of predictors and samples, the parameters can be estimated quickly [174]. Hence, the package *glmnet* can be used not only for small data sets but also for large real-world data.

In order to perform a group LASSO, we use the package *gglasso* [509], available in R. However, there are alternative packages in R that could be used, such as *oem* [256, 507].

13.3 Ridge Regression

The motivation for improving OLS regression is the fact that the estimates from such models have often a low bias but a large variance. This relates to the prediction accuracy of a model because it is known that either by shrinking the values of regression coefficients or by setting some coefficients to zero, the accuracy of a prediction can be improved [223]. The reason for this is that by introducing some estimation bias, the variance can be actually reduced.

Ridge regression has been introduced by [243], and the corresponding model is formulated as follows:

$$\hat{\boldsymbol{\beta}}^{RR} = \arg \min \left\{ \frac{1}{2n} \sum_{i=1}^n (y_i - \sum_j x_{ij} \beta_j)^2 + \lambda \|\boldsymbol{\beta}\|_2^2 \right\} \quad (13.7)$$

$$= \arg \min \left\{ \frac{1}{2n} \text{RSS}(\boldsymbol{\beta}) + \lambda \|\boldsymbol{\beta}\|_2^2 \right\} \quad (13.8)$$

$$= \arg \min \left\{ \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_2^2 \right\}. \quad (13.9)$$

Here, $\text{RSS}(\boldsymbol{\beta})$ is the residual sum of squares (RSS) called the loss of the model, $\lambda \|\boldsymbol{\beta}\|_2^2$ is the regularization term or penalty, and λ is the tuning or regularization parameter. Specifically, the parameter λ controls the shrinkage of coefficients. The L2-penalty in Eq. 13.25 is sometimes also called Tikhonov regularization.

Ridge regression has an analytical solution, which is given by

$$\hat{\boldsymbol{\beta}}^{RR}(\lambda) = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_p)^{-1} \mathbf{X}^T \mathbf{y}. \quad (13.10)$$

Here, $\mathbf{I}_p$ is the $p \times p$ identity matrix. A problem of OLS is that if $\text{rank}(\mathbf{X}) < p$, then $\mathbf{X}^T \mathbf{X}$ does not have an inverse. However, a non-zero regularization parameter λ leads usually to a matrix $\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_p$ for which an inverse exists.

In Fig. 13.1, we show an example of the simulated data in Listing 13.1. Specifically, in Fig. 13.1a–c, we show the regression coefficients depending on λ , because the solution in Eq. 13.10 depends on the tuning parameter. On the top

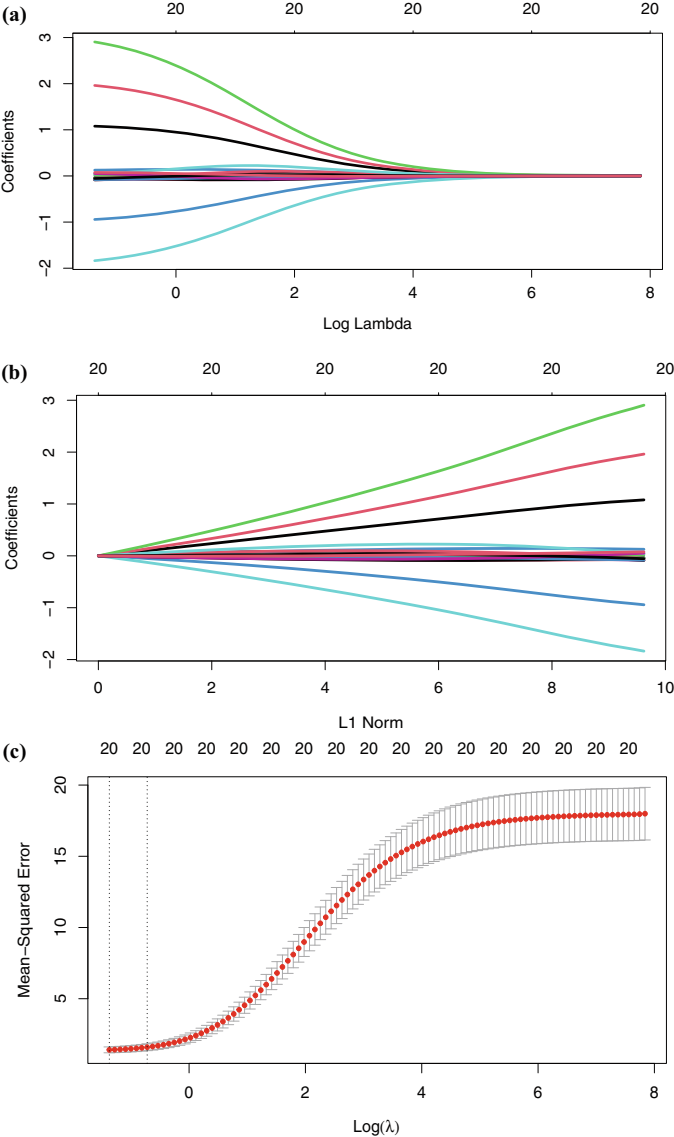


Fig. 13.1 Results of the ridge regression model from Listing 13.2. Coefficient paths for the ridge regression model against $\log(\lambda)$ (a), the L1-norm (b). (c) Mean-squared error against $\log(\lambda)$

of each figure, the numbers of non-zero regression coefficients are shown. One can see that for decreasing value of λ , the values of the regression coefficients are decreasing. This is the shrinkage effect of the tuning parameter. Furthermore, one can see that none of the coefficients becomes zero. Instead, all regression coefficients assume a small but non-zero value. These observations are typical for general results from ridge regression [223].

In Fig. 13.1c, we show results for the mean-squared error. Again, the numbers on top of the figures give the number of non-zero regression coefficients. The MSA can be used to identify an optimal value of λ (see the vertical dashed lines). As one can see, the mean-squared error suggests a smaller value of the tuning parameter that indeed shrinks the coefficients.

Overall, the advantages of a ridge regression is that it can reduce the variance at the price of an increased bias. This can improve the prediction accuracy of a model, which works best in situations where the OLS estimates have a high variance and $p < n$. A disadvantage of ridge regression is that it does not shrink the coefficients to zero. From this, it follows that ridge regression does not perform variable selection.

13.3.1 Example

Ridge regression can be performed using the package *glmnet* [173], available in R. Listing 13.2 was used to obtain the above results.

Listing 13.2: Ridge regression, visualization of results see Fig. 13.1

```
library(glmnet)
fit.Ridge <- glmnet(x=x.train, y=y.train, alpha=0)

plot(fit.Ridge, xvar = "lambda", lwd=2) # top figure

plot(fit.Ridge, xvar = "norm", lwd=2) # middle figure

cvfit.Ridge <- cv.glmnet(x=x.train, y=y.train, alpha=0)
plot(cvfit.Ridge) # bottom figure

cvfit.Ridge$lambda.min
cvfit.Ridge$lambda.1se
```

The last two lines in Listing 13.2 give estimates for $\hat{\lambda}_{min}$ and $\hat{\lambda}_{1se}$ (discussed in Sect. 13.5). Briefly, these values provide optimal estimates for λ . In order to use these, e.g., for predictions given a new data set, one needs to define a new ridge regression model by setting the “lambda” option. This is shown in Listing 13.3.

Listing 13.3: Ridge regression, making new predictions with the optimal model

```

fit.Ridge.opt <- glmnet(x=x.test, y=y.test, alpha=0,
  lambda=cvfit.Ridge$lambda.min)
pred <- predict(fit.Ridge.opt, s = cvfit.Ridge$lambda.min, newx =
  x.test)

RSS <- sum((pred - y.test) ^ 2)
MSE <- RSS/length(y.test)

```

13.4 Non-negative Garrote Regression

The next model we discuss is the non-negative garrote. Interestingly, the non-negative garrote has been mentioned as a motivation for the introduction of the LASSO [462]. For this reason, we discuss this model before the LASSO.

The non-negative garrote has been introduced by [55], and it is defined by

$$\hat{\beta} = \arg \min \left\{ \frac{1}{2n} \|y - Z\mathbf{d}\|_2^2 + \lambda \sum_{j=1}^p d_j \right\}, \quad (13.11)$$

for $\mathbf{d} = (d_1, \dots, d_p)^T$ with $d_j > 0$ for all j . Importantly, the regression model is formulated for the scaled variables $\mathbf{Z}$ given by $Z_j = X_j \hat{\beta}_j^{OLS}$. That means the model, first, estimates ordinary least squares parameter $\hat{\beta}_j^{OLS}$ for the unregularized regression (Eq. 11.27) and then performs in a second step a regularized regression for the scaled predictors $\mathbf{Z}$.

The estimates of the non-negative garrote can be expressed with the OLS regression coefficients and the regularization coefficients in the following way [516]:

$$\hat{\beta}_j^{NG}(\lambda) = d_j(\lambda) \hat{\beta}_j^{OLS} \quad (13.12)$$

Breiman showed that the non-negative garrote has consistently a lower prediction error than subset selection, and it is competitive with ridge regression except when the true model has many small non-zero coefficients. A disadvantage of the non-negative garrote is its explicit dependency on the OLS estimates [462].

13.5 LASSO

The LASSO (least absolute shrinkage and selection operator) has been made popular by R Tibshirani in 1996 [462], but it has been studied in the literature before, see, e.g., [155, 412]. The LASSO is a regression model that performs both

regularization and variable selection in order to enhance the prediction accuracy and interpretability of the regression model.

The LASSO estimate of $\hat{\beta}$ is given by

$$\hat{\beta} = \arg \min \left\{ \frac{1}{2n} \sum_{i=1}^n (y_i - \sum_j \beta_j x_{ij})^2 \right\} \quad (13.13)$$

$$\text{subject to: } \|\beta\|_1 \leq t. \quad (13.14)$$

Equation 13.13 is called the constrained form of the regression model. In Eq. 13.14, t is a tuning parameter (also called regularization parameter or penalty parameter) and $\|\beta\|_1$ the L1-norm (see Eq. 13.4).

One can show that Eq. 13.13 can be written in the Lagrange form given by

$$\hat{\beta} = \arg \min \left\{ \frac{1}{2n} \sum_{i=1}^n (y_i - \sum_j \beta_j x_{ij})^2 + \lambda \|\beta\|_1 \right\} \quad (13.15)$$

$$= \arg \min \left\{ \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \|\beta\|_1 \right\} \quad (13.16)$$

The relationship between both forms holds due to the duality and the KKT (Karush-Kuhn-Tucker) conditions. Furthermore, for every $t > 0$ there exists a $\lambda > 0$ such that both equations lead to the same solution [223].

In general, the LASSO lacks a closed form solution because the objective function is not differentiable. However, it is possible to obtain closed form solutions for the special case of an orthonormal design matrix.

In the LASSO regression model, Eq. 13.16, λ is a parameter that needs to be estimated. This is accomplished using cross-validation. Specifically, for each fold F_k , the mean-squared error is estimated by

$$e(\lambda)_k = \frac{1}{\#F_k} \sum_{j \in F_k} (y_j - \hat{y}_j)^2. \quad (13.17)$$

Here, $\#F_k$ is the number of samples in set F_k . Then the average over all K folds is taken leading to

$$CV(\lambda) = \frac{1}{K} \sum_{k=1}^K e(\lambda)_k. \quad (13.18)$$

This is called the cross-validation mean-squared error. To obtain an optimal λ from $CV(\lambda)$, two approaches are commonly used. The first approach estimates the λ that minimizes the function $CV(\lambda)$:

$$\hat{\lambda}_{min} = \arg \min CV(\lambda). \quad (13.19)$$

The second approach, first, estimates $\hat{\lambda}_{min}$ and then identifies the maximal λ that has a cross-validation MSE (mean-squared error) smaller than $CV(\hat{\lambda}_{min}) + SE(\hat{\lambda}_{min})$:

$$\hat{\lambda}_{1se} = \max_{CV(\lambda) \leq CV(\hat{\lambda}_{min}) + SE(\hat{\lambda}_{min})} \lambda \quad (13.20)$$

13.5.1 Example

In Listing 13.4, we present the R code used to analyze the simulated data. The results of this analysis are visualized in Fig. 13.2. The first two rows in Fig. 13.2 show coefficient paths for the LASSO regression model depending on $\log(\lambda)$ (top) and the L1-norm (middle). One can see that the five regression coefficients for the true predictors are nicely recovered, while the remaining (false) coefficients assume very small values (left-hand side) before they vanish. At the bottom of the figure, we show the mean-squared error depending on $\log(\lambda)$. The vertical dashed lines correspond to $\hat{\lambda}_{min}$ and $\hat{\lambda}_{1se}$ (see Listing 13.4 for numerical values corresponding to Fig. 13.2).

Listing 13.4: LASSO model, visualization of results see Fig. 13.2

```
fit.LASSO <- glmnet(x.train, y.train, family="gaussian",
  standardize=TRUE, alpha=1)
plot(fit.LASSO, xvar="lambda") #top

plot(fit.LASSO, xvar="norm") #middle

cvfit.LASSO <- cv.glmnet(x=x.train, y=y.train, alpha=1,
  type.measure="mse")
plot(cvfit.LASSO) #bottom

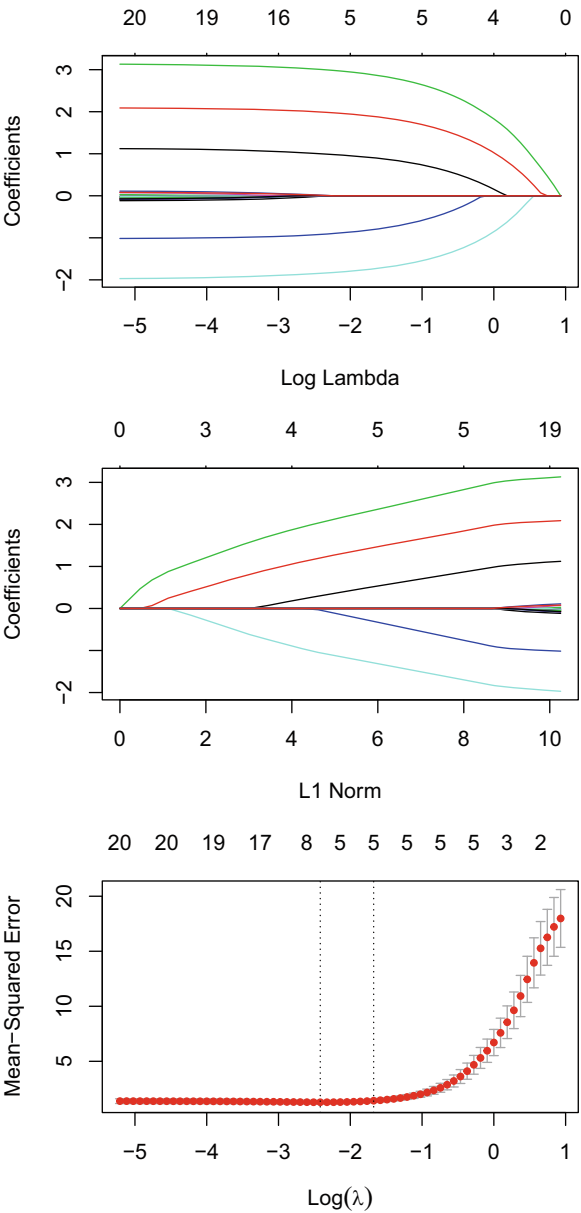
cvfit.LASSO$lambda.min # -> 0.08906576
cvfit.LASSO$lambda.1se # -> 0.1874749
min(cvfit.LASSO$cvm) # -> 1.280746 (minimal MSE)
beta.hat <- coef(cvfit.LASSO, s=cvfit.LASSO$lambda.1se) #
  estimated coefficients for optimal lambda
```

At the bottom of Listing 13.4, we show how to access important estimated entities including $\hat{\lambda}_{min}$, $\hat{\lambda}_{1se}$, as well as the regression coefficients. As one can see, the values of these estimates are needed, e.g., to obtain the regression coefficients of a particular model, as specified by $\hat{\lambda}_{1se}$.

13.5.2 Explanation of Variable Selection

From Fig. 13.2, one can see that decreasing values of λ lead to the shrinkage of the regression coefficients (see top and middle row in Fig. 13.2), and some of these

Fig. 13.2 Results for the LASSO from Listing 13.4. Shown are coefficient paths against $\log(\lambda)$ (top) and the L1-norm (middle). Bottom: Mean-squared error against $\log(\lambda)$



become even zero. In order to understand this behavior, we depict in Fig. 13.3 a two-dimensional LASSO (Panel A) and ridge regression (Panel B) model. The regularization term of each regression model is depicted in blue, corresponding to the diamond shape for the L1-norm and the circle for the L2-norm. The solution of the optimization problem is given by the intersection of the ellipsis and the boundary

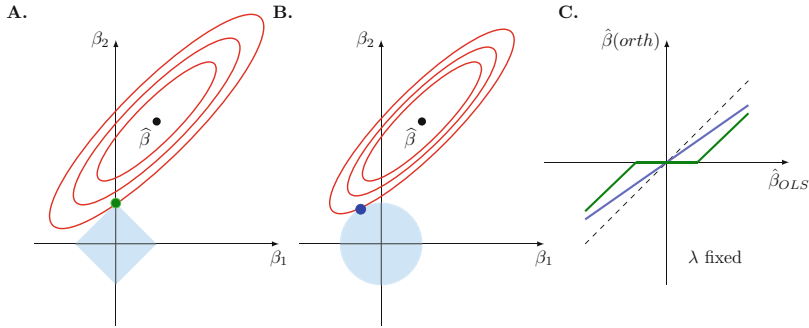


Fig. 13.3 Visualizing the difference between the L1-norm (a) used by the LASSO and L2-norm (b) used by ridge regression. (c) Solution for the orthonormal case

of the penalty shapes. These intersections are highlighted by a green point for the LASSO and a blue point for the ridge regression, respectively.

In order to shrink a coefficient to zero, an intersection needs to occur alongside the two coordinate axis. For the shown situation, this is only achieved using the LASSO but not the ridge regression. In general, the probability for a LASSO to shrink a coefficient to zero is much larger than for the ridge regression.

In order to understand this, it is helpful to look at the solution of the coefficients for the orthonormal case, because in this case the solution for the LASSO can be found analytically. The analytical solution is given by

$$\hat{\beta}_i^{LASSO}(\lambda; orth) = \text{sign}(\beta_i^{\hat{OLS}}) S(\beta_i^{\hat{OLS}}, \lambda) \quad (13.21)$$

Here, $S()$ is the soft-threshold operator defined as

$$S(\beta_i^{\hat{OLS}}, \lambda) = \begin{cases} \beta_i^{\hat{OLS}} - \lambda & , \quad \text{if } \beta_i^{\hat{OLS}} > \lambda; \\ 0 & , \quad \text{if } |\beta_i^{\hat{OLS}}| \leq \lambda; \\ \beta_i^{\hat{OLS}} + \lambda & , \quad \text{if } \beta_i^{\hat{OLS}} < -\lambda. \end{cases} \quad (13.22)$$

For the ridge regression, the orthonormal solution is given by

$$\hat{\beta}_i^{RR}(\lambda; orth) = \frac{\beta_i^{\hat{OLS}}}{1 + \lambda}. \quad (13.23)$$

In Fig. 13.3c, we show Eq. 13.21 (green) and Eq. 13.23 (blue). As a reference, we added the ordinary least square solution as a dashed diagonal line (black) because it is just the identity mapping:

$$\hat{\beta}_i^{OLS}(orth) = \beta_i^{\hat{OLS}} \quad (13.24)$$

As one can see, ridge regression leads to a change in the slope of the line and, hence, a shrinkage of the coefficient. However, it does not lead to a zero coefficient except for the point at the origin of the coordinate system. In contrast, LASSO shrinks the coefficient to zero for $|\beta_i^{\hat{OLS}}| \leq \lambda$.

13.5.3 Discussion

The key idea of the LASSO is to realize that the theoretically ideal penalty to achieve sparsity is the L0-norm (i.e., $\|\beta\|_0 = \text{\#non-zero elements}$; see Eq. 13.6), which is computationally intractable; however, this can be mimicked using the L1-norm, which makes the optimization problem convex [476].

There are three major differences between ridge regression and the LASSO:

1. The non-differentiable corners of the L1-ball produce sparse models for sufficiently large values of λ .
2. The lack of rotational invariance limits the use of the singular value theory.
3. The LASSO has no analytic solution, making both computational and theoretical results more difficult to obtain.

The first point implies that the LASSO is better than OLS for interpretation purpose. With a large number of independent variables, we often would like to identify a smaller subset of these variables that exhibit the strongest effects. The sparsity of the LASSO is mainly counted as an advantage due to a simpler interpretation, but it is important to highlight that the LASSO is not able to select more than n variables.

13.5.4 Limitations

There are a number of limitations for the LASSO estimator, which cause problems for variable selection in certain situations.

1. In the $p > n$ case, the LASSO selects at most n variables before it saturates. This could be a limiting factor if the true model consists of more than n variables.
2. The LASSO has no grouping property, that means it tends to select only one variable from a group of highly correlated variables.
3. In the $n > p$ case and high correlations between predictors, it has been observed that the prediction performance of the LASSO is inferior to the ridge regression.

13.6 Ridge Regression

Ridge regression was suggested by Frank and Friedman [171]. It minimizes the RSS subject to a constraint depending on a parameter q :

$$\hat{\beta} = \arg \min \left\{ \sum_{i=1}^n (y_i - \sum_j \beta_j x_{ij})^2 + \lambda \sum_{j=1}^p |\beta_j|^q \right\} \quad (13.25)$$

$$= \arg \min \left\{ \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{j=1}^p |\beta_j|^q \right\}. \quad (13.26)$$

The regularization term has the form of L $_q$ -norm, although q can assume all positive values, i.e., $q > 0$. For the special case $q = 2$, one obtains ridge regression and for $q = 1$ the LASSO. Although ridge regression was introduced in 1993 before the LASSO, the model hasn't been studied at that time. This justifies the LASSO as a new method because in [462] the authors presented a full analysis.

13.7 Dantzig Selector

Next, we discuss briefly the Dantzig selector [65]. This regression model was particularly introduced for the case where p is large ($p \gg n$), i.e., when we have many more parameters than observations.

The regression model solves the following problem:

$$\hat{\beta} = \arg \min \left\{ \|\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})\|_\infty + \lambda \|\boldsymbol{\beta}\|_1 \right\}. \quad (13.27)$$

Here, the L $_\infty$ norm is the maximum absolute value of the components of the argument. It is worth remarking that, in contrast with the LASSO, here the term $\mathbf{X}^T$ is added to the loss (residual sum) in Eq. 13.27. This term makes the solution rotation invariant.

An advantage of the Dantzig selector is that it is computationally simple, because technically, it can be reduced to a linear programming problem. This inspired the name of the method to pay tribute to George Dantzig for his seminal work on the simplex method for linear programming [222]. As a consequence of its computational efficiency, this regression model can be used for very high-dimensional data for which the LASSO becomes burdensome.

The disadvantages of the Dantzig selector are similar to the LASSO except that it can result in more than n non-zero coefficients in the case $p > n$ [126]. Also, the Dantzig selector is sensitive to outliers because the L $_\infty$ norm is very sensitive to outliers. This hampers the practical application of the model.

For a computational analysis of the Dantzig selector, the R package *flare* can be used [307].

13.8 Adaptive LASSO

The adaptive LASSO which was introduced in [528] is similar to the LASSO but with an oracle property. An oracle procedure is one that has the following oracle properties:

1. Consistency in variable selection.
2. Asymptotic normality.

Put simply, the oracle property means that a model performs as well as the true underlying model, if this was known [525]. Specifically, the first property above means that a model selects all non-zero coefficients with probability one, i.e., an oracle identifies the correct subset of true variables. The second property means that non-zero coefficients are estimated as in the true model, if this was known. Importantly, it has been shown that the adaptive LASSO is an oracle procedure but the LASSO is not [528].

The basic idea of the adaptive LASSO is to introduce weights for the penalty of each regression coefficient. Specifically, the adaptive LASSO is a two-step procedure. In the first step, a weight vector $\hat{\mathbf{w}}$ is estimated from OLS estimates of $\hat{\boldsymbol{\beta}}_{init}$, and a connection between both is given by

$$\hat{\mathbf{w}} = \frac{1}{|\hat{\boldsymbol{\beta}}_{init}^\gamma|} \quad (13.28)$$

Here, γ is a positive tuning parameter, i.e., $\gamma > 0$.

Second, for the weight vector, $\mathbf{w} = (w_1, \dots, w_p)^T$, the following weighted LASSO is formulated:

$$\hat{\boldsymbol{\beta}} = \arg \min \left\{ \frac{1}{2n} \sum_{i=1}^n (y_i - \sum_j \beta_j x_{ij})^2 + \lambda \sum_{j=1}^p w_j |\beta_j| \right\} \quad (13.29)$$

$$= \arg \min \left\{ \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{j=1}^p w_j |\beta_j| \right\}. \quad (13.30)$$

It can be shown that for certain data-dependent weight vectors, the adaptive LASSO has oracle properties. Typically, the values of $\hat{\boldsymbol{\beta}}_{init}$ are chosen according to the following cases:

- For the case where p is small ($p \ll n$): $\hat{\boldsymbol{\beta}}_{init} = \hat{\boldsymbol{\beta}}_{OLS}$.
- For the case where p is large ($p \gg n$): $\hat{\boldsymbol{\beta}}_{init} = \hat{\boldsymbol{\beta}}_{RR}$.

The adaptive LASSO penalty can be seen as an approximation to the L_q penalties, with $q = 1 - \gamma$. One advantage of the adaptive LASSO is that, for appropriate initial estimates, the criterion Eq. 13.29 is convex in β . Furthermore, if the initial estimates are consistent, it has been shown in [528] that the method recovers the true model under more general conditions than the LASSO.

13.8.1 Example

In Listing 13.4, we present an example in R using the adaptive LASSO to analyze the simulated data. In Fig. 13.4, we show the results for $\gamma = 1$. The figures show the coefficient paths depending on $\log(\lambda)$ (top) and the results for the mean-squared

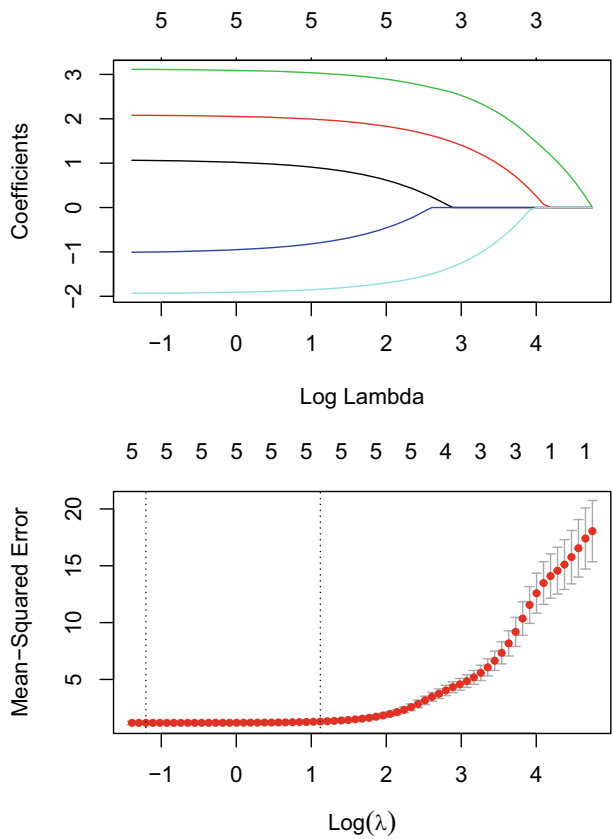


Fig. 13.4 Results for the adaptive LASSO using Listing 13.4. Top: Coefficient paths against $\log(\lambda)$. Bottom: Mean-squared error against $\log(\lambda)$

error (bottom). One can observe the shrinking and selecting property of the adaptive LASSO.

Listing 13.5: Adaptive LASSO model, visualization of results see Fig. 13.4

```
Ridge1_cv <- cv.glmnet(x = x.train, y = y.train,
  type.measure = "mse", nfold = 10, alpha = 0)
best_Ridge_coef <- as.numeric(coef(Ridge1_cv, s =
  Ridge1_cv$lambda.min))[-1]

aLASSO <- glmnet(x = x.train, y = y.train, alpha=1,
  penalty.factor = 1 / abs(best_Ridge_coef))
plot(aLASSO, xvar = "lambda") # top

aLASSO.cv <- cv.glmnet(x = x.train, y = y.train,
  type.measure = "mse",
  nfold = 10,
  alpha = 1,
  penalty.factor = 1 / abs(best_Ridge_coef),
  keep = TRUE)
plot(aLASSO.cv) # bottom
```

For the above analysis, we used $\gamma = 1$. However, γ is a tuning parameter that needs to be estimated from the data. We leave this as an exercise (see Exercise 3 below).

13.9 Elastic Net

The elastic net regression model was introduced in [529] to extend the LASSO, by improving some of its limitations, especially with respect to the variable selection. Importantly, the elastic net encourages a grouping effect, keeping strongly correlated predictors together in the model. In contrast, the LASSO tends to split such groups keeping only the strongest variable. Furthermore, the elastic net is particularly useful in cases where the number of predictors (p) in a data set is much larger than the number of observations (n). In such a case, the LASSO is not able to select more than n predictors, but the elastic net has this capability.

Assuming standardized predictors and response, the elastic net solves the following problem:

$$\hat{\beta} = \arg \min \left\{ \frac{1}{2N} \sum_{i=1}^N (y_i - \sum_j \beta_j x_{ij})^2 + \lambda P_{\alpha}(\beta) \right\} \quad (13.31)$$

$$= \arg \min \left\{ \frac{1}{2n} \|y - X\beta\|_2^2 + \lambda P_{\alpha}(\beta) \right\}; \quad (13.32)$$

$$P_{\alpha}(\beta) = \alpha \|\beta\|_2^2 + (1 - \alpha) \|\beta\|_1 \quad (13.33)$$

$$= \sum_{j=1}^p \alpha \beta_j^2 + (1 - \alpha) |\beta_j|. \quad (13.34)$$

Here, $P_\alpha(\beta)$ is the elastic net penalty [529]. $P_\alpha(\beta)$ is a combination between the ridge regression penalty, with $\alpha = 1$, and the LASSO penalty, with $\alpha = 0$. This form of penalty turned out to be particularly useful in the case $p > n$ or in situations where we have many (highly) correlated predictor variables.

In the correlated case, it is known that ridge regression shrinks the regression coefficients of the correlated predictors toward each other. In the extreme case of k identical predictors, each of them obtains the same estimates of the coefficients [174]. From theoretical considerations, it is further known that the ridge regression is optimal if there are many predictors, and all have non-zero coefficients. LASSO, on the other hand, is somewhat indifferent to very correlated predictors and will tend to pick one and ignore the rest.

Interestingly, it is known that the elastic net with $\alpha = \varepsilon$, for some very small $\varepsilon > 0$, performs similarly to the LASSO, but removes any degeneracies caused by the presence of correlations between the predictors [174]. More generally, the penalty family given by $P_\alpha(\beta)$ creates a non-trivial mixture between ridge regression and the LASSO. For a given λ , if we decrease α from 1 to 0, the number of regression coefficients, equal to zero, increases monotonically from 0 (full ridge regression model) to the sparsity of the LASSO solution. Here, sparsity refers to the fraction of regression coefficients equal to zero. For more detail, see Friedman et al. [174] where an efficient implementation of the elastic net penalty, for a variety of loss functions, was provided.

13.9.1 Example

In Listing 13.6, we present an example, in R, using the elastic net to analyze the simulated data. In order to obtain an elastic net model, one needs to set a value for the option “alpha.” For our analysis, we used $\alpha = 0.5$. The results of this analysis are visualized in Fig. 13.5. Here, the coefficient paths are shown depending on $\log(\lambda)$ (top) and the mean-squared error depending on $\log(\lambda)$ (bottom).

Listing 13.6: Elastic net model, visualization of results see Fig. 13.5

```
fit.LASSO <- glmnet(x=x.train, y=y.train, alpha=0.5)
plot(fit.LASSO, xvar = "lambda", lwd=2) # top

cvfit.LASSO <- cv.glmnet(x=x.train, y=y.train, alpha=0.5)
plot(cvfit.LASSO) # bottom
```

Since α is a parameter, one needs to optimize this value via model selection (see Exercise 4).

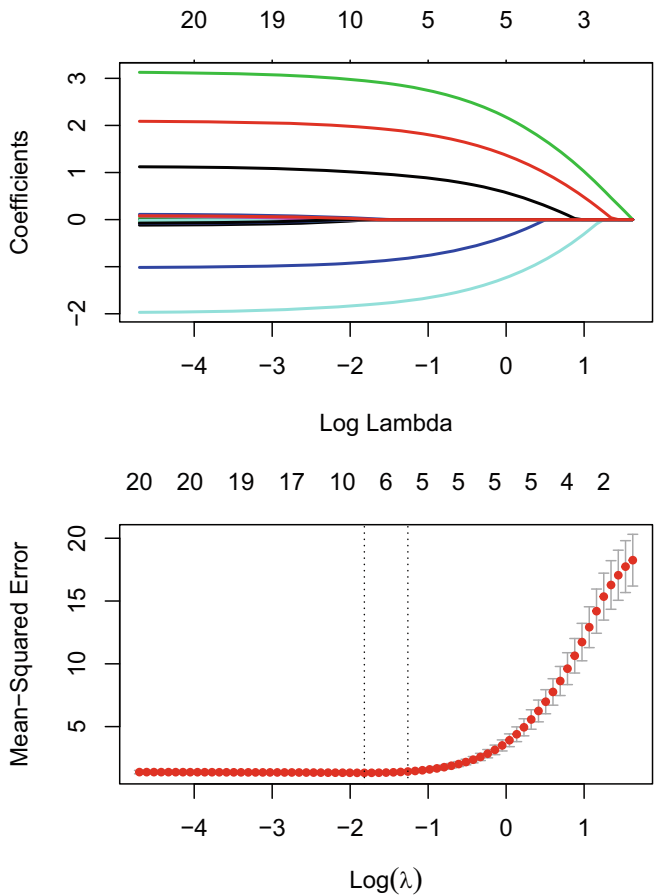


Fig. 13.5 Results for the elastic net from Listin 13.6. Top: Coefficient paths against $\log(\lambda)$ for $\alpha = 0.5$. Bottom: Mean-squared error against $\log(\lambda)$

13.9.2 Discussion

The elastic net has been introduced to counteract the drawbacks of the LASSO and ridge regression. The idea was to use a penalty for the elastic net which is based on a combined penalty of the LASSO and ridge regression. The penalty parameter α determines how much weight should be given to either the LASSO or ridge regression. An elastic net with $\alpha = 1.0$ performs a ridge regression, and an elastic net with $\alpha = 0$ performs the LASSO. Specifically, several studies [19, 471] showed that:

1. In the case of correlated predictors, the elastic net can result in lower mean-squared errors compared to ridge regression and the LASSO.

2. In the case of correlated predictors, the elastic net selects all predictors, whereas the LASSO selects one variable from a correlated group of variables but tends to ignore the remaining correlated variables.
3. In the case of uncorrelated predictors, the additional Ridge penalty brings little improvement.
4. The elastic net identifies correctly a larger number of variables compared to the LASSO (model selection).
5. The elastic net has often a lower false-positive rate compared to ridge regression.
6. In the case $p > n$, the elastic net can select more than n predictor variables, whereas the LASSO selects at most n .

The last point means that the elastic net is capable of performing group selection of variables, at least to a certain degree. To further improve this property, the group LASSO has been introduced (see below Sect. 13.10).

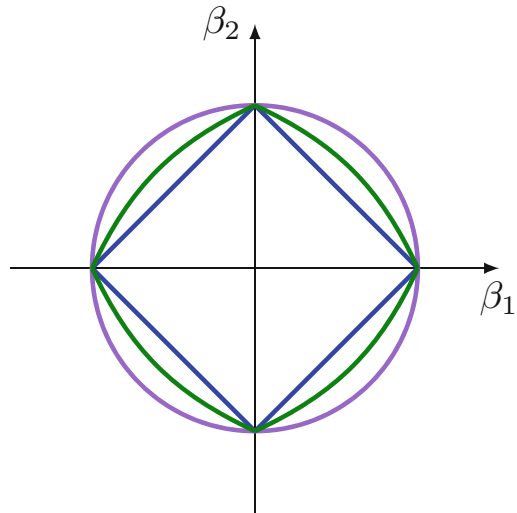
It can be shown that the elastic net penalty is a convex combination of the LASSO penalty and the Ridge penalty. Specifically, for all $\alpha \in (0, 1)$ the penalty function is strictly convex. In Fig. 13.6, we visualize the effect of the tuning parameter, α , on the regularization. As one can see, the elastic net penalty (in green) is located between the LASSO penalty (in blue) and the Ridge penalty (in purple).

The orthonormal solution of the elastic net is similar to the LASSO in Eq. 13.21. It is given by [529]

$$\hat{\beta}_i^{EN}(\lambda; orth) = \text{sign}(\beta_i^{\hat{OLS}}) \frac{S(\beta_i^{\hat{OLS}}, \lambda_1)}{1 + \lambda_2}, \quad (13.35)$$

with $S(\beta_i^{\hat{OLS}}, \lambda_1)$ defined as

Fig. 13.6 Visualization of the elastic net regularization (green) combining the L2-norm (purple) of ridge regression and the L1-norm (blue) of LASSO



$$S(\beta_i^{\hat{OLS}}, \lambda_1) = \begin{cases} \beta_i^{\hat{OLS}} - \lambda_1 & , \text{ if } \beta_i^{\hat{OLS}} > \lambda_1; \\ 0 & , \text{ if } |\beta_i^{\hat{OLS}}| \leq \lambda_1; \\ \beta_i^{\hat{OLS}} + \lambda_1 & , \text{ if } \beta_i^{\hat{OLS}} < -\lambda_1. \end{cases} \quad (13.36)$$

Here, the parameters λ_1 and λ_2 are connected to λ and α in Eq. 13.31 by

$$\alpha = \frac{\lambda_2}{\lambda_1 + \lambda_2}, \quad (13.37)$$

$$\lambda = \lambda_1 + \lambda_2, \quad (13.38)$$

resulting in the following alternative form of the elastic net:

$$\hat{\beta} = \arg \min \left\{ \frac{1}{2n} \|y - X\beta\|_2^2 + \lambda_2 \|\beta\|_2^2 + \lambda_1 \|\beta\|_1 \right\}. \quad (13.39)$$

In contrast with the LASSO in Eq. 13.21, only the slope of the line for $|\beta_i^{\hat{OLS}}| > \lambda_1$ is different due to the denominator $1 + \lambda_2$. That means the Ridge penalty, controlled by λ_2 , performs a second shrinkage effect on the coefficients. Hence, an elastic net performs a *double shrinkage* on the coefficients, one from the LASSO penalty and one from the Ridge penalty. Hence, from Eq. 13.35, one can also see the variable selection property of the elastic net, similar to the LASSO.

13.10 Group LASSO

The last modern regression model we are discussing is the group LASSO, introduced in [515]. The group LASSO is different from the other regression models because it focuses on groups of variables instead of individual variables. The reason for this is that there are many real-world application problems, e.g., pathways of genes, portfolios of stocks, or substage disorders of patients, which have substructures where a set of predictors forms a group that either should have non-zero or zero coefficients simultaneously.

The various forms of group LASSO penalty are designed for such situations.

Let's suppose that the p predictors are divided into G groups and p_g is the number of predictors in group $g \in \{1, \dots, G\}$. The matrix $X_g \in \mathbb{R}^{n \times p_g}$ represents the predictors corresponding to group g , and the corresponding regression coefficient vector is given by $\beta_g \in \mathbb{R}^{p_g}$.

The group LASSO solves the following convex optimization problem:

$$\hat{\beta} = \arg \min \left\{ \frac{1}{2n} \|y - \sum_{g=1}^G X_g \beta_g\|_2^2 + \lambda \sum_{g=1}^G \sqrt{p_g} \|\beta_g\|_2 \right\}. \quad (13.40)$$

Here, the term p_g accounts for the varying group sizes. If $p_g = 1$ for all groups g , then the group LASSO becomes the ordinary LASSO. If $p_g > 1$, the group LASSO works like the LASSO but on the group level, instead of the individual predictors.

13.10.1 Example

In Listing 13.7, we present an example, in R, using the group LASSO. The data analyzed are from the simulated data in Listing 13.1, and for the group labels we assumed $\{1, 1, 1, 2, 2, 3, 3, \dots, 3\}$ for the 20 predictors.

The results of this analysis are shown in Fig. 13.7. The top figure shows the coefficient paths depending on $\log(\lambda)$ and the bottom figure the mean-squared error depending on $\log(\lambda)$. The coefficient paths are colored according to the three groups (group 1, blue; group 2, purple; and group 2 magenta). As one can see, either all variables of a group are zero or not. From Fig. 13.7 (top), it is clear that first the coefficients of the variables 6–20 (in magenta) vanish and then the variable 4 and 5 (in purple) (see Listing 13.1 for this information).

Listing 13.7: Group LASSO model, visualization of results see Fig. 13.7

```
gr <- rep(3, 20)
gr[1:3] <- 1
gr[4:5] <- 2
l.seq <- seq(0.001, 3, 0.01)

m1 <- gglasso(x=x.train, y=y.train, group=gr, loss="ls",
              lambda=l.seq)
plot(m1, log.l=TRUE, lwd=3) # top

cv.glasso <- cv.gglasso(x=X, y=Y, group=gr, loss="ls",
                        pred.loss="L2", lambda.factor=0.05, nfolds=5, lambda=l.seq)
plot(cv.glasso) # bottom
```

According to the results from the mean-squared error in Fig. 13.7 (bottom), the optimal solution involves only two groups, consisting of the first five variables.

We would like to remark that the group information for the variables may not always be easy to obtain or there may be even alternative groupings. Furthermore, one may consider the finding of the groups as a model selection problem. Unfortunately, even for a moderate number of variables, this can quickly become computationally challenging if one wants to do an exhaustive search.

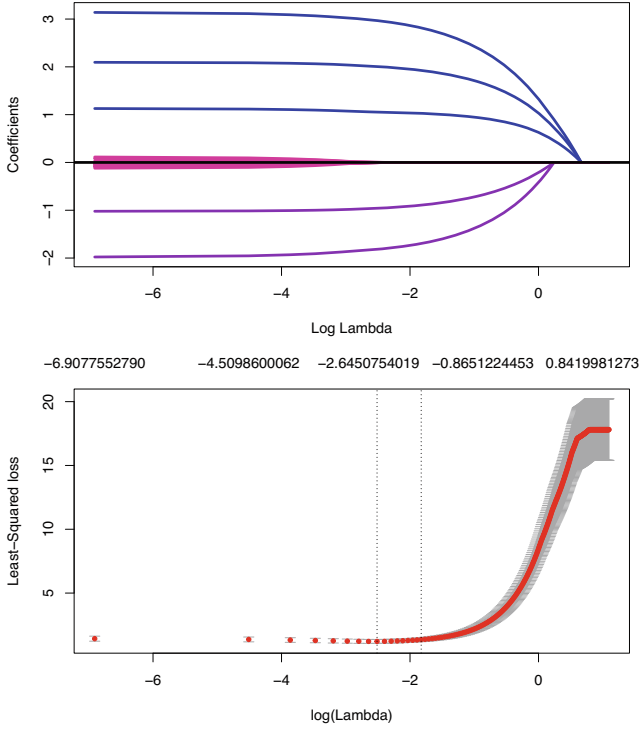


Fig. 13.7 Group LASSO. Top: Coefficient paths against $\log(\lambda)$. Bottom: Mean-squared error against $\log(\lambda)$

13.10.2 Remarks

1. The group LASSO has either zero coefficients of all members of a group or non-zero coefficients.
2. The group LASSO cannot achieve sparsity within a group.
3. The groups need to be predefined, i.e., the regression model does not provide a direct mechanism to obtain the grouping.
4. The groups are mutually exclusive (non-overlapping).

Finally, we just want to briefly mention that to overcome the limitation of the group LASSO to obtain sparsity within a group (point (2)), the sparse group LASSO has been introduced in [436], and the corresponding optimization problem writes

$$\hat{\beta} = \arg \min \left\{ \frac{1}{2n} \left\| \mathbf{y} - \sum_{g=1}^G \mathbf{X}_g \boldsymbol{\beta}_g \right\|_2^2 + (1 - \alpha) \lambda \sum_{g=1}^G \sqrt{p_g} \|\boldsymbol{\beta}_g\|_2 + \alpha \lambda \|\boldsymbol{\beta}\|_1 \right\}. \quad (13.41)$$

Table 13.1 Summary of key features of the regularized regression models

Method	Analytical solution	Variable selection	Can select $> n$	Grouping	Oracle
Ridge regression	Yes	No	Yes	Yes	No
Non-negative garrote	No	Yes	No	No	No
LASSO	No	Yes	No	No	No
Dantzig selector	No	Yes	Yes	No	No
Adaptive LASSO	No	Yes	No	No	Yes
Elastic net	No	Yes	Yes	Yes	No
Group LASSO	No	Yes	Yes	Yes	No

For $\alpha \in [0, 1]$, this is a convex optimization problem combining the group LASSO penalty (with $\alpha = 0$) with the LASSO penalty (with $\alpha = 1$). Here, $\beta \in \mathbb{R}^p$ is the complete coefficient vector.

13.11 Discussion

The modern regression models discussed in this chapter extend OLS regression. In contrast to the OLS regression and ridge regression, all of these models are computational in nature because the solution to the various regularizations can only be found by means of numerical approaches.

In Table 13.1, we summarize key features of these regression models. A common feature of all the extensions of OLS regression and ridge regression is that these models perform variable selection (coefficient shrinkage to zero). This allows to obtain interpretable models because the smaller the number of variables in a model, the easier it is to find plausible explanations. Considering this, the most satisfying method is the adaptive LASSO because it possesses the oracle property, enabling (under certain conditions) to identify only the coefficients that are non-zero in the true model.

In general, one considers data as high-dimensional if either (I) p is large or (II) $p > n$ [19, 265, 330]. Case (I) can be handled by all regression models, including the OLS regression. However, case (II) is more difficult because it may require to select more variables than available samples. Only ridge regression, Dantzig selector, elastic net, and the group LASSO are capable of this, while the elastic net is particularly more suitable for this situation.

Finally, the grouping of variables is useful, e.g., in cases where variables are highly correlated with each other. Again ridge regression, the elastic net, and the group LASSO have this property, and the latter has been specifically introduced to deal with this problem.

In Fig. 13.8, we show a numerical comparison of three models. The underlying data are again for 20 uncorrelated covariates where only 5 contribute to the response variable. Specifically, Fig. 13.8 shows the MSE depending on the sample size of

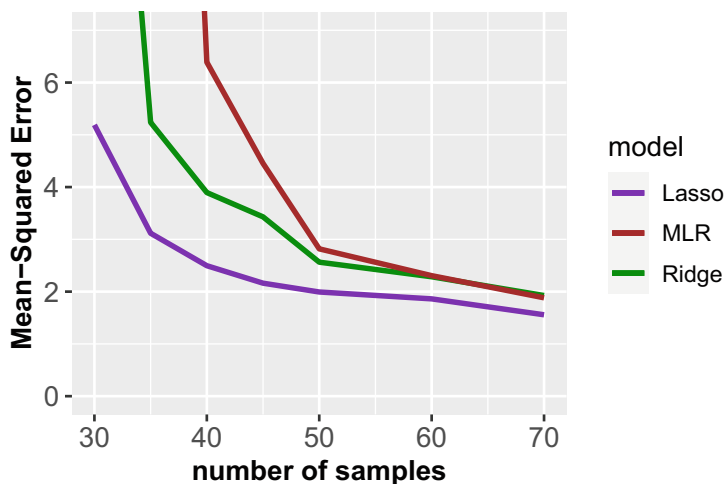


Fig. 13.8 Comparison of the MSE for three different models: LASSO, ridge regression and multiple linear regression (MLR)

the training data. The results are averaged over 100 independent data sets. As one can see, with an increasing sample size, the distance between the three models becomes smaller, and even an unregularized multiple linear regression (MLR) model performs satisfactorily. On the other hand, for smaller sample sizes, the advantage of a regularization becomes apparent.

This example demonstrates that the advantage of a model is data-dependent. Therefore, this needs to be investigated on a case by case basis. However, this makes a high-dimensional regression analysis non-trivial requiring insights from the analyst.

13.12 Summary

In this chapter, we discussed regularized regression models. Over the years, many different regularization models have been introduced, where each addresses a particular problem; hence, none of the methods dominates the others, and each has specific strengths and weaknesses. In this chapter, we discussed ridge regression, non-negative garrote regression, LASSO, ridge regression, Dantzig selector, adaptive LASSO, elastic net, and the group LASSO. Especially, the LASSO is a very popular model that can be found frequently in many applications, ranging from biology to psychology.

Learning Outcome 13: Regularization

Regularization is a mathematical concept that modifies the optimization functional of a regression model. This can lead to the shrinkage of regression coefficients, which can even vanish.

Regularization is a powerful framework to influence the optimization of regression coefficients. We have seen that depending on the mathematical formulation, different models can be obtained. It is interesting to note that when regression coefficients are shrunk to zero, regularization performs model selection on the number of regression coefficients.

13.13 Exercises

1. Perform a regression analysis with the ridge regression model for the simulated data in Listing 13.1.
 - Reproduce the results in Fig. 13.1.
 - Use $\hat{\lambda}_{min}$ to define an optimal model, and make prediction for the testing and training data. Compare and discuss the results.
 - Repeat this analysis for $\hat{\lambda}_{1se}$.
2. Perform a regression analysis with the LASSO model for the simulated data in Listing 13.1.
 - Reproduce the results in Fig. 13.2.
 - Why do the values of $\hat{\lambda}_{min}$ and $\hat{\lambda}_{1se}$ change when the analysis is repeated? Hint: See Chap. 4 and our discussion about k-fold CV.
3. Perform a regression analysis with the adaptive LASSO model for the simulated data in Listing 13.1.
 - Reproduce the results in Fig. 13.4.
 - Estimate the optimal value of γ in Eq. 13.28. Hint: Formulate the analysis as a model selection problem; see Chap. 12.
4. Perform a regression analysis with the elastic net model for the simulated data in Listing 13.1.
 - Reproduce the results in Fig. 13.5.
 - Estimate the optimal value of α in Eq. 13.34. Hint: Formulate the analysis as a model selection problem; see Chap. 12.

Chapter 14

Deep Learning



14.1 Introduction

Deep learning models are new estimation models in artificial intelligence (AI). Recent breakthrough results in image analysis and speech recognition have generated a massive interest in this field because applications in many other domains, generating big data, seem possible. On a downside, the mathematical and computational methodology underlying deep learning models is very challenging especially for interdisciplinary scientists.

In general, deep learning (DL) describes a family of learning algorithms rather than a single method that can be used to learn complex prediction models, e.g., multilayer neural networks with many hidden units [298]. Importantly, DL received much attention [236] in recent years, and it has been successfully applied to several application problems. For instance, a deep learning method sets the record for the classification of handwritten digits of the MNIST data set with an error rate of 0.21% [487]. Further application areas include image recognition [289, 298], speech recognition [208], natural language understanding [413], acoustic modeling [338], and computational biology [6, 305, 440, 441, 523].

Interestingly, models of artificial neural networks have been used since about the 1950s [404]; however, the current wave of deep learning neural networks started around 2006 [236]. A common characteristic of the many variants of supervised and unsupervised deep learning models is that these models learn many layers of hidden neurons, e.g., using a restricted Boltzmann machine (RBM) in combination with backpropagation and error gradients of the stochastic gradient descent [400]. Due to the heterogeneity of deep learning approaches, a comprehensive discussion is very challenging, and for this reason, many introductions aim usually at dedicated sub-topics. For instance, a bird's eye view without detailed explanations can be found in [298], whereas a historical summary with many detailed references was provided in [420]. In addition, reviews are available for various application domains, including

image analysis [397, 429], speech recognition [514], natural language processing [513], and biomedicine [66].

This chapter is organized as follows. In Sect. 14.2, we discuss major architectures, distinguishing classical neural networks with deep neural networks. Then, we discuss a number of deep neural networks in detail: deep feedforward neural networks (in Sect. 14.3), convolutional neural networks (in Sect. 14.4), deep belief networks (in Sect. 14.5), autoencoders (in Sect. 14.6), and long short-term memory networks (in Sect. 14.7). In Sect. 14.8, we provide a discussion of important issues when learning neural networks models. Finally, this chapter finishes with a summary (Sect. 14.9). Throughout these sections, we provide a number of numerical examples using R.

14.2 Architectures of Classical Neural Networks

Historically, artificial neural networks (ANNs) are mathematical models that have been motivated by the functioning of the brain. However, the models we discuss in the following do not aim at providing biologically realistic models. Instead, the purpose of these models is to analyze data.

14.2.1 Mathematical Model of an Artificial Neuron

The basic entity of any neural network is the model of a neuron. In Fig. 14.1a we show such a model of an artificial neuron.

The basic idea of a neuron model is that an input, $\mathbf{x}$, weighted by $\mathbf{w}$, together with a bias, b , are summarized. The bias, b , is a scalar value, whereas the input $\mathbf{x}$ and the

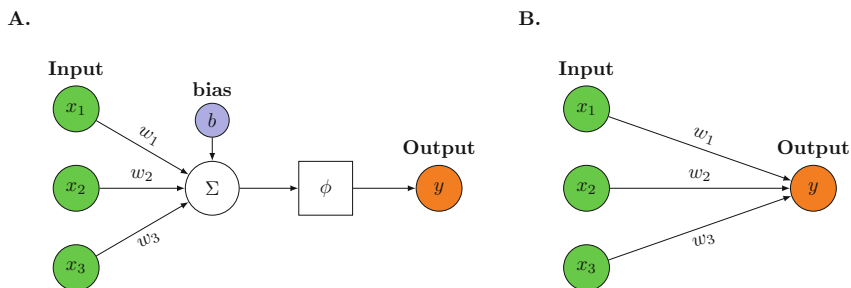


Fig. 14.1 (a) Representation of a mathematical artificial neuron model. The input to the neuron is summed up and filtered by activation function ϕ (for examples see Table 14.1). (b) Simplified Representation of an artificial neuron model. Only the key elements are depicted, i.e., the input, the output, and the weights

Table 14.1 An overview of frequently used activation functions for neuron models

Activation function	$\phi(x)$	$\phi'(x)$	Values
Hyperbolic tangent	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - \phi(x)^2$	$(-1, 1)$
Sigmoid	$S(x) = \frac{1}{1+e^{-x}}$	$\phi(x)(1 - \phi(x))$	$(0, 1)$
ReLU	$R(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Heaviside function	$H(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$\delta(x)$	$[0, 1]$
Signum function	$\operatorname{sgn}(x) = \begin{cases} -1 & \text{for } x < 0 \\ 0 & \text{for } x = 0 \\ 1 & \text{for } x > 0 \end{cases}$	$2\delta(x)$	$[-1, 1]$
Softmax	$y_i = \frac{e^{x_i}}{\sum_j^n e^{x_j}}$	$\frac{\partial y_i}{\partial j} = y_i(\delta_{ij} - y_j)$	$(0, 1)$

weights $\mathbf{w}$ are vector valued, i.e., $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^n$, with $n \in \mathbb{N}$ corresponding to the dimension of the input. Note that the bias term is not always present, but it is sometimes omitted. The sum of these terms, i.e., $z = \mathbf{w}^T \mathbf{x} + b$ then forms the argument of an activation function, ϕ , resulting in the output of the neuron model:

$$y = \phi(z) = \phi(\mathbf{w}^T \mathbf{x} + b). \quad (14.1)$$

Considering only the argument of ϕ , one obtains a linear discriminant function [495].

The activation function, ϕ (also known as unit function or transfer function), performs a nonlinear transformation of z . In Table 14.1, we give an overview of frequently used activation functions.

The ReLU activation function, also called rectified linear unit or rectifier [352], is the most popular activation function for deep neural networks. Another useful activation function is the softmax function [296], given by

$$y_i = \frac{e^{x_i}}{\sum_j^n e^{x_j}}. \quad (14.2)$$

The softmax maps an n -dimensional vector $\mathbf{x}$ into an n -dimensional vector $\mathbf{y}$ with the property $\sum_i y_i = 1$. Hence, the components of $\mathbf{y}$ represent probabilities for each of the n elements. The softmax is often used in the final layer of a network. If the Heaviside step function is used as activation function, the neuron model is known as *perceptron* [404].

In general, the neuron model depicted in Fig. 14.1a can be described more simplistically as shown in Fig. 14.1b, where merely the input and output parts are depicted.

14.2.2 Feedforward Neural Networks

In order to build neural networks (NNs), the neurons need to be connected with each other. The simplest architecture of a NN is the feedforward structure. In Fig. 14.2a and b, we show examples for a shallow and a deep architecture (discussed in detail in Sect. 14.3).

In general, the depth of a network denotes the number of nonlinear transformations between the separating layers, whereas the dimensionality of a hidden layer, i.e., the number of hidden neurons, is called its width. For instance, the shallow architecture in Fig. 14.2a has a depth of 2, whereas the architecture Fig. 14.2b has a depth of 4 (i.e., total number of layers minus one (the input layer)). The required value for the depth to call a feedforward neural network (FFNN) architecture deep is debatable, but architectures with more than two hidden layers are commonly considered as deep [30].

A feedforward neural network, also called a multilayer perceptron (MLP), can use linear or nonlinear activation functions [202]. Importantly, there are no cycles in the NN that would allow a direct feedback. Equation 14.3 defines how the output

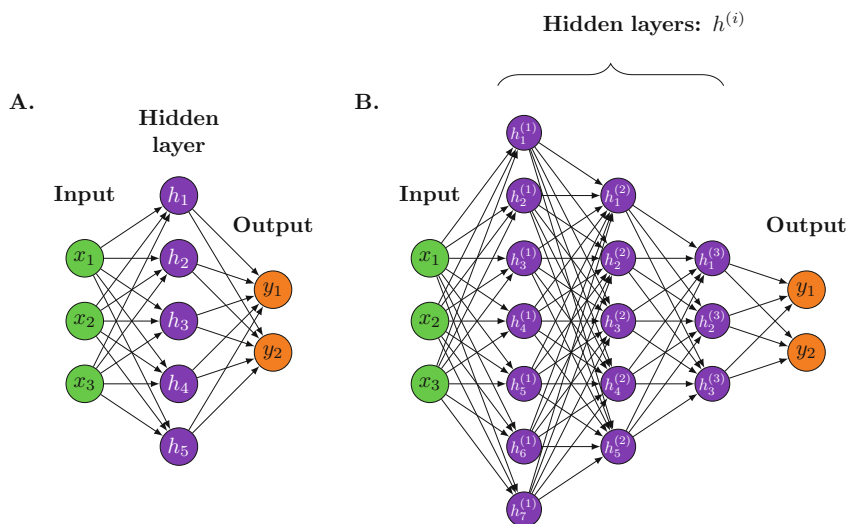


Fig. 14.2 Two examples of feedforward neural networks (FFNN). (a) A shallow FFNN. (b) A deep feedforward neural network (D-FFNN) with 3 hidden layers (see Sect. 14.3 for details about D-FFNN)

of a MLP is obtained from the input [495]:

$$f(\mathbf{x}) = \varphi^{(2)}(W^{(2)}\varphi^{(1)}(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}). \quad (14.3)$$

Equation 14.3 is the discriminant function of the neural network [495]. To find the optimal parameters of the model, one needs to define a learning rule. A common approach is to define an error function (or cost function) together with an optimization algorithm to find the optimal parameters by minimizing the error for the training data.

14.2.3 Recurrent Neural Networks

The family of recurrent neural network (RNN) models has two subclasses that can be distinguished based on their signal processing behavior. The first consists of finite impulse recurrent networks (FRNs) and the second infinite impulse recurrent networks (IIRNs). The difference is that a FRN is given by a directed acyclic graph (DAG) that can be unrolled in time and replaced with a feedforward neural network, whereas an IIRN is a directed cyclic graph (DCG) for which such an unrolling is not possible.

14.2.3.1 Hopfield Networks

A Hopfield network (HN) [248] is an example of a FRN. An HN is defined as a fully connected network consisting of McCulloch-Pitts neurons. A McCulloch-Pitts neuron is a binary model with an activation function given by

$$s = \text{sgn}(x) = \begin{cases} +1 & \text{for } x \geq 0, \\ -1 & \text{for } x < 0. \end{cases} \quad (14.4)$$

The activity of the neurons x_i , i.e.,

$$x_i = \text{sgn}\left(\sum_{j=1}^N w_{ij}x_j - \theta_i\right), \quad (14.5)$$

is updated either synchronously or asynchronously. To be precise, x_j refers to x_j^t and x_i to x_i^{t+1} (time progression).

Hopfield networks have been introduced to serve as model of a content-addressable (“associative”) memory, i.e., for storing patterns. In this case, it has been shown that the weights are given by

$$w_{ij} = \sum_{k=1}^P t_i(k)t_j(k), \quad (14.6)$$

where P is the number of patterns, $t(k)$ is the k th pattern, and $t_i(k)$ its i th component. From Eq. 14.6, one can see that the weights are symmetrical. An interesting question, in this context, is “what is the maximal value of P or P/N ?” The ratio P/N is called the network capacity (here, N is the total number of patterns). In [231], it was shown that the network capacity is ≈ 0.138 .

It is interesting to note that the neurons in a Hopfield network cannot be distinguished as input neurons, hidden neurons, and output neurons because at the beginning every neuron is an input neuron, during the processing, every neuron is a hidden neuron and at the end every neuron is an output neuron.

14.2.3.2 Boltzmann Machine

A Boltzmann machine [235] can be described as a *noisy* Hopfield network because it uses a probabilistic activation function

$$p(s_i = 1) = \frac{1}{1 + \exp(-x_i)}, \quad (14.7)$$

where x_i is obtained as in Eq. 14.5. This model is important because it is one of the first neural networks that uses hidden units (latent variables). To learn the weights, the contrastive divergence algorithm (see Algorithm 14.11) can be used to train Boltzmann machines. Put simply, Boltzmann machines are neural networks consisting of two layers—a visible layer and a hidden layer. Each edge between the two layers is undirected, implying that information can flow in a bidirectional way. The whole network is fully connected, which means that each neuron in the network is connected to all other neurons via undirected edge (see Fig. 14.10a and b).

14.2.4 Overview of General Network Architectures

There are many different network architectures that can be used as deep learning models. In Table 14.2, we show an overview of some of the most popular deep learning models, which can be found in the literature [30, 298].

It is interesting to note that some of the models in Table 14.2 are composed by other networks. For instance, CDBNs are based on RBMs and CNNs [301]; DBMs are based on RBMs [410]; DBNs are based on RBMs and MLPs; dAEs are stochastic autoencoders that can be stacked on top of each other to build stacked denoising autoencoders (SdAEs).

Table 14.2 An overview of some popular deep learning models, available learning algorithms (unsupervised, supervised) and software implementations in R or python

Deep learning model	Unsupervised	Supervised	Software
Autoencoder	✓		Keras [77], R: dimRed [288], h2o [64], RcppDL [287]
Convolutional deep belief network (CDBN)	✓	✓	R & python: TensorFlow [2], Keras [77], h2o [64]
Convolutional neural network (CNN)	✓	✓	R & python: Keras [77] MXNet [73], Tensorflow [2], h2o [64], fastai (python)[252]
Deep belief network (DBN)	✓	✓	RcppDL (R) [287], python: Caffee [264], Theano [459], Pytorch [374], R & python: TensorFlow [2], h2o [64]
Deep Boltzmann machine (DBM)		✓	python: boltzmann-machines [49], pydbm [75]
Denoising autoencoder (dA)	✓		Tensorflow (R, python) [2], Keras (R, python) [77], RcppDL (R) [287]
Long short-term memory (LSTM)		✓	rnn (R) [392], OSTSC (R) [110], Keras (R and python) [77], Lasagne (python) [107], BigDL (python) [90], Caffee (python) [264]
Multilayer perceptron (MLP)		✓	SparkR (R) [479], RSNNs (R) [38], keras (R and python) [77], sklearn (python) [381], tensorflow (R and python) [2]
Recurrent neural network (RNN)		✓	RSNNs (R) [38], rnn (R) [392], keras (R and python) [77]
Restricted Boltzmann machine (RBM)	✓	✓	RcppDL (R) [287], deepnet (R) [403], pydbm (python) [75], sklearn (python) [75], Pylearn2 [200], TheanoLM [153]

In the following sections, we discuss the major core architectures of deep learning models in detail. Specifically, we discuss deep feedforward neural networks (D-FFNN), convolutional neural networks (CNNs), deep belief networks (DBNs), autoencoders (AEs), and long short-term memory networks (LSTMs).

14.3 Deep Feedforward Neural Networks

It can be proven that a feedforward neural network (FFNN), with one hidden layer and a finite number of neurons in the hidden layer, can approximate any continuous function on a compact subset of $\mathbb{R}^n$ [249]. This is called the *universal approximation theorem*. The reason for using a FFNN with more than one hidden layer is that the universal approximation theorem does not provide information on how to learn such a network (how to estimate its parameters), which is a difficult

problem. A related issue that contributes to the difficulty of learning such networks is that their width can become exponentially large. Interestingly, the universal approximation theorem can also be proven for FFNN with many hidden layers and a bounded number of hidden neurons [317] for which learning algorithms have been found. Hence, D-FFNNs are used instead of shallow FFNNs for practical reasons of learnability.

Formally, the idea of approximating an unknown function f^* can be written as follows:

$$y = f^*(x) \approx f(x, w) \approx \phi(x^T w). \quad (14.8)$$

Here, f is a function from a specific family that depends on the parameters θ , and ϕ is a nonlinear activation function for one layer. For many hidden layers, ϕ has the form

$$\phi = \phi^{(n)}(\dots \phi^{(2)}(\phi^{(1)}(x)) \dots). \quad (14.9)$$

Instead of *guessing* the correct family of functions from which f should be chosen, D-FFNNs learn this function by approximating it via ϕ , which itself is approximated by the n hidden layers.

Practically, the learning of the parameters of a D-FFNN (see Fig. 14.2b) can be accomplished with the *backpropagation algorithm*, although, for computational efficiency, nowadays the *stochastic gradient descent* is used [52]. The stochastic gradient descent calculates a gradient for a set of randomly chosen training samples (batch) and updates the parameters for this batch sequentially. This results in a faster learning. A drawback is an increase in imprecision. However, for data sets with a large number of samples, the speed advantage outweighs this drawback.

14.3.1 Example: Deep Feedforward Neural Networks

In the following, we provide two examples of deep feedforward neural network model implementations. The first example implements a D-FFNN from scratch, using R, while the second example uses the *Keras* package. The first example is for understanding the functioning of D-FFNN models, which mainly performs backward and forward propagation and readjusts the weights of the neurons. In contrast, the second example shows that the *Keras* package allows to simplify such an analysis.

For the first example, we generate simulated data from a mixture of normal distributions, giving data for a two-class classification problem. Listing 14.1 generates these data, and a visualization of the resulting data is shown in Fig. 14.3 (top).

Listing 14.1: Simulating classification data for the DFNN in Listing 14.2

```

#Generating simulated data with two input features for two
  classes from normal distributions.

## Class 1
x1 <- cbind(rnorm(200,2), rnorm(200,2))
x2 <- cbind(rnorm(200,-2), rnorm(200,-2))

## class 2
x3 <- cbind(rnorm(200,-2), rnorm(200,2))
x4 <- cbind(rnorm(200,2), rnorm(200,-2))
## Combining data
x <- rbind(rbind(x1, x2), rbind(x3,x4))

## Creating labels
y <- matrix(0, nrow=nrow(x), ncol=2)
y[1:400,1]<- 1
y[401:800,2]<- 2

## Visualizing simulated data of two labels
df <- data.frame(cbind(x,y[,1]))
df$X3 <- as.factor(df$X3)
ggplot(df, aes(x=X1, y=X2, color=X3)) + geom_point()

## Selecting training and testing data
smp <- sample(nrow(x), 200 )
train <- x[-smp, ]
train_y <- y[-smp,]
test_x<- x[smp,]
test_y<- y[smp,]

## Training the model
## creating a dnn model shown in Figure 14.6 A containing two
  input features, and 6, 4, 3 neurons in three middle layers
  and two output nodes.

obj <- dnn.archx(train,train_y, lyr=c(6,4,3),fn="softmax")

## number of iterations
itr<- 10000
## learning rate
lr<- .01

er <- c()
for(i in 1:itr){
  obj <- dnn.ffwd(obj)

  obj <- dnn.backprop(obj, lr=lr)
  er <- c(er, errorf(obj))
}
## error visualization
edf <- data.frame(error=er, iterations=1:itr)
ggplot(edf, aes(x = iterations, y = error, color="red")) +
  geom_point()

## predicting test results
zz <- apply(test_y, 1,which.max)
zz1 <- predictf(obj,test_x)
table(zz,zz1)

```

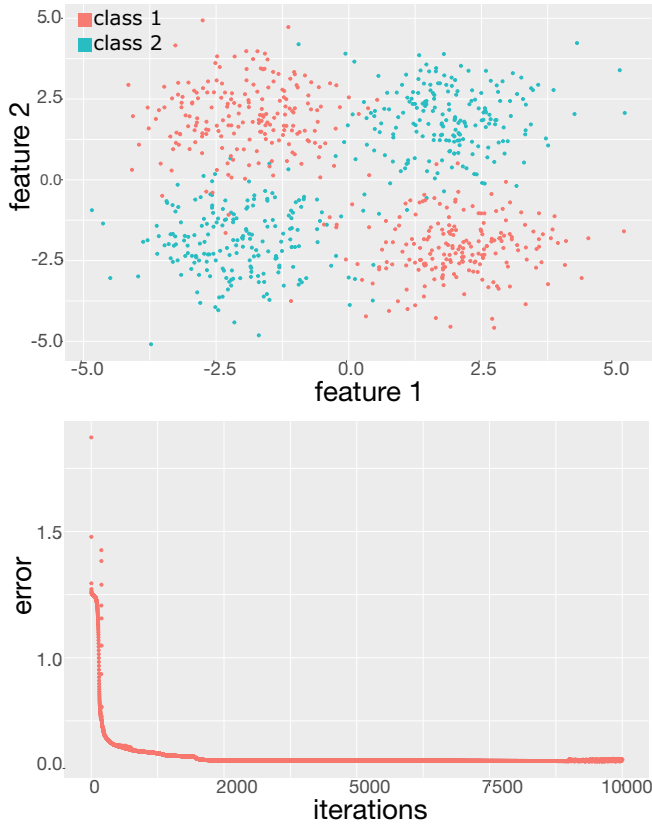


Fig. 14.3 A visualization of the simulated data for a two-class classification problem generated in Listing 14.1 (top). Bottom: The training error of the D-FFNN in Listing 14.2 is shown in dependence on the number of iterations for training the model to optimize the weights of the neural network

Implementing a D-FFNN from scratch requires three main functions, implemented in Listing 14.2. The function `dnn.arch()` allows an initialization of the weights, the user-defined layers, and the number of neurons in each layer based on input data and then generates a list object containing input data (X), weights of the hidden layer (w_{hi}), output o_{hi} , and the data output y . For input and each hidden layer, we add also a “bias” neuron. The functions `dnn.ffwd()` and `dnn.backprop()` are for feedforward and backpropagation operations. The function `dnn.ffwd` uses the list object, generated using the function `dnn.arch()`, to perform matrix multiplications between the previous layer’s output and the neurons’ weights sequentially, until it generates the output for the last layer of the neural network. The function `dnn.backprop` takes the updated list object as input from the forward operation. It calculates the mean square error (loss function) between output and class labels, and then it calculates derivatives in the feedforward network to optimize the weights of the neurons of each layer using a gradient descent approach.

Listing 14.2: A DFNN implementation in R

```

#This function create the initial architecture and initialize
random weights of DNN based on users' input and output data.
dnn.arch <- function(inp=c(10,3), y=NULL, lyr=c(2,3), oup=1,
  fn="relu", fno="softmax"){
  if(is.null(y)){
    y <- rep(1, inp[1])
    oup <- ncol(y)
  }
  mlyr <- list()
  if(class(inp)[1]=="matrix"){
    minp<- inp

    minp <- cbind(rep(1, nrow(inp)), inp)
    inp <- dim(minp)
    oup <- ncol(y)
    mlyr[["input"]]<- list(op=minp, fn=NULL, wt=NULL)
  }
  cnt=1
  print(lyr)
  for(i in 1:length(lyr)){
    print(i)
    if(i==1){
      nr=ncol(mlyr[[length(mlyr)]]$op)
      nc=lyr[i]
      tmpwt <- runif(nr*nc, min=-1, max=1)
      mtmp <- matrix(tmpwt, nrow=nr, ncol=nc)
    }else{
      nr=lyr[i-1]+1
      nc=lyr[i]
      tmpwt <- runif(nr*nc, min=-1, max=1)
      mtmp <- matrix(tmpwt, nrow=nr, ncol=nc)
    }
    #opx <- mlyr[[length(mlyr)]]$op%%mtmp
    mlyr[[paste0("middle_",cnt)]] <- list(wt=mtmp,fn=fn, op=
      NULL)
    cnt <- cnt+1
  }
  nc <- ncol(y)
  nr <- lyr[i]+1
  tmpwt <- runif(nc*nr, min=-1, max=1)
  moup <- matrix(tmpwt, ncol=nc, nrow=nr)
  mlyr[["output_layer"]] <- list(wt=moup, fn=fno, op=NULL)
  y <- as.matrix(y)
  mlyr <- list(model=mlyr, label=y)
  mlyr
}

#This function performs feed forwarding
dnn.ffwd <- function(obj){
  mdl <- obj$model
  ln <- length(mdl)-1
  res <- list()
  for(i in 1:ln){
    b <- rep(1, nrow(mdl[[i+1]]$wt))
    optmp <- mdl[[i]]$op%%mdl[[i+1]]$wt
    if(mdl[[i+1]]$fn=="relu"){
      res <- optmp
    }
  }
}

```

```

        res <- apply(res, c(1,2), relu)
        #res <- cbind(rep(1, nrow(res)), res)
    }
    if(mdl[[i+1]]$fn=="softmax"){
        res <- sigmoid(optmp)
        #res <- cbind(rep(1, nrow(res)), res)
    }
    if(i<ln){
        res <- cbind(rep(1, nrow(res)), res)
    }
    mdl[[i+1]]$op <- res

}
list(model=mdl, label=obj$label)
}

## This function is for backpropogation operation
dnn.backprop <- function(obj, lr=.1){
    mdl <- obj[[1]]
    y <- obj[[2]]
    ln <- length(mdl)
    cnt <- 1
    er <- 1/nrow(y)*(sum((mdl$output_layer$op-y)^2))
    yd <- NULL
    for(i in ln:2){
        mdlprv <- mdl[[i-1]]
        if(cnt==1){
            yd <- (y-mdl[[i]]$op)*(dervsig(mdl[[i]]$op))
        }else{
            if(mdl[[i]]$fn=="relu"){
                yd<- yd %%% t(mdl[[i+1]]$wt)*drvmrelu(mdl[[i]]$op)
            }else{
                yd <- yd %%% t(mdl[[i+1]]$wt)*dervsig(mdl[[i]]$op)
            }
        }
        #print(yd)
        if(i==ln){
            wtmp <- mdl[[i]]$wt+(lr*t(mdlprv$op))%%yd
        }else{
            yd <- yd[, -1]
            wtmp <- mdl[[i]]$wt+ ((lr*t(mdlprv$op))%%yd)
        }
        mdl[[i]]$wt<- wtmp

        cnt <- cnt+1
    }
    list(model=mdl, label=obj[[2]])
}

#Activation functions
relu <- function(x){
    max(c(x,0))
}
sigmoid = function(x){
    1 / (1 + exp(-x))
}

#derivatives of activation functions
dervsig <- function(x){
    x*(1-x)
}

```



```

dervrelu <- function(x){
  if(x<0)
    return(0)
  else
    return(1)
}
drvmrelu <- function(x){
  apply(x, c(1,2), dervrelu)
}
## Error function
errorf <- function(obj){
  1/nrow(obj$label)*(sum((obj$model$output_layer$op-obj$label)^2))
}
## Prediction function for test results from trained model
predictf <- function(obj=NULL, test_x){
  obj$model$input$op <- cbind(rep(1,nrow(test_x)),test_x)
  obj <- dnn.ffwd(obj)
  p<- obj$model$output_layer$op
  prd <- apply(p,1, function(x)which.max(x))
}

```

Listing 14.3: Installation of the *keras* libraries

```

install.packages("reticulate")
install.packages("keras")
library(keras)
library(reticulate)
install_keras()

# install package for obtaining data for deep learning examples

install.packages('pmlbr')
library("pmlbr")

```

The functions, implemented in Listing 14.2, are used in Listing 14.1 to generate the results shown in Fig. 14.3. Specifically, Fig. 14.3 (bottom) shows the training error depending on the iterations required for the D-FFNN to learn.

For the second example, we use the *keras* package to implement the D-FFNN. In order to use the functionality of this package, we need to install two libraries, *keras* and *reticulate*. In Listing 14.3, we show how to install these packages.

As an example, we develop a simple D-FFNN to classify breast cancer data (class 1, “good prognosis” vs class 2, “bad”); see Chap. 5 for a heatmap of the data. In Listing 14.4, we show how such a D-FFNN is defined using *keras*. The input breast cancer data contains 106 input features of different genes and two output labels. We split the data into two parts for training (80%) and testing (20%). The initialized model contains an input layer of size 106 (neurons); three hidden layers of size 128, 64, and 28; and two output neurons in the last layer, respectively. The loss function

for the D-FFNN is *mean_squared_error*, the output values of the hidden layers are computed using the function *relu()*, and the final output is computed using the function *softmax()*. The other hyperparameters used for training the D-FFNN are a dropout rate (0.2: to avoid overfitting and for error generalization), epoch (50), and batch size (30) for an iterative optimization.

In Fig. 14.4, we show the loss and accuracy depending on the epochs, for training and testing (validation). It can be observed that the loss (respectively the accuracy) decreases (respectively increases) for longer epochs. However, this is not the case for the validation data. For the validation data, there is a discontinuation at moderate epoch values.

The results above are for one training and one validation data set. However, in Chap. 12 we discussed that for model selection, one needs to use CV. For this reason, in the next example, we show how to use cross-validation for the same data. The results of tenfold cross-validation for the loss and accuracy (training and validation) are shown in Fig. 14.5. The main reason for implementing cross-validation is to highlight the importance of hyperparameters in deep learning neural networks. A user must tune the hyperparameters to optimize the output. The various hyperparameters in a deep neural network are the following: network architecture, number of neurons per layer, learning rate, batch-size, epoch, regularization constant, and activation functions.

14.4 Convolutional Neural Networks

A convolutional neural network (CNN) is a special feedforward neural network utilizing convolution, ReLU, and pooling layers. Standard CNNs are usually composed of several feedforward neural network layers including convolution, pooling, and fully connected layers.

Typically, in traditional ANNs, each neuron in a layer is connected to all neurons in the next layer, where each connection is a parameter in the network. This can result in a very large number of parameters. Instead of using fully connected layers, a CNN uses a local connectivity between neurons, i.e., a neuron is only connected to nearby neurons in the next layer. This can significantly reduce the total number of parameters in the network.

Furthermore, all the connections between local receptive fields and neurons use a common set of weights, called a kernel. A kernel will be shared with all the other neurons that connect to their local receptive fields, and the results of these calculations between the local receptive fields and neurons using the same kernel will be stored in a matrix called an *activation map*. The sharing property is referred to as weight sharing of CNNs [297]. Consequently, different kernels will result in different activation maps, and the number of kernels can be adjusted with hyperparameters. Thus, regardless of the total number of connections between the neurons in a network, the total number of weights corresponds only to the size of the local receptive field, i.e., the size of the kernel. This is visualized in Fig. 14.6b,

Listing 14.4: Deep Feed Forward Neural Networks using keras in R

```

#loading Breast cancer data
#The data is a two class breast cancer data divided into "good
  prognosis" and "poor prognosis" of cancer

load("GeneExpData.RData")
load("GoodPrognostis.RData")
lb <- rep(1, ncol(data))
names(lb) <- colnames(data)
lb[goodProg]<- 0
pdat <- t(data)
lb1 <- to_categorical(lb)
rownames(lb1) <- names(lb)
lb <- as.matrix(lb1)

ind <- sample(2,nrow(pdat),replace = TRUE,prob = c(0.80,0.20))
train <- pdat[which(ind==1),]
test <- pdat[which(ind==2),]
train_lab <- as.matrix(lb[rownames(train),])
test_lab <- as.matrix(lb[rownames(test),])
train <- as.matrix(train)
test <- as.matrix(test)
### Deep Neural network function
dnn <- function(loss="mean_squared_error", optimizer =
  optimizer_rmsprop(), metrics = "accuracy", input_shape=c(106),
  dr=.2, in_layer_size=c(128, 64,28),
  output_shape=2){
  model <- keras_model_sequential()
  model <- layer_dense(model, units = in_layer_size[1],
    activation = 'relu', input_shape = c(106))
  model <- layer_dropout(model, rate = dr)
  for(i in in_layer_size[-1]){
    print(i)
    model <- layer_dense(model, units = i, activation = 'relu')
    model <- layer_dropout(model, rate = dr)
  }
  model <- layer_dense(model, units = 2, activation = 'softmax')
  model <- compile(model, loss = loss, optimizer = optimizer,
    metrics = metrics)
}
## creating DNN and training with the created training data
model <- dnn()

##Training model
vs = .2
trn <- model %>% fit(train, train_lab, epochs = 50, batch_size =
  30, validation_split = vs)
## Evaluating the trained model
model %>% evaluate(test, test_lab)
## Visualizing accuracy and loss while training
plot(trn)
## Testing data accuracy##
## Make sure caret library is installed
library(caret)
testthat <- predict(model, test)
testthat <- apply(testthat, 1, function(x)which.max(x)-1)
testlab <- apply(test_lab, 1, function(x)which(x==1)-1)
cm <- confusionMatrix(table(testthat, testlab))

```

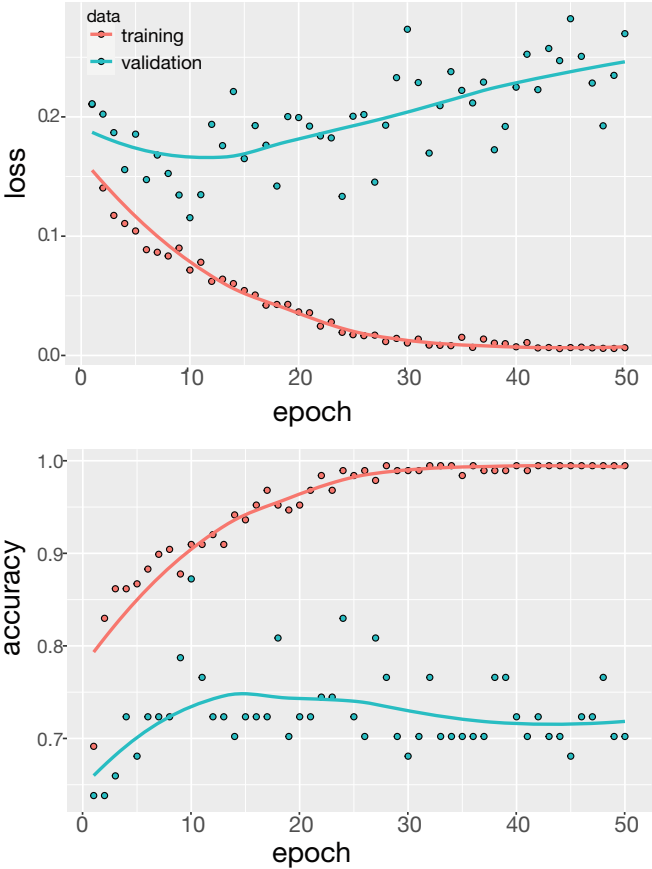
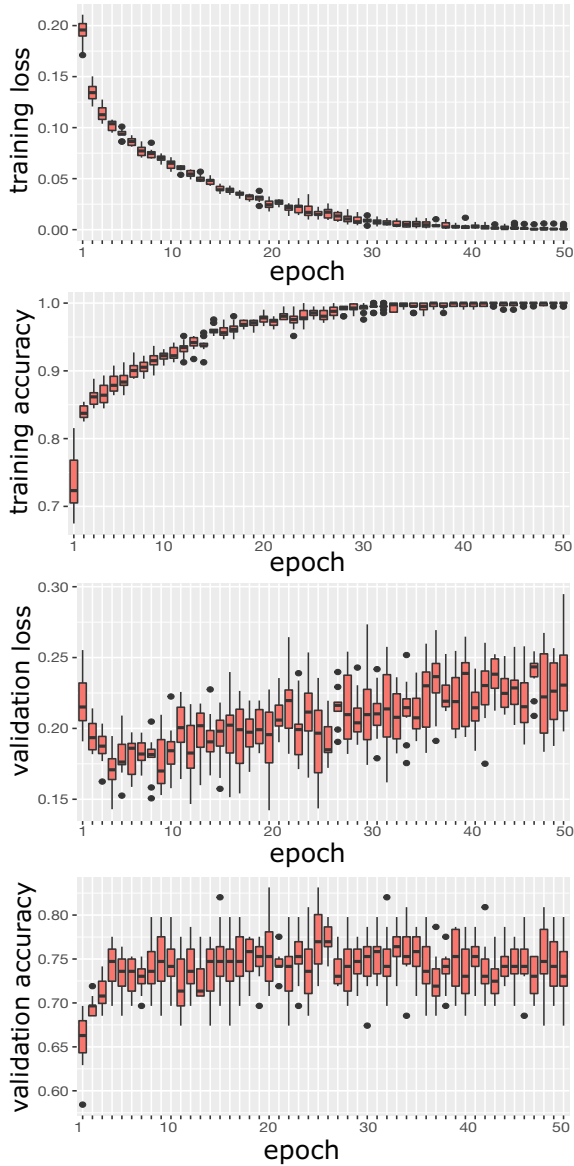


Fig. 14.4 The loss (top) and validation accuracy (bottom) of a D-FFNN trained with the breast cancer data. The results are obtained using Listing 14.4

where the total number of connections between the two layers is 9 but the size of the kernel is only 3.

By combining weight sharing and the local connectivity property, a CNN is able to handle data with high dimensions. See Fig. 14.6a for a visualization of a CNN with three hidden layers. In Fig. 14.6a, the red edges highlight the locality property of hidden neurons, i.e., only very few neurons connect to the succeeding layers. This locality property of CNN makes the network sparse compared to a FFNN, which is fully connected.

Fig. 14.5 Boxplots of loss and accuracy of the D-FFNN for training data (first and second row) and validation data (third and fourth row) for a tenfold cross-validation using the breast cancer data example. The results are obtained using Listing 14.5



14.4.1 Basic Components of CNN

14.4.1.1 Convolutional Layer

A convolutional layer is an essential part in building a convolutional neural network. Similar to a hidden layer of an ordinary neural network, a convolutional layer has

Listing 14.5: 10-fold cross validation of example in Listing 14.4

```

# The dnn model in Listing 14.4 is applied for cross validation.
# The cvk function is used for cross validation and returns
  training loss, training accuracy, validation loss and
  validation accuracy.
cvk <- function(data, goodProg,k=10,vs=.3, ...){
  trn1 <- tst1 <- list()
  lb <- rep(1, ncol(data))
  names(lb) <- colnames(data)
  lb[goodProg]<= 0
  pdat <- t(data)
  lb1 <- to_categorical(lb)
  rownames(lb1) <- names(lb)
  lb <- as.matrix(lb1)
  train <- pdat
  train_lab <- as.matrix(lb)
  train <- as.matrix(train)
  fold <- createFolds(y = train_lab[,1], k = k, list = F)
  fld <- unique(fold)
  for(f in fld){

    ind <- which(fld == f)
    train_df <- train[-ind,]
    y_train <- as.matrix(train_lab[-ind, ])
    valid_df <- as.matrix(train[ind,])
    y_valid <- as.matrix(train_lab[ind, ])
    modelx <- NULL
    modelx <- dnn()
    trn <- modelx > fit(train, train_lab,
                        epochs = 50, batch_size = 30,
                        validation_split = vs)

    trn1[[f]] <- trn
  }
  loss <- sapply(trn1, function(x)x$metrics$loss)
  vloss <- sapply(trn1, function(x)x$metrics$val_loss)
  acc <- sapply(trn1, function(x)x$metrics$accuracy)
  vacc <- sapply(trn1, function(x)x$metrics$val_accuracy)
  list(loss=t(loss), vloss=t(vloss), acc=t(acc), vacc=t(vacc))
}
res <- cvk(data, goodProg)

```

the same goal, which is to convert the input into a representation of a more abstract level. However, instead of using a full connectivity, the convolutional layer uses a local connectivity to perform the calculations between input and the hidden neurons. A convolutional layer uses at least one kernel to slide across the input, performing a convolution operation between each input region and the kernel. The results are stored in the activation maps, which can be seen as the output of the convolutional layer. Importantly, the activation maps can contain features extracted by different kernels. Each kernel can act as a feature extractor and will share its weights with all neurons.

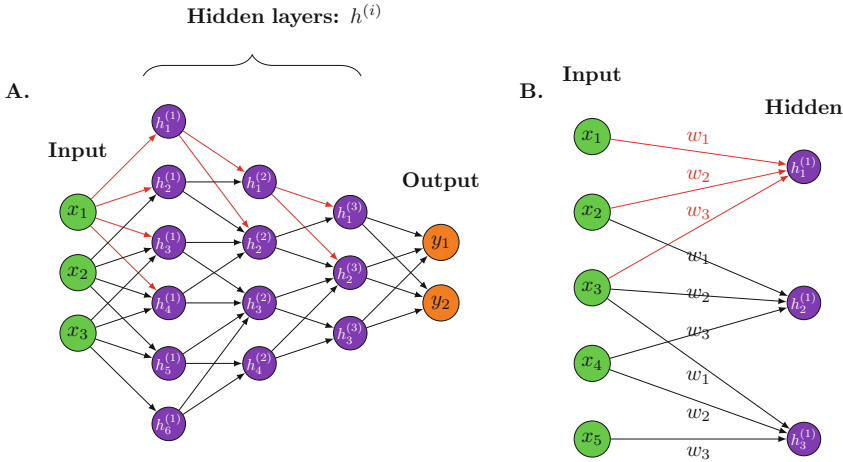


Fig. 14.6 (a) An example for a convolutional neural network. The red edges highlight the fact that hidden layers are connected in a “local” way, i.e., only very few neurons are connected with the succeeding layers. (b) An example of shared weights and local connectivity in CNN. The red edges highlight the fact that hidden layers are connected in a “local” way, i.e., only very few neurons are connected with the succeeding layers. The labels w_1 , w_2 , w_3 indicate the assigned weight for each connection; three hidden nodes share the same set of weights w_1 , w_2 , w_3 when connecting to three local patches

For the convolution process, some spatial arguments need to be defined in order to produce the activation maps of a certain size. Essential attributes include:

1. Size of kernels (N). Each kernel has a window size, which is also referred to as the receptive field. The kernel will perform a convolution operation with a region matching its window size from the input and produces results in its activation map.
2. Stride (S). This parameter defines the number of pixels the kernel will move for the next position. If it is set to 1, each kernel will make convolution operations around the input volume and then shift 1 pixel at a time until it reaches the specified border of the input. Hence, the stride can be used to downsize the dimension of the activation maps since the larger the stride the smaller the activation maps.
3. Zero-padding (P). This parameter is used to specify how many zeros one wants to pad around the border of the input. This is very useful for preserving the dimension of the input.

These three parameters are the most common hyperparameters used to control the output volume of a convolutional layer. Specifically, for an input of dimension $W_{input} \times H_{input} \times Z$, for the hyperparameters size of the kernel (N), stride (S), and zero-padding (P) the dimension of the activation map, i.e., $W_{out} \times H_{out} \times D$ can be calculated as follows:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 0 & 2 & 2 \\ 0 & 3 & 2 & 3 \\ 3 & 1 & 5 & 4 \\ 0 & 3 & 2 & 3 \end{pmatrix}$$

Input matrix: 6×6 Kernel: 3×3 Activation map

Fig. 14.7 Example of calculation of the values in the activation map. Here, the stride is 1 and the zero-padding is 0. The kernel slides by 1 pixel at a time from left to right starting from the top left position. After reaching the border, the kernel will move to the next row and repeat the process until the entire input matrix is covered. The red area indicates the local patch (receptive field) to be convoluted with the kernel, and the result is stored in the green field in the activation map

$$\begin{aligned}
 W_{out} &= \frac{(W_{input} - N + 2P)}{S + 1}, \\
 H_{out} &= \frac{(H_{input} - N + 2P)}{S + 1}. \\
 D &= Z
 \end{aligned} \tag{14.10}$$

An example of how to calculate the result between an input matrix and a kernel is depicted in Fig. 14.7.

The shared weights and the local connectivity help to reduce significantly the total number of parameters of the network. For example, assuming that an input has dimension $100 \times 100 \times 3$, and that the convolutional layer and the number of kernels is 2, and each kernel has a local receptive field of size 4, then the dimension of each kernel is $4 \times 4 \times 3$ (3 is the depth of the kernel which will be the same as the depth of the input volume). For 100 neurons in the layer, there will be in total only $4 \times 4 \times 3 \times 2 = 96$ parameters in this layer because all the 100 neurons will share the same weights for each kernel. This considers only the number of kernels and the size of the local connectivity but does not depend on the number of neurons in the layer.

In addition to the reduction of the number of parameters, shared weights and a local connectivity are important for processing images efficiently. The reason behind this is that local convolutional operations on an image result in values that contain certain characteristics of the image, because in images local values are generally highly correlated and the statistics formed by the local values are often invariant in the location [298]. Hence, using a kernel that shares the same weights can detect patterns from all the local regions in the image, and different kernels can extract different types of patterns from the image.

A nonlinear activation function (for instance, ReLu, tanh, sigmoid, etc.) is often applied to the values from the convolutional operations between the kernel and the

input. These values are stored in the activation maps, which will be later passed to the next layer of the network.

14.4.1.2 Pooling Layer

A pooling layer is usually inserted between a convolutional layer and the following layer. Pooling layers aim at reducing the dimension of the input using some pre-specified pooling method, resulting in a smaller input by conserving as much information as possible. A pooling layer is also able to introduce spatial invariance into the network [419], which can help to improve the generalization of the model. In order to perform pooling, a pooling layer uses a stride, a zero-padding, and a pooling window size as hyperparameters. The pooling layer will scan the entire input using the specified pooling window size in the same manner as the kernel in a convolutional layer. For instance, using a stride of 2, a window size of 2, and a 0 zero-padding for pooling will half the size of the input dimension.

There are many types of pooling methods, e.g., averaging-pooling, min-pooling, and some advanced pooling methods such as fractional max pooling and stochastic pooling. The most commonly used pooling method is max pooling, as it has been shown to be superior in dealing with images by capturing invariances efficiently [419]. Max pooling extracts the maximum value within each specified sub-window across the activation map. The max pooling can be formulated as $A_{i,j,k} = \max(R_{i-n:i+n, j-n:j+n, k})$, where $A_{i,j,k}$ is the maximum activation value from the matrix R of size $n \times n$ centered at index i, j in the k th activation map with n as the window size.

14.4.1.3 Fully Connected Layer

A fully connected layer is the basic hidden layer unit in FFNN (see Sec. 14.2.2). Interestingly, for traditional CNN architectures also, a fully connected layer is often added between the penultimate layer and the output layer to further model nonlinear relationships of the input features [289, 437, 454]. However, recently the benefit of this has been questioned because it introduced many parameters, leading potentially to overfitting [437]. As a result, more and more researchers started to construct CNN architecture without such a fully connected layer using other techniques like max-over-time pooling [279, 311] to replace the role of linear layers.

14.4.2 Important Variants of CNN

VGGNet VGGNet [437] was a pioneer in exploring how the depth of the network influences the performance of a CNN. VGGNet was proposed by the Visual

Geometry Group and Google DeepMind, and they studied architectures with a depth of 19 (e.g., compared to 11 for AlexNet [289]).

VGG19 extended the network from 8 weight layers (a structure proposed by AlexNet) to 19 weight layers by adding 11 more convolutional layers. In total, the parameters increased from 61 millions to 144 millions; however, the fully connected layer takes up most of the parameters. According to their reported results, the error rate dropped from 29.6 to 25.5 for top-1 val.error (percentage of times the classifier did not give the correct class with the highest score) on the ILSVRC data set and from 10.4 to 8.0 for top-5 val.error (percentage of times the classifier did not include the correct class among its top 5) using the ILSVRC data set from ILSVRC2014. This indicates that a deeper CNN structure is able to achieve better results than shallower networks. In addition, they stacked multiple 3x3 convolutional layers without a pooling layer placed in between to replace the convolutional layer with a large filter sizes, e.g., 7x7 or 11x11. They suggested such an architecture is capable to receive the same receptive fields as those composed of larger filter sizes. Consequently, two stacked 3x3 layers can learn features from a 5x5 receptive field, but with less parameters and more nonlinearity.

GoogLeNet with Inception The most intuitive way to improve the performance of a convolutional neural network is to stack more layers and add more parameters to the layers [437]. However, this will impose two major problems. One is that too many parameters will lead to overfitting, and the other is that the model becomes hard to train.

GoogLeNet [454] was introduced by Google. Until the introduction of the inception network architecture, traditional state-of-the-art CNN architectures mainly focused on increasing the size and depth of the neural network, which also increased the computation cost of the network. In contrast, GoogLeNet introduced an architecture to achieve state-of-the-art performance with a light-weight network structure.

The idea underlying an inception network architecture is to keep the network as sparse as possible while utilizing the fast matrix computation feature provided by a computer. This idea facilitates the first inception structure; see Fig. 14.8.

As one can see in the Fig. 14.8, several parallel layers including 1x1 convolution and 3x3 max pooling operate at the same level on the input. Each tunnel (namely, one separated sequential operation) has a different child layer including 3x3 convolutions, 5x5 convolutions, and 1x1 convolution layer. All the results from each tunnel are concatenated together at the output layer. In this architecture, a 1x1 convolution is used to downscale the input image while preserving input information [311]. The authors argued that concatenating all the features extracted by different filters corresponds to the idea that image information should be processed at different scales and only the aggregated features should be sent to the next level. Hence, the next level can extract features from different scales. Moreover, this sparse structure, introduced by an inception block, requires much less parameters, and hence, it is much more efficient.

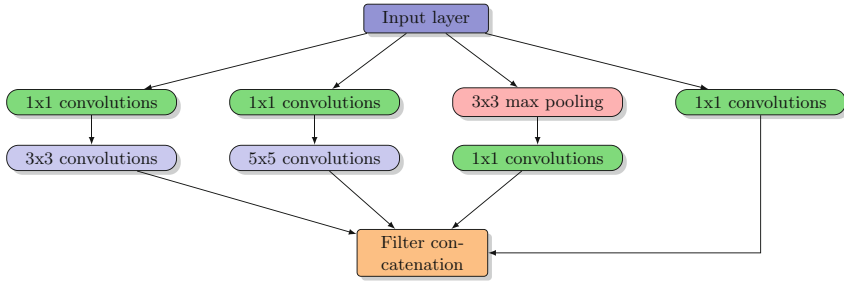


Fig. 14.8 Inception block structure. Here, multiple blocks are stacked on the top of each other, forming the input layer for the next block

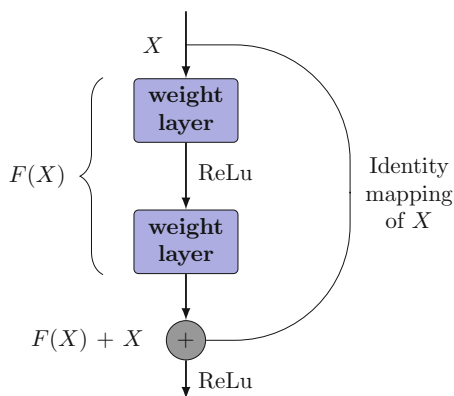
By stacking the inception structure throughout the network, GoogLeNet won the first place in the classification task on ILSVRC2014, demonstrating the quality of the inception structure. Subsequently, inception v1, inception v2, v3, and the latest version v4 were introduced. Each generation introduced some new features, making the network faster, more light-weight, and more powerful.

ResNet In principle, CNNs with a deeper structure perform better than shallow ones [437]. In theory, deeper networks have a better ability to represent high-level features from the input and therefore improving the accuracy of predictions [115]. However, one cannot simply stack more and more layers. In [226], the authors observed that more layers can actually hinder the performance of the model. Specifically, in their experiment, they consider a network A with N layers and a network B with $N + M$ layers, where the initial N layers had the same structure. Interestingly, when training on the CIFAR-10 and ImageNet data sets, network B showed a higher training error compared to network A. In theory, the extra M layers should result in a better performance, but instead they obtained higher errors, which cannot be explained by overfitting. The reason for this is that the loss is getting optimized to a local minima, which differs from the vanishing gradient phenomena. This is referred to as the degradation problem [226].

ResNet [226] was introduced to overcome the degradation problem of CNNs, pushing the depth of a CNN to its limit. In [226], the authors proposed a novel structure of a CNN, which, in theory, can be extended to an infinite depth without losing accuracy. In their paper, they proposed a deep residual learning framework, which consists of multiple residual blocks to address the degradation problem. The structure of a residual block is shown in Fig. 14.9.

Instead of trying to learn the desired underlying mapping, $H(x)$, from each few stacked layers, they used an identity mapping for input x from input to the output of the layer and then let the network learn the residual mapping $F(x) = H(x) - x$. After adding the identity mapping, the original mapping can be reformulated as $H(x) = F(x) + x$. The identity mapping is realized by making shortcut connections from the input node directly to the output node. This can help address

Fig. 14.9 The structure of a residual block. Inside a block, there can be many weight layers



the degradation problem as well as the vanishing (exploding) gradient issue of deep networks. In extreme cases, deeper layers can just learn the identity map of the input to the output layer, by simply calculating the residuals as 0. This enables a deep network to perform at least not worse than shallow ones. Also, in practice, the residuals are never 0, which makes it possible for much deeper layers to always learn something new from the residuals, therefore producing better results. The implementation of ResNet helped to push the layers of CNNs to 152 by stacking so-called residual blocks throughout the network. ResNet achieved the best result in the ILSVRC2016 competition with an error rate of 3.57.

14.4.3 Example: CNN

In Listing 14.6, we present an example using a Convolutional Neural Network in R. For this example, we use again the *keras* package. Furthermore, we use the MNIST data, which provides images of handwritten digits 0–9.

For the MNIST data classification with a CNN, we use three convolution layers with a kernel-size of 3×3 and with max pooling layers and two fully connected layers. The example is presented in Listing 14.6.

The results of Listing 14.6 are shown in Table 14.3. The table shows various error measures including sensitivity, specificity, and F1-score evaluating the performance of the CNN model.

14.5 Deep Belief Networks

A deep belief network (DBN) is a model that combines different types of neural networks with each other to form a new neural network model. Specifically, DBNs

Listing 14.6: Convolutional neural network using *keras*

```

#loading mnist data using pmlbr library (
  url:https://epistasislab.github.io/pmlb/index.html
library(pmlbr)
library(keras)
library(reticulate)
pdat <- fetch_data('mnist')
labs <- as.matrix(pdat[,1], ncol=1)
# splitting data into training and testing data, and converting
  each sample into a 2D array.
ind <- sample(2,length(labs),replace = TRUE,prob = c(0.70,0.30))
train <- array(as.matrix(pdat[which(ind==1),2:785]/255),
  dim=c(length(which(ind==1)),28,28,1))
train_y <- matrix(labs[which(ind==1)], ncol=1)

test <- array(as.matrix(pdat[which(ind==2),2:785]/255),
  dim=c(length(which(ind==2)),28,28,1))
test_y <- matrix(labs[which(ind==2)], ncol=1)

model <- keras_model_sequential() %>%
# First convolution layer with 28 filters on input image with 3x3
  kernel
layer_conv_2d(filters = 28, kernel_size = c(3,3), activation =
  "relu", input_shape = c(28,28,1)) %>%
#Pooling layer
layer_max_pooling_2d(pool_size = c(2,2)) %>%
layer_dropout(rate = 0.2)%>%
layer_conv_2d(filters = 64, kernel_size = c(3,3), activation =
  "relu") %>%
layer_max_pooling_2d(pool_size = c(2,2)) %>%
layer_dropout(rate = 0.2)%>%
layer_conv_2d(filters = 64, kernel_size = c(3,3), activation =
  "relu")%>%
layer_dropout(rate = 0.2)

model %>%
layer_flatten() %>%
layer_dense(units = 128, activation = "relu") %>%
layer_dropout(rate = 0.2) %>%
layer_dense(units = 64, activation = "relu") %>%
layer_dropout(rate = 0.2) %>%
layer_dense(units = 10, activation = "softmax")

model %>% compile(
  optimizer = optimizer_adam(),
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)

history <- model %>%
  fit(
    x = train, y =train_y,
    epochs = 10,
    validation_split = 0.2,
    verbose = 2
  )

model %>% evaluate(test, test_y)

```

```
testthat <- predict(model, test)
testthat <- apply(testthat, 1, function(x)which.max(x)-1)
#confusion matrix using function from caret library
cm <- confusionMatrix(table(as.vector(test_y), testthat))
```

Table 14.3 Sensitivity, specificity, precision, recall, and other metrics of test data from convolutional neural network classifying the MNIST data of Listing 14.6

	Sensitivity	Specificity	Precision	Recall	F1	Prevalence	Detection Rate	Balanced accuracy
Class: 0	0.99	1.00	1.00	0.99	0.99	0.10	0.10	1.00
Class: 1	0.99	1.00	0.99	0.99	0.99	0.11	0.11	1.00
Class: 2	0.99	1.00	0.99	0.99	0.99	0.10	0.10	0.99
Class: 3	0.99	1.00	1.00	0.99	0.99	0.11	0.10	0.99
Class: 4	0.99	1.00	0.99	0.99	0.99	0.10	0.10	0.99
Class: 5	1.00	1.00	0.98	1.00	0.99	0.09	0.09	1.00
Class: 6	0.99	1.00	0.99	0.99	0.99	0.10	0.10	0.99
Class: 7	0.99	1.00	0.99	0.99	0.99	0.10	0.10	1.00
Class: 8	0.99	1.00	0.99	0.99	0.99	0.10	0.10	0.99
Class: 9	0.99	1.00	0.99	0.99	0.99	0.10	0.10	0.99

integrate restricted Boltzmann machines (RBMs) with deep feedforward neural networks (D-FFNN). The RBMs form the input unit, whereas the D-FFNNs form the output unit. Frequently, RBMs are stacked on top of each other, meaning that more than one RBM is used sequentially. This adds to the depth of the DBN.

Due to the different nature of the networks RBM and D-FFNN, two different types of learning algorithms are used. Practically, the RBMs are used to initialize the model in an unsupervised way. Thereafter, a supervised method is applied for the fine-tuning of the parameters [30]. In the following, we describe these two phases of the training of a DBN in more detail.

14.5.1 Pre-training Phase: Unsupervised

Theoretically, neural networks can be learned only using supervised methods. However, in practice it was found that such a learning process can be very slow. For this reason, unsupervised learning is used to initialize the model parameters. The standard neural network learning algorithm (backpropagation) was initially able to learn only shallow architectures. However, using an RBM for the unsupervised initialization of the parameters, one obtains a more efficient training of the neural network [236].

An RBM is a special type of a Boltzmann machine (BM); see Sect. 14.2.3.2. The difference between an RBM and a Boltzmann machine is that RBMs have

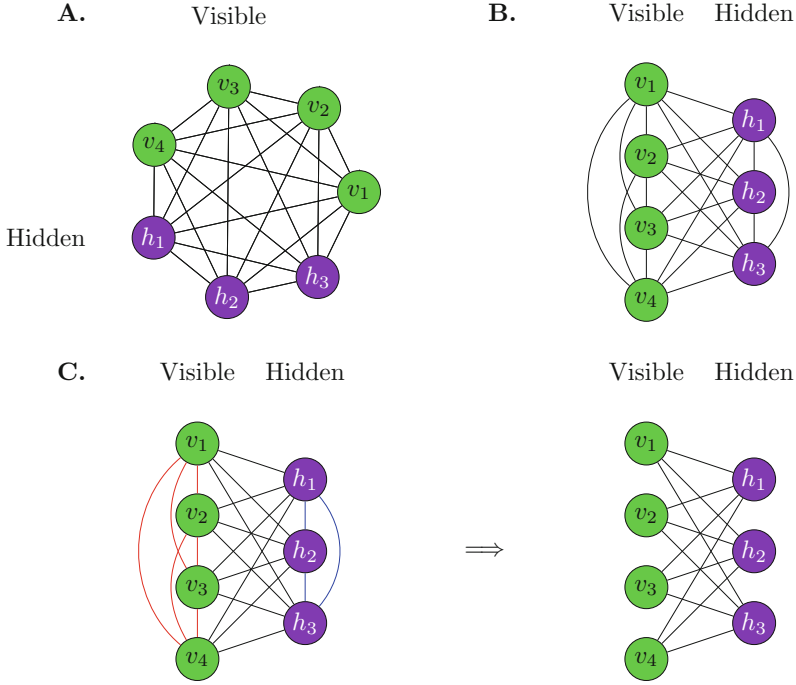


Fig. 14.10 Examples of Boltzmann machines. (a) The neurons are arranged on a circle. (b) The neurons are separated according to their type. Both Boltzmann machines are identical and differ only in their visualization. (c) Transition from a Boltzmann machine (left) to a restricted Boltzmann machine (right)

constraints in the connectivity of their structure [162]. Specifically, there can be no connections between nodes in the same layer. For an example, see Fig. 14.10c.

The values of the neurons, $\mathbf{v}$, in the visible layer are known, but the neuron values, $\mathbf{h}$, in the hidden layer are unknown. The parameters of the network are learned by defining an energy function, E , of the model, which is then minimized.

Frequently, an RBM is used with binary values, i.e., $v_i \in \{0, 1\}$ and $h_i \in \{0, 1\}$. The energy function for such a network is given by [232]

$$E(\mathbf{v}, \mathbf{h}) = - \sum_i^m a_i v_i - \sum_j^n b_j h_j - \sum_i^m \sum_j^n v_i h_j w_{i,j}, \quad (14.11)$$

where $\Theta = \{\mathbf{a}, \mathbf{b}, \mathbf{W}\}$ is the set of model parameters.

Each configuration of the system corresponds to a probability defined via the Boltzmann distribution, in Eq. 14.11:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h})}. \quad (14.12)$$

In Eq. 14.12, Z is the partition function given by

$$Z = \sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}. \quad (14.13)$$

The probability for the network assigned to a visible vector $\mathbf{v}$ is obtained by summing over all possible hidden vectors:

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}. \quad (14.14)$$

Maximum likelihood estimation (MLE) is used to estimate the optimal parameters of this probabilistic model [225]. For a training data set $\mathcal{D} = \mathcal{D}_{train} = \{\mathbf{v}_1, \dots, \mathbf{v}_l\}$, consisting of l patterns, and assuming that the patterns are iid (independent and identical distributed), the log-likelihood function is given by

$$L(\theta) = \ln \mathcal{L}(\theta|\mathcal{D}) = \ln \prod_{i=1}^l p(\mathbf{v}_i|\theta) = \sum_{i=1}^l \ln p(\mathbf{v}_i|\theta). \quad (14.15)$$

For simple cases, one may be able to find an analytical solution for Eq. 14.15, by solving $\frac{\partial}{\partial \theta} \ln \mathcal{L}(\theta|\mathcal{D}) = 0$. However, usually the parameters need to be found numerically. For this, the gradient of the log-likelihood is a typical approach to estimate the optimal parameters:

$$\theta^{(t+1)} = \theta^{(t)} + \Delta\theta^{(t)} = \theta^{(t)} + \eta \frac{\partial L(\theta^t)}{\partial \theta^{(t)}} - \lambda \theta^{(t)} + \nu \Delta\theta^{(t-1)}. \quad (14.16)$$

In Eq. 14.16, the constant, η , in front of the gradient is the learning rate, and the first regularization term, $-\lambda\theta^{(t)}$, is the weight-decay. The weight-decay is used to constrain the optimization problem by penalizing large values of θ [232]. The parameter λ is also called *the weight-cost*. The second regularization term, in Eq. 14.16, is called the *momentum*. The purpose of the momentum is to speed up the learning and reduce possible oscillations. Overall, this should stabilize the learning process.

For the optimization, the Stochastic Gradient Ascent (SGA) is utilized, using *mini-batches*. That means one selects randomly a number of samples from the training set, k , which are much smaller than the total sample size, and then estimates the gradient. The parameters, θ , are then updated for the mini-batch. This process is repeated iteratively until an epoch is completed. An *epoch* is characterized by using the whole training set once. A common problem is encountered when using mini-batches that are too large because this can slow down the learning process considerably. Frequently, k is chosen between 10 and 100 [232].

Before the gradient can be used, one needs to approximate the gradient in Eq. 14.16. Specifically, the derivatives with respect to the parameters can be written

in the following form:

$$\begin{cases} \frac{\partial \mathcal{L}(\theta|\mathbf{v})}{\partial w_{ij}} = p(H_j = 1|\mathbf{v})v_i - \sum_{\mathbf{v}} p(\mathbf{v})p(H_j = 1|\mathbf{v})v_i, \\ \frac{\partial \mathcal{L}(\theta|\mathbf{v})}{\partial a_i} = v_i - \sum_{\mathbf{v}} p(\mathbf{v})v_i, \\ \frac{\partial \mathcal{L}(\theta|\mathbf{v})}{\partial b_j} = p(H_j = 1|\mathbf{v}) - \sum_{\mathbf{v}} p(\mathbf{v})p(H_j = 1|\mathbf{v}). \end{cases} \quad (14.17)$$

In Eq. 14.17, H_i denotes the value of the hidden unit i , and $p(\mathbf{v})$ is the probability defined in Eq. 14.14. For the conditional probability, one finds

$$p(H_j = 1|\mathbf{v}) = \sigma\left(\sum_{i=1}^n w_{ij}v_i + b_j\right), \quad (14.18)$$

and correspondingly

$$p(V_i = 1|\mathbf{h}) = \sigma\left(\sum_{j=1}^m w_{ij}h_j + a_i\right). \quad (14.19)$$

Using the above equations, in the presented form, would be inefficient because these equations require a summation over all visible vectors. For this reason, the contrastive divergence (CD) method is used to increase the speed for the estimation of the gradient. In Fig. 14.11 A, we present the pseudocode of the CD algorithm.

The CD uses Gibbs sampling to draw samples from conditional distributions, so that the next value depends only on the previous one. This generates a Markov chain [221]. Asymptotically, for $k \rightarrow \infty$ the distribution becomes the true stationary distribution. Interestingly, already $k = 1$ can lead to satisfactory approximations for the pre-training [68].

In general, pre-training of DBNs consists of stacking RBMs. That means the next RBM is trained using the hidden layer of the previous RBM as visible layer. This initializes the parameters for each layer [233]. Interestingly, the order of this training is not fixed, but it can vary. For instance, the last layer can be trained first and then the remaining ones [236]. In Fig. 14.12, we show an example for the stacking of RBMs.

14.5.2 Fine-Tuning Phase: Supervised

After the initialization of the parameters of the neural network, as described in the previous step, these can now be fine-tuned. For this step, a supervised learning approach is used, i.e., the labels of the samples, omitted in the pre-training phase, are now utilized.

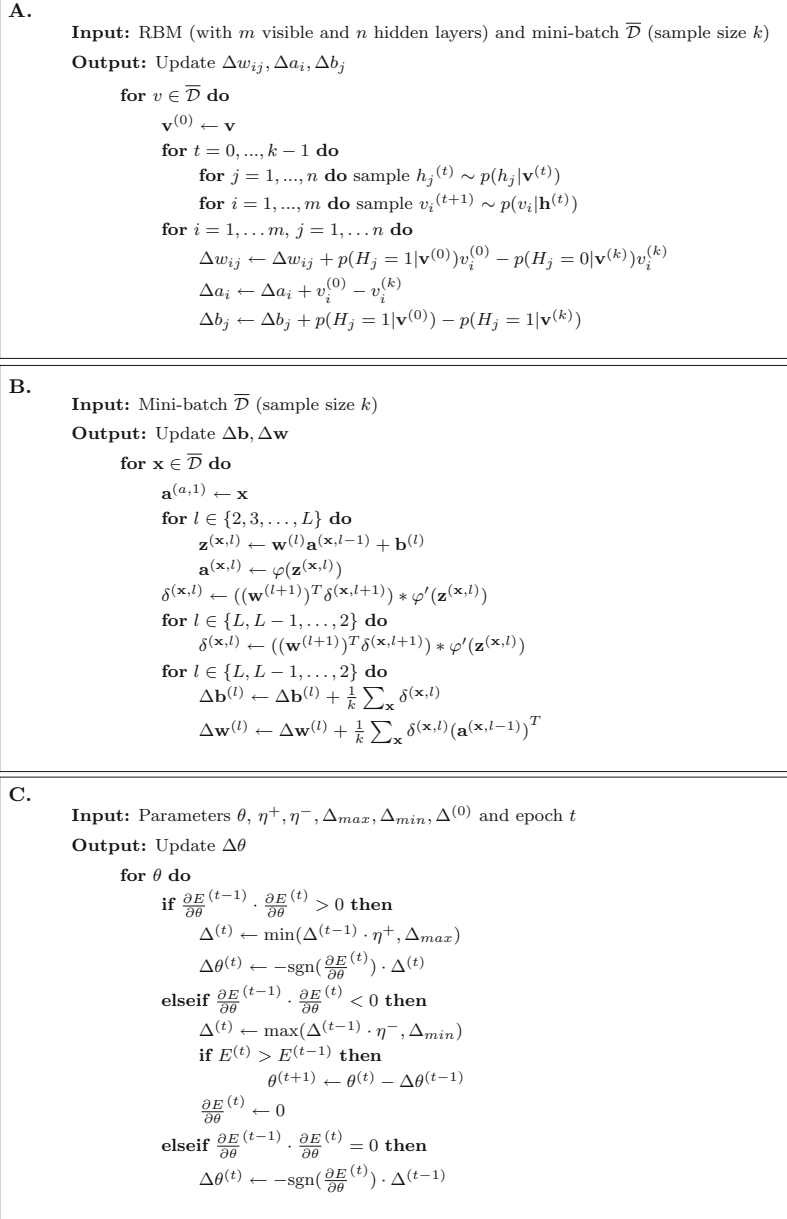


Fig. 14.11 (a) Contrastive divergence k-step algorithm using Gibbs sampling. (b) Backpropagation algorithm. (c) iRprop⁺ algorithm

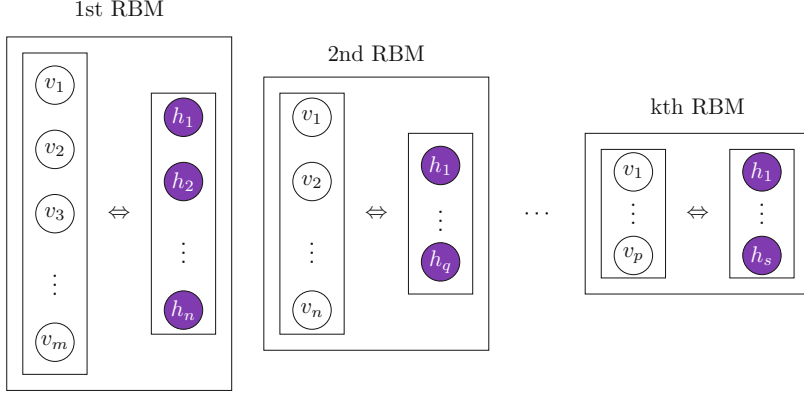


Fig. 14.12 Visualizing the stacking of RBMs in order to learn the parameters Θ of a model in an unsupervised way

For learning the model, one minimizes an error function (also called loss function or sometimes objective function). An example for such an error function is the mean squared error (MSE).

$$E = \frac{1}{2n} \sum_{i=1}^n \|\mathbf{o}_i - \mathbf{t}_i\|^2 \quad (14.20)$$

In Eq. 14.20, $\mathbf{o}_i = \boldsymbol{\phi}(\mathbf{x}_i)$ is the i th output from the network function $\boldsymbol{\phi} : \mathbb{R}^m \rightarrow \mathbb{R}^n$, given the i th input $\mathbf{x}_i$ from the training set $\mathcal{D} = \mathcal{D}_{train} = \{(\mathbf{x}_1, \mathbf{t}_1), \dots, (\mathbf{x}_l, \mathbf{t}_l)\}$ where $\mathbf{t}_i$ is the target output.

Similarly, to maximize the log-likelihood function of an RBM (see Eq. 14.16), one uses gradient descent to find the parameters that minimize the error function as follows:

$$\theta^{(t+1)} = \theta^{(t)} - \Delta\theta^{(t)} = \theta^{(t)} - \eta \frac{\partial E}{\partial \theta^{(t)}} - \lambda \theta^{(t)} + \nu \Delta\theta^{(t-1)} \quad (14.21)$$

Here, the parameters (η , λ , and ν) have the same meaning as explained above. Again, the gradient is typically not used for the entire training data $\mathcal{D}$, but instead smaller batches are used via the *stochastic gradient descent* (SGD).

The gradient of the RBM log-likelihood can be approximated using the CD algorithm (see Fig. 14.11a). For this, the *backpropagation algorithm* is used [298].

Let us denote by a_i^l the activation of the i th unit in the l th layer ($l \in \{2, \dots, L\}$), b_i^l the corresponding bias, and w_{ij}^l the weight for the edge between the j th unit of the $(l-1)$ th layer and the i th unit of the l th layer. For the activation function, φ , the activation of the l th layer with the $(l-1)$ th layer as input is $\mathbf{a}^l = \varphi(\mathbf{z}^l) = \varphi(\mathbf{w}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})$.

Application of the chain rule leads to [365]:

$$\begin{cases} \delta^{(L)} = \nabla_{\mathbf{a}} E \cdot \varphi'(\mathbf{z}^{(L)}), \\ \delta^{(l)} = ((\mathbf{w}^{(l+1)})^T \delta^{(l+1)}) \cdot \varphi'(\mathbf{z}^{(l)}), \\ \frac{\partial E}{\partial b_i^{(l)}} = \delta_i^{(l)}, \\ \frac{\partial E}{\partial w_{ij}^{(l)}} = x_j^{(l-1)} \delta_i^{(l)}. \end{cases} \quad (14.22)$$

In Eq. 14.22, the vector δ^L contains the errors of the output layer (L), whereas the vector δ^l contains the errors of the l th layer. Here, $\cdot$ indicates the element-wise product of vectors. From this, the gradient of the error of the output layer is given by

$$\nabla_{\mathbf{a}} E = \left\{ \frac{\partial E}{\partial a_1^{(L)}}, \dots, \frac{\partial E}{\partial a_k^{(L)}} \right\}. \quad (14.23)$$

In general, the result depends on E . For instance, for the MSE, we obtain $\frac{\partial E}{\partial a_j^{(L)}} = (a_j - t_j)$. As a result, the pseudocode for the backpropagation algorithm can be formulated as shown in Algorithm 14.11b [365]. The estimated gradients from Algorithm 14.11b are then used to update the parameters (weights and biases) via SGD (see Eq. 14.21). More updates are performed using mini-batches until all training data have been used [439].

The *resilient backpropagation algorithm* (Rprop) is a modification of the backpropagation algorithm that was originally introduced to speed up the basic backpropagation (Bprop) algorithm [400]. There exist at least four different versions of Rprop [258], and in Algorithm 14.11d the pseudocode for the iRprop⁺ algorithm (which improves Rprop with weight-backtracking) is shown [439].

As one can see in Algorithm 14.11c, iRprop⁺ uses information about the sign of the partial derivative from the time step ($t - 1$) to make a decision for the update of the parameter. Importantly, the results of comparisons have shown that the iRprop⁺ algorithm is faster than Bprop [258].

It has been shown that the backpropagation algorithm with SGD can learn good neural network models even without a pre-training stage, when the training data are sufficiently large [298].

In Fig. 14.13, we show an example of the overall DBN learning procedure. The left-hand side shows the pre-training phase and the right-hand side the fine-tuning.

14.6 Autoencoder

The next model we discuss is an autoencoder. An autoencoder is an unsupervised neural network model used for representation learning, e.g., feature selection or

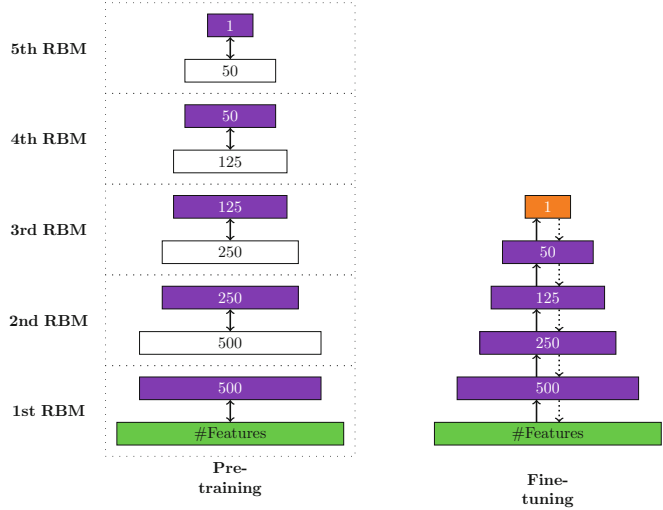


Fig. 14.13 The two stages of DBN learning. Left: The hidden layer (purple) of one RBM is the input of the next RBM. For this reason, their dimensions are equal. Right: The two edges in fine-tuning denote the two stages of the backpropagation algorithm—the input feedforwarding and the error backpropagation. The orange layer indicated the output

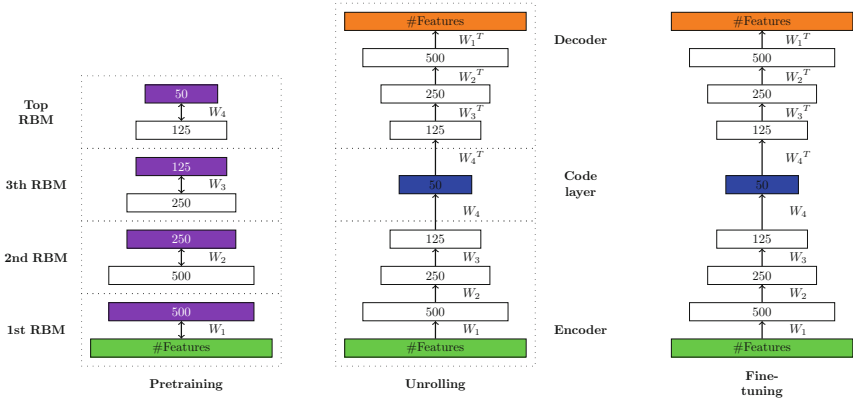


Fig. 14.14 Visualizing the concept of autoencoder learning. The new learned encoding of the input is represented in the code layer (shown in blue)

dimension reduction. A common property of autoencoders is that the size of the input and output layer is the same with a symmetric architecture [233]. The underlying idea is to learn a mapping from an input pattern $\mathbf{x}$ to a new encoding $\mathbf{c} = h(\mathbf{x})$, which ideally gives an output pattern identical to the input pattern, i.e., $\mathbf{x} \approx \mathbf{y} = g(\mathbf{c})$. Hence, the encoding $\mathbf{c}$, which has usually lower dimension than $\mathbf{x}$, allows to reproduce (or code for) $\mathbf{x}$.

The construction of autoencoders is similar to DBNs. Interestingly, the original implementation of an autoencoder [233] pre-trained only the first half of the network with RBMs and then unrolled the network, creating, in this way, the second part of the network. Similar to DBNs, a pre-training phase is followed by a fine-tuning phase. In Fig. 14.14, an illustration of the learning process is shown. Here, the coding layer corresponds to the new encoding, c , providing, e.g., a reduced dimension of x .

An autoencoder does not utilize labels, and, hence, it is an unsupervised learning model. In applications, the model has been successfully used for dimensionality reduction. Autoencoders can achieve a much better two-dimensional representation of array data, when an adequate amount of data is available [233]. Importantly, PCAs implement a linear transformation, whereas autoencoders are nonlinear. Usually, this results in a better performance. We would like to highlight that there are many extensions of these models, e.g., sparse autoencoder, denoising autoencoder, or variational autoencoder [100, 390, 480].

14.6.1 Example: Denoising and Variational Autoencoder

In the following, we present two examples of autoencoders for the MNIST data. The first example is for a denoising autoencoder, and the second is for a variational autoencoder.

The implementation of the denoising autoencoder is shown in Listing 14.7. A denoising autoencoder is a type of convolutional denoising autoencoder with encoder and decoder blocks. The encoder block has two convolutional layers and one fully connected layer with four output nodes. The decoder block is connected with the output of the encoder. The first three layers are the transposed convolutions layers for transforming the encoded information into the reverse direction to regenerate the input data. For the input data of the autoencoder, we add noise from the normal distribution with mean of zero and a variance of 0.1. In Fig. 14.15, we show some sample inputs used (middle row) and outputs (bottom row) used and obtained from the denoising autoencoder.

The implementation of the variational autoencoder is shown in Listing 14.8. The variational autoencoder implements encoder block, sampling layer, and decoding block. The encoder block has four output nodes: latent space parameters denoted $z_mean \mapsto R^2$ and $z_var \mapsto R^2$. The sampling block receives the output of the encoder (latent variables) and generates samples from latent variables as follows: $z = z_mean + \exp(z_var/2) * \epsilon$. Here, ϵ is a random value.

The sampling layer is created by combining the latent space layers. The decoder block is merged with the sampling layer, one dense layer, and three transposed convolutions layers. The implemented loss function considers two losses: the reconstruction loss and the Kullback-Leibler (KL) divergence between the latent space distribution and the prior distribution. The two losses are the *gradient loss* and the *latent loss*, and the former is used to compare input and output data of the

Listing 14.7: Denoising Autoencoder

```

library(keras)
library(reticulate)
library(abind)

pdats <- fetch_data('mnist')
labs <- as.matrix(pdats[,1], ncol=1)
# splitting data into training and testing data, and converting
  each sample into a 2D array.
ind <- sample(2,length(labs),replace = TRUE,prob = c(0.70,0.30))
x_train <- as.matrix(pdats[which(ind==1),2:785])/255
y_train <- matrix(labs[which(ind==1)], ncol=1)

x_test <- as.matrix(pdats[which(ind==2),2:785])/255
y_test <- matrix(labs[which(ind==2)], ncol=1)

## Adding noise to the input data ##
x_train_d <- t(apply(x_train, 1,
  function(x)x+(rnorm(length(x),0,1)*.1)))

x_test_d <- t(apply(x_test, 1,
  function(x)x+(rnorm(length(x),0,1)*.1)))

## Converting images to array ##
x_train_d <- array(x_train_d, dim=c(nrow(x_train),28,28,1))
x_test_d <- array(x_test_d, dim=c(nrow(x_test),28,28,1))

## Constructing encoding layers
inp <- layer_input(shape = c(28,28,1))
ecdc <- inp %>% layer_conv_2d(filters=32, kernel_size=c(3,3),
  activation="relu", input_shape=c(28,28,1), padding="same")
ecdc <- ecdc %>% layer_dropout(rate = 0.2)
ecdc <- ecdc %>% layer_max_pooling_2d(c(2, 2), padding="same")
ecdc <- ecdc %>% layer_conv_2d(filters=32, kernel_size=c(3,3),
  activation="relu", padding="same")
#final encoding layer
ecdc <- ecdc %>% layer_dropout(rate = 0.2)
ecdc <- ecdc %>% layer_max_pooling_2d(c(2, 2), padding="same")

##decoding layers
ecdc <- ecdc %>% layer_conv_2d_transpose(filters=32,
  kernel_size=c(3, 3), strides=2, activation="relu",
  padding="same")
ecdc <- ecdc %>% layer_dropout(rate = 0.2)
ecdc <- ecdc %>% layer_conv_2d_transpose(filters=32,
  kernel_size=c(3, 3), strides=2, activation="relu",
  padding="same")
ecdc <- ecdc %>% layer_dropout(rate = 0.2)
ecdc <- ecdc %>% layer_conv_2d_transpose(filters=1,
  kernel_size=c(3, 3), activation="sigmoid", padding="same")

autoencoder = keras_model(inp,ecdc)
autoencoder %>% compile(optimizer="adam",
  loss="binary_crossentropy")
summary(autoencoder)

autoencoder %>% fit(
  x_train_d,

```

```

x_train_d,
epochs=100,
batch_size=1000,
shuffle=TRUE,
validation.split=0.50
)
## Predicting results from test data

prd <- predict(autoencoder, x_test_d)
## Visualizing random 20 predictions
smp <- sample(dim(x_test_d)[1], 20)
tmp1 <- c(); for(i in smp) tmp1 <-
  rbind(tmp1, array(x_test[i,], c(28,28)))
tmp2 <- c(); for(i in smp) tmp2 <- rbind(tmp2, x_test_d[i,,])
tmp3 <- c(); for(i in smp) tmp3 <- rbind(tmp3, p[i,,])
image(cbind(tmp1, tmp2, tmp3))

```



Fig. 14.15 Visualization of 10 randomly selected digits (test data) used for predicting the output of an autoencoder. The original digits are shown at the top row and the noisy digits used for training in the middle row, and the output of the denoising autoencoder (as produced by Listing 14.7) is shown in the bottom row

variational autoencoder model. The latent loss is the KL divergence loss, which is used for comparing latent vector output distribution with the standard Gaussian distribution of zero mean and unit variance ($N(0, I_{l/2})$, where l is the latent space dimension). A latent output distribution other than normal leads to high loss. A KL divergence loss aims to avoid overfitting and to obtain a sufficient variation for generative properties of the model.

The implemented functions `sampling()` and `vae_loss()`, used in various examples provided in *keras*' library documentation, are similar. The training process of examples on a variational autoencoder automatically updates the objects “encoder” (encoder), “decoder” (decoder), and “vae” (variational autoencoder) simultaneously.

In order to explain the results from the variational autoencoder, we provide in the following three visualizations. The first visualization is for the latent space of the test data, which is the output from the encoder. Here, we consider means from the output of the encoder model, $z_mean = \{z_1, z_2\}$, which are z_1 and z_2 and shown in the scatter plot in Fig. 14.16 (generated using Listing 14.10). In this figure, different digits are highlighted by color (see legend in Fig. 14.16). This shows the projection of the distribution of means from the output of the encoder (latent space) for the test data, where different colors represent different digits from 0 to 9. The color of different distributions of digits shows that the digits are grouped together, except

Listing 14.8: Variational Autoencoder

```

if (tensorflow::tf$executing_eagerly())
  tensorflow::tf$compat$v1$disable_eager_execution()

library(keras)
library(reticulate)
library(abind)

pdat <- fetch_data('mnist')
labs <- as.matrix(pdat[,1], ncol=1)
# splitting data into training and testing data, and converting
# each sample into a 2D array.
ind <- sample(2,length(labs),replace = TRUE,prob = c(0.70,0.30))
x_train <- as.matrix(pdat[which(ind==1),2:785])/255
y_train <- matrix(labs[which(ind==1)], ncol=1)

x_test <- as.matrix(pdat[which(ind==2),2:785])/255
y_test <- matrix(labs[which(ind==2)], ncol=1)

x_train <- array(x_train, dim=c(nrow(x_train),28,28,1))
x_test <- array(x_test, dim=c(nrow(x_test),28,28,1))

# Functions for sampling and loss calculation in a VAE.

sampling <- function(arg){

  z_mean <- arg[, 1:(latent_dim)]
  z_var <- arg[, (latent_dim + 1):(2 * latent_dim)]
  #print(class(arg))
  epsilon <- k_random_normal(
    shape = c(k_shape(z_mean)[[1]]),
    mean=0.,
    stddev=epsilon_std
  )
  #print(dim(z_mean))
  #print(dim(z_var))

  xx <- z_mean + k_exp(z_var/2)*epsilon
  #print(dim(xx))
  xx
}

img_rows <- 28
img_cols <- 28
latent_dim <- 2
epsilon_std <- 1.0

## VAE loss calculation function while training
vae_loss <- function(x, x_decoded_mean_squash) {
  latent_dim <- 2
  epsilon_std <- 1
  x <- k_flatten(x)
  x_decoded_mean_squash <- k_flatten(x_decoded_mean_squash)
  xent_loss <- 1.0 * img_rows * img_cols *
    loss_binary_crossentropy(x, x_decoded_mean_squash)
  kl_loss <- -0.5 * k_mean(1 + z_var - k_square(z_mean) -
    k_exp(z_var), axis = -1L)

```

```

    k_mean(xent_loss + kl_loss)
  }

  ### encoder layers###
  enc0 <- layer_input(shape=c(28, 28, 1))
  enc1 <- layer_conv_2d(filters=1, kernel_size=c(1,1),
    activation="relu", strides=1, padding="same", input_shape =
    c(28,28,1))
  enc2 <- layer_conv_2d(filters=32, kernel_size=c(3,3),
    activation="relu", strides=2, padding="same", input_shape =
    c(28,28,1))
  enc3 <- layer_conv_2d(filters=64, kernel_size=c(3,3),
    activation="relu", strides=2, padding="same")
  enc4 <- layer_flatten()
  enc5 <- layer_dense(units=16, activation="relu")
  enctmp <- enc5(enc4(enc3(enc2(enc0))))

  ### sampling ##
  z_mean <- layer_dense(enctmp, units=latent_dim, name="z_mean")
  z_var <- layer_dense(enctmp, units=latent_dim, name="z_var")
  #z <- layer_lambda(layer_concatenate(list(z_mean,
  z_var)),f=sampling)
  z <- layer_concatenate(list(z_mean, z_var), trainable=TRUE)
  zs <- layer_lambda(z, sampling)

  #### decoder layers ###
  dec0 <- layer_input(shape=latent_dim)
  dec1 <- layer_dense(units=7 * 7 * 64, activation="relu")
  dec2 <- layer_reshape(target_shape=c(7, 7, 64))
  dec3 <- layer_conv_2d_transpose(filters=64, kernel_size=c(3,3),
    activation="relu", strides=2, padding="same")
  dec4 <- layer_conv_2d_transpose(filters=32, kernel_size=c(3,3),
    activation="relu", strides=2, padding="same")
  dec5 <- layer_conv_2d_transpose(filters=1, kernel_size=c(3,3),
    activation="sigmoid", padding="same")
  dectmp <- dec5(dec4(dec3(dec2(dec1(dec0)))))
  encoder<- keras_model(enc0, z_mean)
  decoder <- keras_model(dec0, dectmp)

  ## Variational Autoencoder by combining encoding and decoding
  layers
  vae<- keras_model(enc0, dec5(dec4(dec3(dec2(dec1(zs)))))
  vae %>% compile(optimizer = "rmsprop", loss = vae_loss)
  summary(vae)
  vae %>% fit(
    x_train, x_train,
    shuffle = TRUE,
    epochs = 50,
    batch_size = 1500,
    validation_data = list(x_test, x_test)
  )

```

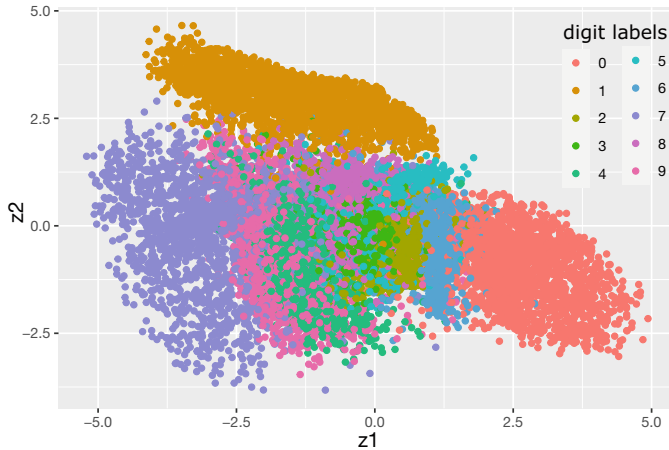


Fig. 14.16 Visualization of the latent space of the test data encoded by the constructed encoder of VAE shown in Example 14.10

digits 0, 1, and 7, which are clearly separated. The distribution of digits shows that the autoencoder estimate the distribution of latent space variables of input digits. However, a perfect separation of the distribution of different digits is also a matter of fine-tuning of the model, considering various model-related parameters and hyperparameters.

Listing 14.9: Prediction results by sampling from the latent space for the variational autoencoder in Listing 14.8

```
## Generating sample points between maximum and minimum values of
## z1 and z2 shown in Figure 14.16.

z1 <- seq(-6, 6, length.out = 30)
z2 <- seq(-6, 6, length.out = 20)

img_gen <- c()
for(i in 1:length(z1)){
  tmp <- c()
  for(j in 1:length(z2)){
    z_smp <- matrix(c(z1[i], z2[j]), ncol = 2)
    prd <- predict(decoder, z_smp)
    tmp <- cbind(tmp, prd[, ,])
  }
  img_gen <- rbind(img_gen, tmp)
}
plot(as.raster((img_gen)))
```

Listing 14.10: Latent space of test data for the variational autoencoder in Listing 14.8

```
#Visualizing latent space of different labels of testing data
prd <- predict(encoder, x_test)
prd <- cbind(prd, y_test)
prd <- as.data.frame(prd)
colnames(prd) <- c("z1", "z2", "digit_label")
prd$digit_label <- as.factor(prd$'digit_label')
p <- ggplot(prd, aes(z1, z2, color=digit_label))+geom_point()
p
```

The second visualization, shown in Fig. 14.17 (top), is the output of the decoder sampled between the maximum and minimum values of z_1 and z_2 from the latent space, shown in Fig. 14.16. These results can be obtained using Listing 14.9. The sampling from the latent space (prior distribution) allows the decoder part of VAE to be used as a generative model. The generated new data parameters are based on sampling from estimated distributions of input data labels used for training (latent space). In Fig. 14.17 (top), one can see the generation of different digits using samples from the maximum and minimum values of z_1 and z_2 .

The third visualization is the output from the variational autoencoder of sampled test data. These are generated using Listing 14.11. The generated figures of ten randomly selected samples of each digit are shown in Fig. 14.17 (bottom row). From Fig. 14.17, one can see the reconstruction of the input test data images, which systematically map the test data to the latent space and samples from latent space parameters to recreate the input data. In contrast with Fig. 14.17 (top row), one can observe some identifying errors in the prediction.

14.7 Long Short-Term Memory Networks

The last model we discuss are long short-term memory (LSTM) networks. LSTM networks were introduced by Hochreiter and Schmidhuber in 1997 [241]. LSTM is a variant of a RNN that has the ability to address the shortcomings of RNNs, which do not perform well, e.g., when handling long-term dependencies [206]. Furthermore, LSTMs avoid the gradient vanishing or exploding problem [186, 240]. In 1999, an LSTM with a forget gate, and which could reset the cell memory was introduced. This improved the initial LSTM and became the standard structure of LSTM networks [186]. In contrast with deep feedforward neural networks, LSTMs contain feedback connections. Furthermore, they can not only process single data points such as vectors or arrays, but sequences of data. For this reason LSTMs are particularly useful for analyzing speech or video data.

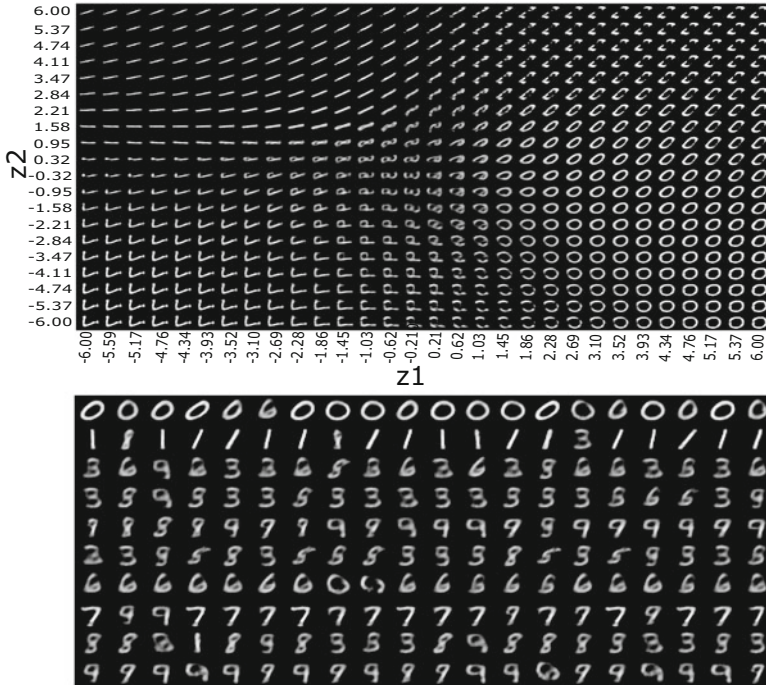


Fig. 14.17 Generated images by randomly sampled values from latent space parameters of encoder and using decoder to generate images (top) in Listing 14.9. Visualization of predicted results of test images by VAE for randomly selected ten samples of digits from 0 to 9 (bottom)

Listing 14.11: Prediction results by sampling from test data for the variational autoencoder in Listing 14.8

```

prd_digit <- predict(vae, x_test)
zz <- tmp <- c()
for(ii in 0:9){
  ind <- which(y_test==ii)
  smp <- sample(ind, 20)
  tmp <- c()
  for(i in smp){
    tmp <- rbind(tmp, prd_digit[i,,])
  }
  zz <- cbind(zz, tmp)
}
image(zz)

```

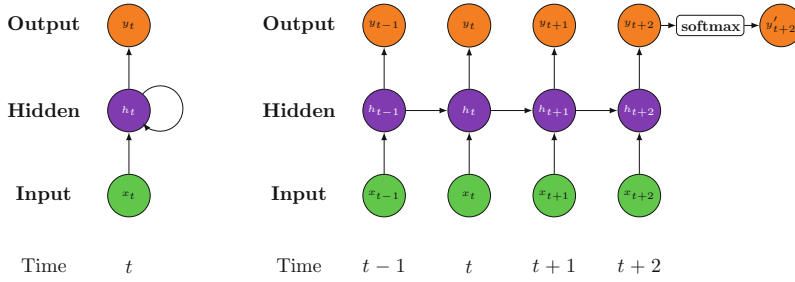


Fig. 14.18 Left: A folded structure of an LSTM network model. Right: An unfolded structure of an LSTM network model. x_i is the input data at time i , and y_i is the corresponding output (i is the time step starting from $(t - 1)$). In this network, only y'_{t+2} , activated by softmax function, is the final network output

14.7.1 LSTM Network Structure with Forget Gate

Figure 14.18 shows an unrolled structure of an LSTM network model [490]. In this model, the input and output are organized vertically, while the information is delivered horizontally over time.

In a standard LSTM network, the basic entity is called LSTM unit or a memory block [186]. Each unit is composed of a cell, the memory part of the unit, and three gates: an input gate, an output gate, and a forget gate (also called keep gate) [187]. An LSTM unit can remember values over arbitrary time intervals, and the three gates control the flow of information through the cell. The central feature of an LSTM cell is a part called “constant error carousel” (CEC) [312]. In general, an LSTM network is formed exactly like a RNN, except that the neurons in the hidden layers are replaced by memory blocks.

In Fig. 14.19, we show a schematic description of an LSTM block with one cell. In the following, we discuss some core concepts and technicalities of LSTM. Let W and U denote the weights and b is the bias. Then, we have the following definitions:

- **Input gate:** A unit with sigmoidal function that controls the flow of information into the cell. It receives its activation from both the output of the previous time $h^{(t-1)}$ and the current input $x^{(t)}$. Under the effect of the sigmoid function, an input gate i^t generates values between zero and one. Zero indicates it blocks the information entirely, whereas values of one allow all the information to pass.

$$i^t = \sigma(W^{(ix)}x^{(t)} + U^{(ih)}h^{(t-1)} + b^i) \quad (14.24)$$

- **Cell input layer:** The cell input has a similar flow as the input gate, receiving $h^{(t-1)}$ and $x^{(t)}$ as input. However, a $\tanh$ activation is used to squish input values into a range between -1 and 1 (denoted by l^t in Eq. 14.25).

$$l^t = \tanh(W^{(lx)}x^{(t)} + U^{(lh)}h^{(t-1)} + b^l) \quad (14.25)$$

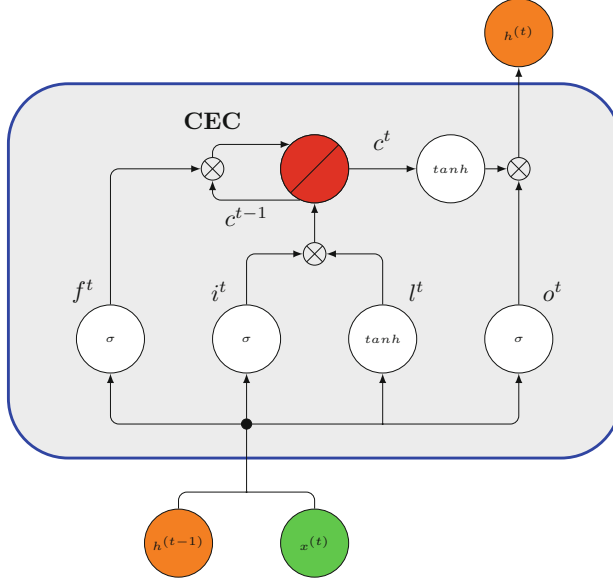


Fig. 14.19 Internal connectivity pattern of a standard LSTM unit (blue rectangle). The output from the previous time step, $h^{(t-1)}$, and $x^{(t)}$, are the input to the block at time t , and then the output $h^{(t)}$ at time t will be an input to the same block in next time step ($t + 1$)

- **Forget gate:** A unit with a sigmoidal function determining which information from previous steps of the cell should be memorized or forgotten. The forget gate f^t (see Eq. 14.26) assumes values between zero and one based on the inputs $h^{(t-1)}$ and $x^{(t)}$. In the next step, the Hadamard product of f^t with old cell state c^{t-1} is used to get the updated new cell state c^t (see Eq. 14.27). In this case, a value of zero means the gate is closed, so it will completely forget the information of the old cell state c^{t-1} , whereas values of one will make all information memorable. Therefore, a forget gate has the right to reset the cell state if the old information is considered meaningless.

$$f^t = \sigma(W^{(fx)}x^{(t)} + U^{(fh)}h^{(t-1)} + b^f) \quad (14.26)$$

- **Cell state:** A cell state stores the memory of a cell over a longer time period [334]. Each cell has a recurrently self-connected linear unit, which is called constant error carousel (CEC) [241]. The CEC mechanism ensures that an LSTM network does not suffer from the vanishing or exploding gradient problem [131]. The CEC is regulated by a forget gate, which can also be used to reset the CEC. At time t , the current cell state c^t is updated by the previous cell state c^{t-1} controlled by the forget gate and the product of the current input and the cell input, i.e., $(i^t \circ l^t)$. Overall, Eq. 14.27 describes the combined update of a cell state:

$$c^t = f^t \circ c^{t-1} + i^t \circ l^t. \quad (14.27)$$

- Output gate: A unit with a sigmoidal function that can control the flow of information out of the cell. An LSTM uses the values of the output gate at time t (denoted by o^t) to control the current cell state c^t activated by a $\tanh$ function, to obtain the final output vector $h^{(t)}$ as follows:

$$o^t = \sigma(W^{(ox)}x^{(t)} + U^{(oh)}h^{(t-1)} + b^o), \quad (14.28)$$

$$h^t = o^t \circ \tanh(c^t). \quad (14.29)$$

14.7.2 Peephole LSTM

A peephole LSTM is a variant of an LSTM proposed in [185]. In contrast with a standard LSTM discussed above, a peephole LSTM uses the cell state c , instead of h , to regulate the forget gate, input gate, and output gate. In Fig. 14.20, we show the internal connectivity of a peephole LSTM unit, where the red arrows represent the new peephole connections.

The key difference between a peephole LSTM and a standard LSTM is that the forget gate f^t , input gate i^t , and output gate o^t do not use $h^{(t-1)}$ as input. Instead,

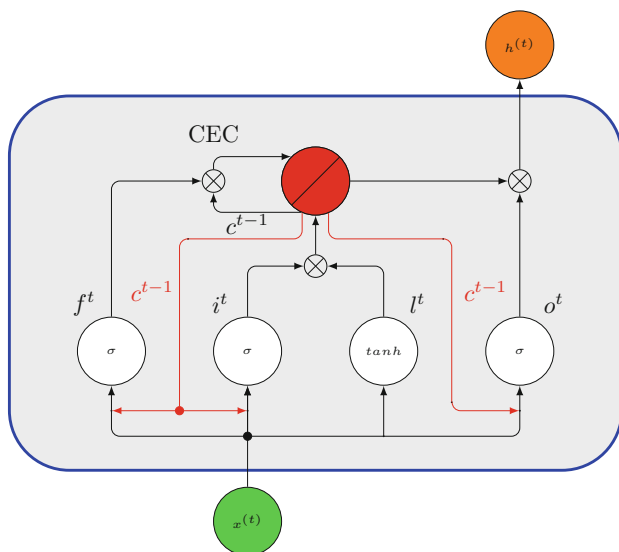


Fig. 14.20 Internal connectivity of a peephole LSTM unit (blue rectangle). Here $x^{(t)}$ is the input to the cell at time t , and $h^{(t)}$ is its output. The red arrows are the new peephole connections added, compared to the standard LSTM in Fig. 14.19

these gates use the cell state c^{t-1} . In order to understand the base idea behind a peephole LSTM, let's assume that the output gate o^{t-1} , in a traditional LSTM network, is closed. Then the output of the network $h^{(t-1)}$ at time $(t-1)$ will be 0, according to Eq. 14.29, and in the next time step t , the regulating mechanism for all three gates will only depend on the network input $x^{(t-1)}$. Therefore, the historical information will be lost completely. A peephole LSTM avoids this problem using a cell state instead of the output, h , to control the gates. The following equations describe a peephole LSTM formally:

$$i^t = \sigma(W^{(ix)}x^{(t)} + U^{(ic)}c^{t-1} + b^i), \quad (14.30)$$

$$l^t = \tanh(W^{(lx)}x^{(t)} + b^l), \quad (14.31)$$

$$f^t = \sigma(W^{(fx)}x^{(t)} + U^{(fc)}c^{t-1} + b^f), \quad (14.32)$$

$$o^t = \sigma(W^{(ox)}x^{(t)} + U^{(oc)}c^{t-1} + b^o), \quad (14.33)$$

$$c^t = f^t \circ c^{t-1} + i^t \circ l^t, \quad (14.34)$$

$$h^t = o^t \circ c^t. \quad (14.35)$$

Aside from these main forms of LSTMs described above, there are further variants. For instance, a bidirectional LSTM network (BLSTM) has been introduced in [207], to access long-range context in both input directions. Furthermore, in 2014, the concept of “gated recurrent unit,” which is viewed as a simplified version of LSTM [76] was proposed. In 2015, Wai-kin Wong and Wang-chun Woo introduced a convolutional LSTM network (ConvLSTM) for precipitation nowcasting [506].

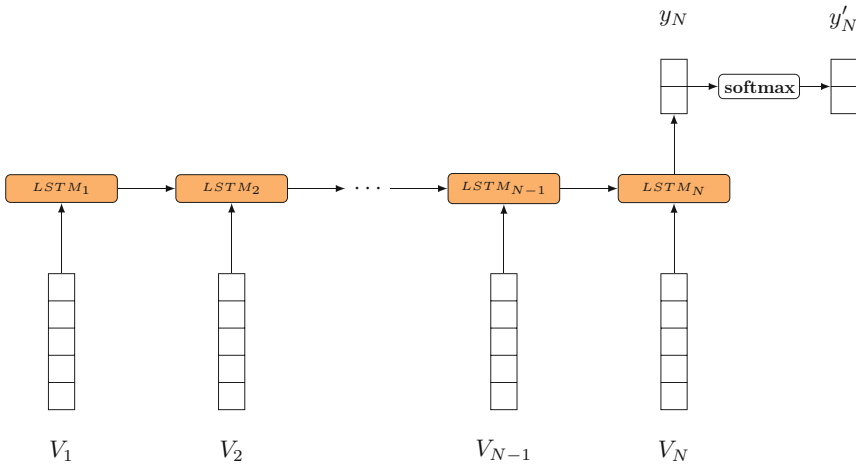


Fig. 14.21 An LSTM classifier model for text classification, where N is the sequence length of the input text (the number of words), V_1 to V_N is a sequence of word embedding vectors used as input to the model at different time steps, and y'_N is the final prediction result

There are further variants of LSTM networks; however, most of them are designed for specific application domains without clear performance advantage.

14.7.3 Applications

LSTMs have a wide range of applications in text generation, text classification, language translation, or image captioning [257, 481]. In Fig. 14.21, an LSTM classifier model for text classification is shown. In this figure, the input of the LSTM structure, at each time step, is a word embedding vector V_i , which is a common choice for text classification problems. A word embedding technique maps the words or phrases, in the vocabulary, to vectors consisting of real numbers. Some common word embedding techniques include word2vec, GloVe, FastText, etc. [524]. The output y_N is the corresponding output at the N -th time step, and y'_N is the final output after softmax activation of y_N , which will determine the classification of the input text.

14.7.4 Example: LSTM

In this section, we discuss three numerical examples for different variations of LSTM models, namely, for:

1. Time series forecasting.
2. Prediction of multiple outputs from multivariate time series data.
3. Automatic text generation, by training the BI-LSTM model with a large English text corpus.

For the time series forecasting model, we use data for the average global temperature. Specifically, we use monthly data from January 1880 until December 2020. The data are provided by NASA's official website page along with various types of geological and image data for the atmosphere of the earth. In the first step, we split the time series data into training and testing data. Next, we prepare data for input and output values. We generate the input temperature vector of $(t - n)$, $(t - n + 1)$, $\dots$, t months as input vector of n features for predicting the temperature of the $(t + 1)^{st}$ month. For our example, we use $n = 6$, i.e., a lag of 6 months. We use 1542 months data for training and the remaining 152 months data for testing.

The implementation of the LSTM model is shown in Listing 14.12. Overall, the LSTM model has two hidden LSTM layers with 50 units that connect to a single output unit. The loss function in the model is the mean square error (*mse*) function. The training steps of the LSTM forecasting models considers other hyperparameters, which are dropout rate, epochs and batch size. Listing 14.12 presents the steps to training the model and visualize the predicted results.

Listing 14.12: LSTM for time series forecasting (average global temperature)

```

#Data is: Average global temperature of the last 150 years
#loading data
d <- readLines("https://data.giss.nasa.gov/
gistemp/tabledata_v4/GLB.Ts+dSST.csv")

#Initial processing of the data
d <- d[-c(1, length(d))]
d <- sapply(d, function(x)strsplit(x, ",")[[1]])
d <- t(d)
rownames(d) <- d[,1]
colnames(d) <- d[1,]
d <- d[-1,]
d <- d[, -1]
d <- d[, 1:12]
d <- t(apply(d, 1, as.numeric))
d <- as.data.frame(d)
tsd <- as.vector(t(d))

## Input data preparation function where step is for defining
input lags (t, t-1, t-step) to predict the t+1 value of a
time series
split_seq = function(x, lag=5){
  z <- y<- c()
  count <- length(x) - lag
  for (i in 1:count) {
    s <- i - 1 + lag
    z <- rbind(z, x[i:s])
    y <- c(y, x[s+1])
  }
  list(x=z, y=y)
}

n <- length(tsd)
tsize <- 150
v <- tsd[1:(n-tsize)]
vt <- tsd[(n-tsize+1):n]

lag<- 6
### LSTM forecasting Model #####
input_ts <- split_seq(v, lag=lag)
model = keras_model_sequential() %>%
  layer_lstm(units = 50, input_shape = c(step,1), return_sequence =
T) %>%
  #layer_lstm(units = 50, input_shape = c(step,1), return_sequence
= T) %>%
  layer_dropout(rate = 0.2) %>%
  layer_lstm(units = 50) %>%
  #layer_lstm(units = 50) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 1)

model %>% compile(
  loss = "mse",
  optimizer = "adam"
)

model %>% summary()

```

```

trn <- array(input_ts$x, dim=c(nrow(input_ts$x), step, 1))
ytrn <- matrix(input_ts$y, ncol=1)
history <- model %>% fit(trn, ytrn, epochs = 100, batch_size =
  32, verbose = 2, validation_split=.2)
scores = model %>% evaluate(trn, ytrn, verbose = 0)

## Visualizing results

input_ts_test <- split_seq(vt, step=step)
tst <- array(input_ts_test$x, dim=c(nrow(input_ts_test$x), step, 1))
ytst <- matrix(input_ts_test$y, ncol=1)
ypred = model %>% predict(tst)

plot(ytrn, col="red", type="l")
ind <- (length(ytrn)+1):(length(ytst)+length(ytrn))
lines(ind, ytst, ylim = c(min(ytst), max(ytst)), col = "red", type
  = "l", lwd = 1,
ylab="Outputs y1 and y2", xlab="Samples")
lines(ind, ypred[, 1], col = "blue", type = "l", lwd = 1)
df1 <- data.frame(temperature=ytrn, index=1:length(ytrn))
#df1 <- data.frame(temperature=1:100, index=1:100)
df2 <- data.frame(temperature=ytst,
  index=(length(ytrn)+1):(length(ytst)+length(ytrn)))
df3 <- data.frame(temperature=ypred,
  index=(length(ytrn)+1):(length(ytst)+length(ytrn)))
p = ggplot() +
  geom_line(data = df1, aes(x = index, y = temperature), color
    = "blue")
p <- p+geom_line(data=df2, aes(x=ind, y=temperature), color='red')
p <- p+geom_line(data=df3, aes(x=ind, y=temperature),
  color='green')

```

The visualization of the predicted results is shown in Fig. 14.22. The temperature values shown in blue and red are training and testing data (actual temperature), respectively. The predicted results, shown in green, approximately capture the trends of growing temperature; however, not so accurately since the temperature growth depends on various factors related to the complexity of the earth's weather patterns, which are not utilized in the model. Additionally, optimizing the predictions requires more effort to fine-tune parameters and hyperparameters for obtaining the best temperature prediction model.

For the second example, we use climate data recorded by the *Max Planck Institute for Biogeochemistry*. The data set consists of fourteen features from which we remove the Kelvin temperature feature (Tpot (k)). From the remaining features, we use the temperature (deg C) and the pressure as output variables, while the remaining variables are used as input.

The LSTM model is developed to predict the $(t + 1)$ th temperature and pressure using 11 input features from time period $t - n, t - n + 1, \dots, t$, where $n = 12$. We first normalize the data and then create an array structure to prepare input data for the LSTM model. In our example, we use 12,000 samples for the training data and 2000 samples for the testing data. Our model uses two LSTM layers, one fully

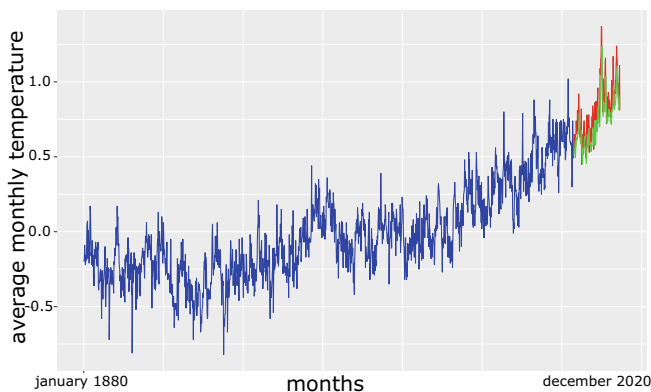


Fig. 14.22 Time series of the average global (monthly) temperature. The values shown in blue were used for the training, whereas the red values were used for testing. The green values are the predicted values by the LSTM model using Listing 14.12

connected layer, and an output layer with two nodes. We use the mean squared error loss function to optimize the model. Listing 14.13 shows the implementation of this LSTM model.

The predicted results for pressure and temperature are shown in Fig. 14.23. The predictions for both outputs capture the trends initially, however, not at the later months where the difference between actual and predicted results becomes large. That shows the complexity of time-series predictions as well as the complexity surrounding the optimization of the parameters of the model.

The third example of the LSTM model is a bidirectional LSTM for text generation. This model is implemented in Listing 14.14. For our analysis, we use a freely available online ebook, “The Republic,” authored by Plato. In the preprocessing step, we convert all upper case characters into lowercases, remove punctuations and digits, and obtain a sequence of text. The next step is the tokenization of each word in text sequence; for that, we use the functions in the library *tokenizers* available in R. The tokenized data is prepared for input and output data for training and testing. We use $\{\mathbf{w}_{\text{input}} = w_{i-n}, w_{i-n+1}, \dots, w_n\}$ words in sequence as input features to predict the (w_{n+1}) th word. The prepared model consists of one BI-LSTM layer, two LSTM layers, and one fully connected layer with an output layer consisting of nodes equal to all tokenized words. The model is trained with a small data set containing 50,000 training samples using “sparse categorical cross-entropy” to optimize the model’s performance. In this example, the training data, model architecture, and hyperparameters play a crucial role, requiring intense effort and time. The results of this model are not optimal; we obtain only 10% accuracy by training the model.

The results of loss and accuracy of the model implemented in Listing 14.14 are shown in Fig. 14.24. In general, for such a modeling it is recommended to consider various configurations of the LSTM and BI-LSTM model and also to study a varieties of text data.

Listing 14.13: LSTM for multi-output forecasting from a multivariate time series

```

# LSTM multi-output prediction
library(readr)
url <- "https://storage.googleapis.com
/tensorflow/tf-keras-datasets/jena_climate_2009_2016.csv.zip"
temp <- tempfile()
download.file(url,temp)
data <- read.table(unz(temp,
  "jena_climate_2009_2016.csv"), sep=",", header = 1)
unlink(temp)

split_seq <- function(d_x, d_y, lag=NULL) {
  x<- y<- c()
  count <- nrow(d_x)- lag
  for (i in 1:count) {
    zz <- i - 1 + lag
    x <- abind(x, (d_x[i:zz,]))
    y <- rbind(y, d_y[zz,])
  }
  x <- aperm(array(x, dim = c(lag, ncol(d_x), count)))
  return(list(x=x, y=y))
}

##Features renaming
columns = c("date_time","pressure","temperature",
  "temperature_kelvin", "temperature_dew_point",
  "relative_humidity",
  "saturation_vapor_pressure", "vapor_pressure",
  "vapor_pressure_deficit", "specific_humidity",
  "water_vapor_concentration", "airtight", "wind_speed",
  "maximum_wind_speed", "wind_direction_in_degrees")

colnames(data) <- columns

## Data normalization
dnorm <- sapply(data, function(x){
  zz<- NULL
  if(class(x)!="character"){
    zz <- (x-mean(x))/sd(x)
  }
  else{zz <- x}
  zz
})

data1 <- as.data.frame(dnorm)
data1 <- sapply(data1, as.numeric)
data1 <- as.data.frame(data1)
data1$date_time <- data$date_time
data <- data1

## Splitting testin and training data
#tstsize <- 30
#trnsize <- round(nrow(data)*(100-70)/100)
trnsize=12000
tstsize=14000
train <- data[1:trnsize,]
test <- data[(trnsize+1):tstsize,]
lag<- 12
dat <- split_seq(train[, -c(1:4)], train[, 2:3], step=step)
trainx <- dat$x

```

```

trainy <- as.matrix(dat$y)

dat <- split_seq(test[, -c(1:4)], test[, 2:3], lag=lag)
testx <- dat$x
testy <- dat$y

train_date <- data$date_time[(step):(trnsize-1)]
test_date <- data$date_time[(trnsize+step):(tstsize-1)]

model = keras_model_sequential() %>%
  layer_lstm(units = 128, input_shape = c(11,12),
    return_sequences=TRUE) %>%
  layer_lstm(units = 64) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units=32) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 2)

model %>% compile(
  loss="mean_squared_error", optimizer = optimizer_rmsprop()
  #loss = "mse",
  #optimizer = "adam")

model %>% summary()
model %>% fit(trainx, trainy, epochs = 100, batch_size = 3000,
  validation_split=.2)
#validation_data= list(testx, testy))

#prediction from training data
prd1 <- predict(model, trainx)

#prediction from test data
prd2 <- predict(model, testx)

pdata <- rbind(prd1, prd2)
actualdata <- rbind(trainy, testy)

library(lubridate)

# Combining actual and predicted values of temperature and
# pressure for visualization
result <- data.frame( date=c(train_date,
  test_date), pressure_predicted=pdata[,1],
  temperature_predicted=pdata[,2],
  pressure_actual=actualdata[,1],
  temperature_actual=actualdata[,2],
  type=c(rep("training", nrow(trainy)),
    rep("testing", nrow(testy))))
result$date <- (parse_date_time(result$date, orders="dmy HMS"))
comb <- c()

#Visualizing pressure output actual and predicted values
p <- ggplot() +
  # blue plot
  geom_line(data=result, aes(x=date, y=pressure_actual,
    color="blue"), size=.5)+
  geom_line(data=result, aes(x=date, y=pressure_predicted,
    color=type), size=.5)+ylab("normalized values of
  pressure")+

```

```

    theme(axis.title.x = element_text(size=14, face="bold"),
          axis.title.y = element_text(size=12),
          axis.text.x=element_text(size=10),
          axis.text.y=element_text(size=10))

#Visualizing temperature output actual and predicted values
p <- ggplot() +
  geom_line(data=result, aes(x=date,
    y=temperature_actual,color="darkblue"), size=.5)+
  geom_line(data=result, aes(x=date,
    y=temperature_predicted,
    color=type),size=.5)+ylab("normalized values of
    temperature")+theme(axis.title.x =
    element_text(size=14, face="bold"), axis.title.y =
    element_text(size=12),
    axis.text.x=element_text(size=10),
    axis.text.y=element_text(size=10))

```

14.8 Discussion

14.8.1 General Characteristics of Deep Learning

A property common to all deep learning models is that they perform the so-called *representation learning*, which is also called feature learning. This describes a model that learns new and better representations compared to the raw data. Importantly, deep learning models do not learn the final representation within one step but multiple ones, corresponding to multilevel representation transformations between the hidden layers [298].

Another common property of deep learning models is that the subsequent transformations between layers are nonlinear (see Fig. 14.2). This increases the expressive power of the model [118]. Furthermore, individual representations are not designed manually, but learned via training data [298]. This makes deep learning models very flexible.

14.8.2 Explainable AI

Any model in data science can be categorized as either an *inferential model* or a *prediction model* [58, 432] (see also Chap. 2). An inferential model does not only make predictions but provides also an interpretable structure. Hence, it is a model of the prediction process itself, e.g., a causal model. In contrast, a prediction model is merely a “black-box” model for making predictions.

The models discussed in this chapter do neither aim to provide physiological models of biological neurons nor do they offer an interpretable structure. Instead, they are prediction models. An example of a biologically motivated learning rule for

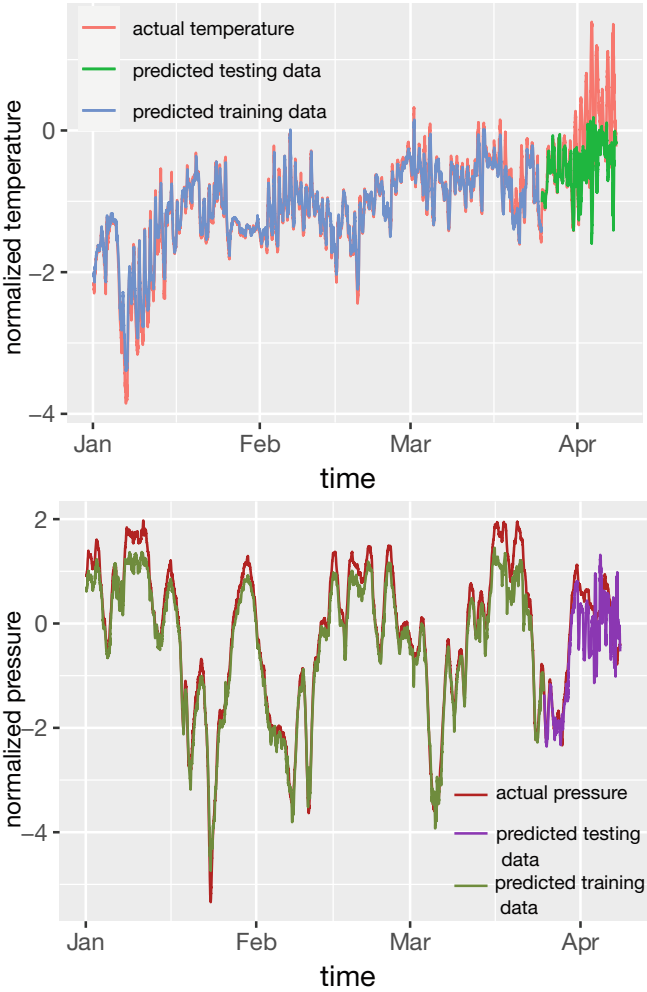


Fig. 14.23 The plots show the temperature (top) and pressure (bottom) values. The red curves correspond to the true values of the temperate and the pressure, the blue curves are the predictions using the training data, and the green (respectively, purple) curves are the predictions for the test data for the temperature (respectively, pressure) obtained using the multi-output LSTM model implemented in Listing 14.13

neural networks is the Hebbian learning rule [227]. Hebbian learning is a form of unsupervised learning of neural networks that does not use global information about the error as backpropagation. Instead, only local information is used from adjacent neurons. There are many extensions of Hebb’s basic learning rule that have been introduced based on new biological insights; see, e.g., [132].

Recently, there is a great interest in interpretable or explainable AI (XAI) [41, 116]. Especially, in the clinical and medical area, one would like to have

Listing 14.14: BI-LSTM for automatic text generation

```

library(keras)
library(readr)
library(stringr)
library(purrr)
library(tokenizers)

maxlen <- 20

# Retrieve text

library(readr)
url <- "https://www.gutenberg.org/cache/epub/1497/pg1497.txt"
path <- get_file(
  'textfile',
  origin=url
)

text <- read_lines(path) %>%
  str_to_lower() %>%
  str_c(collapse = "\n") %>%
  tokenize_words(lowercase=T, simplify = TRUE)

text <- str_replace(text, "\\n", " ")
library(tokenizers)
text <- read_lines(path) %>%
  str_to_lower() %>%
  str_c(collapse = "\n")%>%str_replace_all("[\n]", " ")%>%
  str_replace_all("--", " ")%>%
  str_replace_all("[:punct:]", "")%>%
  str_replace_all("[:digit:]", "")%>%
  str_squish()%>%
  str_split(" ")

text <- text[[1]]

tokenizer <- text_tokenizer() %>%
  fit_text_tokenizer(c(text,"zzzzz"))

word_index <- tokenizer$word_index
print((word_index))

seqvec <- function(x, split=20){
  mat <- c()
  for(i in 1:(length(x)-(split+1))){
    tmp <- x[i:(i+split)]
    tmp <- texts_to_sequences(tokenizer, tmp)
    mat <- rbind(mat, unlist(tmp))
    #print(i)
  }

  invisible(mat)
}

split=20
seqdat <- seqvec(text)

```

```

X <- array(seqdat[,1:split], dim=c(nrow(seqdat),split,1))
#X <- seqdat[,1:20]-1
y <- matrix(seqdat[, (split+1)], ncol=1)
#y = to_categorical(y-1, num_classes=length(unique(y)))

model <- keras_model_sequential()
model%>%layer_embedding(length(tokenizer$index_word),20,
  input_length = dim(X)[2] )
model%>%bidirectional(layer_lstm(units=50,
  return_sequences=TRUE))%>%
layer_dropout(rate = 0.3) %>%
layer_lstm(100)%>%
layer_dropout(rate = 0.3)%>%
layer_dense(units = 128, activation = 'sigmoid')%>%
layer_dropout(rate = 0.3)%>%
layer_dense(units = length(tokenizer$index_word), activation =
  'softmax')

model %>% compile(
  #loss = 'categorical_crossentropy',
  #optimizer = 'adam',
  #metrics = c('accuracy'),
  optimizer = optimizer_adam(),
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)

model %>% fit(
  X, y,
  batch_size = 10,
  epochs = 10,
  validation_split=.2
)

#predicting output of a input sequence input as 'text'. The
#program selects a random location in a sequence and extract
#the input data of size 20, and tokenize it and use is as
#input data to obtain result. The prediction repeated by
#selecting next location from initial position and predict
#output until it completes "total" steps.
getseqpred <- function(text, model, tokenizer, index="random",
  ind=NULL, total=20){
  smp <- sample(length(tokenizer$word_counts),1)
  print(smp)
  txt <- txt1 <- text[smp: (smp+total+100)]
  #print(txt)
  txt1 <- c()
  for(i in 1:total){
    tmp <- (texts_to_sequences(tokenizer, txt[i:(i-1+total)]))
    print(length(txt[i:(i-1+total)]))
    pd <- (pad_sequences(list(unlist(tmp)), maxlen=20,
      truncating='pre'))
    pd <- array(pd, dim=c(1,split))
    yhat = which.max(model%>%predict(pd))
    txt1 <- c(txt1, unlist(tokenizer$index_word[yhat]))
    #print(pd)
    #print(txt)
    if(length(txt1)>total){
      break
    }
  }
}

```

```

    }
  }
  paste0(txt1, collapse=" ")
}
zz <- getseqpred(text, model, tokenizer)

```

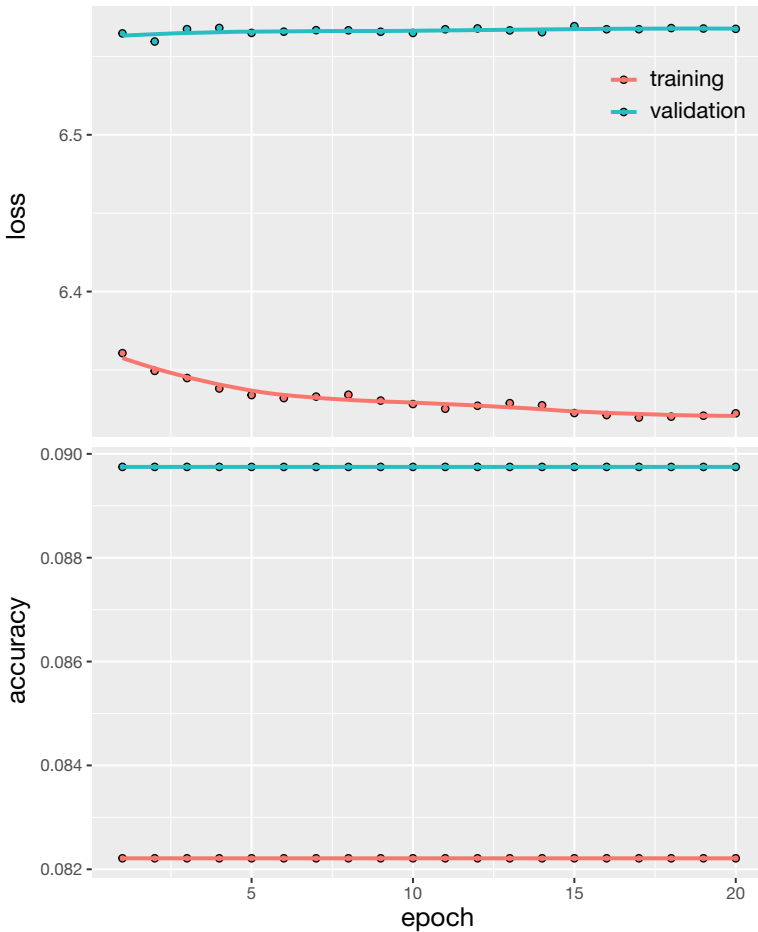


Fig. 14.24 The plots showing loss (top) and accuracy (bottom) for the training and validation data of the text generation model, implemented in Listing 14.14

understandable decisions of statistical prediction models because these affect patients [246]. The field is still in its infancy but if meaningful interpretations of general deep learning models could be found, this would certainly revolutionize the field.

As a note, we would like to add that the distinction between an explainable AI model, and a non-explainable model is not well-defined. For instance, the sparse coding model by [369] was shown to be similar to the coding of images in the human visual cortex [464], and an application of this model can be found in [72], where an unsupervised learning approach was used to learn an optimal sparse coding dictionary for the classification of high spectral imagery (HIS) data. Some may consider this model as an XAI model because of the similarity to the working mechanism of the human cortex, whereas others may question this explanation.

14.8.3 *Big Data vs Small Data*

In statistics, the field of experimental design is concerned with assessing if the available sample sizes are sufficient to conduct a particular analysis (for a practical example, see [450]). In contrast, for all methods discussed in this chapter, we assumed that we are in the big data domain, implying sufficient samples. This corresponds to the ideal case. However, we would like to point out that for practical applications, one needs to assess this situation on a case by case basis to ensure whether the available data (respectively the sample sizes) are sufficient for using deep learning models. Unfortunately, this issue is not well-represented in the current literature. As a *rule of thumb*, deep learning models for image processing usually perform well for tens of thousands of samples, but it is largely unclear how they perform in a small data setting. This leaves it to the user to estimate learning curves of the generalization error, for a given model, to avoid spurious results [140].

As an example to demonstrate this problem, we want to discuss an analysis conducted in [151]. There, the influence of the sample size on the accuracy of the classification of the EMNIST data was explored. Specifically, EMNIST (Extended MNIST) [83] consists of 280,000 handwritten digits (240,000 training samples and 40,000 test samples) for 10 balanced classes corresponding to the digits 0 to 9. A long short-term memory (LSTM) model for a 10-class classifier was used. The model consisted of a four-layer network (three hidden layers and one fully connected layer), and each hidden layer contained 200 neurons. It was found that in order to achieve a classification error below 5%, more than 25,000 training samples (images) are needed. In contrast, in [510] electronic health records (eHR) of patients corresponding to text data have been analyzed for classifying disorder categories. As a result, only hundreds of samples were needed to achieve F-scores over 0.75. A similar order of magnitude for the sample size has been found for gene expression data [441].

Overall, these results demonstrate that the number of samples needed for deep learning models depends crucially on the data type. While image data seem very demanding, other data types require much less data for a deep learning model to perform well.

14.8.4 Advanced Models

Finally, we would like to emphasize that there are additional but more advanced models of deep learning networks, which are outside the core architectures. For instance, deep learning and reinforcement learning have been combined to form deep reinforcement learning [15, 230, 337]. Such models have found application in problems from robotics, games, and healthcare.

Another example of an advanced model is a graph CNN, which is particularly suitable when data have the form of graphs [229, 505] (see also Chap. 2). Such models have been used in natural language processing, recommender systems, genomics and chemistry [308, 511].

14.9 Summary

In this chapter, we provided an overview of deep learning models including deep feedforward neural networks, (D-FFNN), convolutional neural networks (CNNs), deep belief networks (DBNs), autoencoders (AE), and long short-term memory networks (LSTMs). These models can be considered as the core architectures that currently dominate deep learning. In addition, we discussed related concepts needed for a technical understanding of these models, e.g., restricted Boltzmann machines and resilient backpropagation. Given the flexibility of network architectures allowing a *Lego-like* construction of new models, an unlimited number of neural network models can be constructed using elements of the core architectural building blocks discussed in this chapter.

Learning Outcome 14: Deep Learning

Deep learning models allow the realization of analysis models similar to machine learning models. However, the estimation models assume the form of neural networks, which can be learned more efficiently in many practical situations.

We would like to highlight that deep learning does not establish a new learning paradigm, which could not be realized with other machine learning models (see Chap. 17 for a detailed discussion of learning paradigms). Instead, the difference is in the numerical estimation of such models where neural network architectures turned out to provide an economic representation allowing an efficient estimation of its parameters.

14.10 Exercises

1. Discuss and review the components of a mathematical model of an artificial neuron.
2. Study the functional forms an activation function can provide. Discuss the different activation functions in Table 14.1.
3. Implement a simple Feedforward Neural Network using R.
4. Compare the characteristics of deep feedforward neural networks, (D-FFNN), convolutional neural networks (CNNs), deep belief networks (DBNs), autoencoders (AE), and long short-term memory networks (LSTMs) with each other.
5. Repeat the analysis for the D-FFNN by studying the influence of the sample size on the classification error.
6. Repeat the analysis for the LSTM model for the time series forecasting. Vary the model parameters and study the effect on the results.
7. What is the difference between an inferential model and a prediction model?
8. How many samples are needed to learn a deep learning model? Can one provide a generic answer or does this depend on the situation? What influence does the data type have on this?

Chapter 15

Multiple Testing Corrections



15.1 Introduction

When discussing statistical hypothesis testing in Chap. 10, we focused on the underlying concept behind a hypothesis test and on its single application. Here, “single” application means that the hypothesis test is applied only once. However, high-dimensional data make it frequently necessary to apply a statistical hypothesis test multiple times instead of just once. For instance, when analyzing genomic gene expression data, one is interested to identify the activity change for each gene. Given the fact that such data sets contain information for 10–20 thousand genes, one needs to apply a hypothesis test 10–20 thousand times. Similar problems occur in psychology when studying patients, or in Web science for comparing different marketing strategies.

In this chapter, we will see that the transition from one test to multiple tests is not straightforward but requires methodological extensions because otherwise Type 1 errors (below we will see that there is more than one) will increase. Such approaches are summarized under the term *multiple testing procedures (MTPs)* (or *multiple testing corrections (MTCs)* or multiple comparisons (MCs)) [119, 127, 368]. In this chapter, we discuss a number of different MTPs for controlling either the FWER (family-wise error) or the FDR (false discovery rate).

When discussing statistical hypothesis testing in Chap. 10, we have seen that there are two errors one can encounter: Type 1 error and Type 2 error. For this reason, also multiple testing procedures can be evaluated based on these errors. For instance, the FDX (false discovery exceedance [181]) or PFER (per family error rate [205]) are examples for *Type 1 errors*, whereas the FNR (false-negative rate [180]) is a *Type 2 error*. However, in practice, the FWER [50] and the FDR [32, 424] (which are both Type 1 errors) are the most popular ones, and they will be our focus in this chapter.

This chapter is organized as follows. In the next section, we present general preliminaries and definitions required for the subsequent discussion on the MTPs.

Furthermore, we provide information about the practical usage of MTPs using the statistical programming language R. Then, we present a motivation of the problem from both theoretical and experimental perspectives. In Sect. 15.4, we discuss a categorization of different MTPs, and in the Sects. 15.5 and 15.6, we discuss methods to control the FWER and the FDR. We finish by discussing the computational complexity of the most important procedures.

15.2 Preliminaries

In this section, we review briefly some statistical preliminaries needed for the models discussed in the following sections. First, we provide some definitions of our formal setting, error measures, and different types of error control. Second, we describe how to simulate correlated data that can be used for a comparative analysis of different MTPs. For a practical realization of this, we provide also details about the implementation using the statistical programming language R.

15.2.1 Formal Setting

Let's assume that we test m hypotheses where $H_1, H_2, \dots, H_m$ are the corresponding null hypotheses and $p_1, p_2, \dots, p_m$ the corresponding p -values. The p -values are obtained from a comparison of a test statistic, t_i , with a sampling distribution assuming H_i is true. Briefly, assuming two-sided tests, the p -values are given by

$$p_i = Pr(|t_i| > |T(\alpha)| | H_i \text{ is true}), \quad (15.1)$$

where $T(\alpha)$ is a cutoff value determined by the value of the *significance level* α of the individual tests. We indicate the reordered p -values in increasing order as $p_{(1)}, p_{(2)}, \dots, p_{(m)}$ with

$$p_{(1)} \leq p_{(2)} \leq \dots \leq p_{(m)}, \quad (15.2)$$

and the corresponding reordered null hypotheses are $H_{(1)}, H_{(1)}, \dots, H_{(m)}$. When the indices of the reordered p -values are explicitly needed, e.g., for the minP procedure, discussed in Sect. 15.5.6, these p -values are denoted by

$$p_{r_1} \leq p_{r_2} \leq \dots \leq p_{r_m}. \quad (15.3)$$

In general, a MTP can be applied to p -values or cutoff values; however, corrections of p -values are more common because one does not need to specify the type of

Fig. 15.1 A contingency table summarizing the outcome of m hypothesis tests

Truth	Decision		
	reject H_0	accept H_0	
H_0	$N_{1 0}$	$N_{0 0}$	m_0
H_1	$N_{1 1}$	$N_{0 1}$	$m - m_0$
	R	$m - R$	m

the alternative hypothesis (i.e., right-sided, left-sided, or two-sided), and in the following we focus on these.

Depending on the application, the definition of a set of hypotheses, to be considered for a correction, may not always be obvious. In contrast, in genomic or finance applications, tests for genes or stocks provide such definitions naturally, e.g., as pathways or portfolios. In our context such a set of hypothesis is called a *family*. Hence, a MTP is applied to a family of hypothesis tests.

In Fig. 15.1 we summarize the possible outcome of the m hypothesis tests by a *contingency table* (see Chap. 3). Here, we assumed that of the m tests, m_0 are true null hypotheses and $m - m_0$ are false null hypotheses. Furthermore, R is the total number of rejected null hypotheses of which $N_{1|0}$ have been falsely rejected.

The MTCs we discuss, in this chapter, use the following error measures:

$$\text{FWER} = \Pr(N_{1|0} \geq 1); \quad (15.4)$$

$$\text{FDR} = \mathbb{E}[\text{FDP}]; \quad (15.5)$$

$$\text{PFER} = \mathbb{E}[N_{1|0}]; \quad (15.6)$$

$$\text{PCER} = \frac{\mathbb{E}[N_{1|0}]}{m}. \quad (15.7)$$

Here, FWER is the *family-wise error*, which is the probability to make at least one Type 1 error. Alternatively, it can be written as

$$\text{FWER} = 1 - \Pr(N_{1|0} = 0). \quad (15.8)$$

The FDR is the *false discovery rate*. The FDR is the expectation value of the false discovery proportion (FDP) defined as

$$\text{FDP} = \begin{cases} \frac{N_{1|0}}{R} & R \geq 1; \\ 0 & R = 0. \end{cases} \quad (15.9)$$

Finally, PFER is the *per family error rate*, which is the expected number of Type 1 errors, whereas PCER is the *per comparison error rate*, which is the average number of expected Type 1 error across all tests.

Definition 15.1 (Weak Control of FWER) A procedure is said to control the FWER, in the weak sense, if the FWER is controlled at level α only when all null hypotheses are true, i.e., when $m_0 = m$ [239].

Definition 15.2 (Strong Control of FWER) A procedure is said to control the FWER, in the strong sense, if the FWER is controlled at level α for any configuration of null hypotheses.

Similar definitions, like those for a weak and strong control of the FWER stated above, can be formulated for the control of the FDR. In general, a strong control is superior because it allows more flexibility regarding the valid configurations.

Formally, a MTP will be applied to the raw p -values $p_1, p_2, \dots, p_m$, and, according to some method-specific rule,

$$p_i \leq c_i, \quad (15.10)$$

based on cutoff (or critical) values c_i , a decision is made to either reject or accept a null hypothesis. After the application of such an MTP, the problem can be restated in terms of adjusted p -values, i.e., $p_1^{adj}, p_2^{adj}, \dots, p_m^{adj}$. Typically, the adjusted p -values are given as functions of the critical values. For instance, for a single-step Bonferroni correction, the estimation of adjusted p -values corresponds just to a multiplication with a constant factor, $p_i^{adj} = mp_i$, where m is the total number of hypotheses. A more complex example is given by the single-step minP procedure using data-dependent factors [501].

In general, for step-wise procedures, the cutoff values c_i are not constant as they vary with the steps, i.e., with the index i . This makes the estimation of the adjusted p -values more complex. As a result, alternatively, the adjusted p -values can then be used for making a decision, based on the significance level of α , as follows:

$$p_i^{adj} \leq \alpha. \quad (15.11)$$

For historical reasons, we want to mention a very influential conceptual idea that inspired many MTPs, which was introduced by Simes [435]. There, it was proved that the FWER error is weakly controlled if we reject all null hypothesis when there exists an index i with $i \in \{1, \dots, m\}$, such that

$$p_i \leq \frac{i\alpha}{m}. \quad (15.12)$$

That means the (original) Simes correction rejects either all m null hypotheses or none. This makes the procedure practically not very useful because it does not allow to make statements about individual null hypothesis, but conceptually we will find similar ideas in subsequent sections; see Sect. 15.6.1 about the Benjamini-Hochberg procedure.

15.2.2 Simulations Using R

In order to compare MTPs with each other and identify the optimal correction for a given problem, we describe below a general framework that can be applied. Specifically, we show how to generate multivariate normal data with certain correlation characteristics. Since there are many perspectives possible regarding this, we provide two complementary perspectives and the corresponding practical realization using the statistical programming language R [393]. Furthermore, we show how to apply MTPs in R.

15.2.3 Focus on Pairwise Correlations

In general, the population covariance between two random variables X_i and X_j is defined by

$$\sigma_{ij} = \text{cov}(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)]. \quad (15.13)$$

From these, the population correlation between X_i and X_j is defined as follows:

$$\rho_{ij} = \frac{\text{cov}(X_i, X_j)}{\sigma_{ii}\sigma_{jj}} = \frac{\sigma_{ij}}{\sigma_{ii}\sigma_{jj}}. \quad (15.14)$$

In matrix notation, the covariance matrix for a random vector $\mathbf{X}$ of dimension n is given by

$$\mathbf{\Sigma} = \mathbb{E}[(\mathbf{X} - \boldsymbol{\mu}_X)(\mathbf{X} - \boldsymbol{\mu}_X)^T]. \quad (15.15)$$

By utilizing the correlation matrix $\boldsymbol{\rho}$, with elements given by Eq. 15.14, the covariance matrix $\mathbf{\Sigma}$ can be written as

$$\mathbf{\Sigma} = \mathbf{D}_\sigma \boldsymbol{\rho} \mathbf{D}_\sigma, \quad (15.16)$$

with

$$\mathbf{D}_\sigma = \text{diag}(\sigma_{11}, \dots, \sigma_{mm}). \quad (15.17)$$

That means $\mathbf{D}_\sigma$ is a diagonal matrix.

Hence, by specifying the pairwise correlation between the covariates, the corresponding covariance matrix can be obtained. This covariance matrix $\mathbf{\Sigma}$ can then be used to generate multivariate normal data, i.e., $N(\boldsymbol{\mu}, \mathbf{\Sigma})$. To simulate multivariate

normal data with a mean vector of μ and a covariance matrix of Σ , one can use the package *mvtnorm* [183, 184] available in R. An example is presented in Listing 15.1.

Listing 15.1: Generating multivariate normal data with the R package *mvtnorm*. An example is shown for a two-sample, two-sided t-test. Each population is defined by a mean vector of μ (called *mu1* and *mu2*) and a covariance matrix of Σ (called *Sigma1* and *Sigma2*).

```
library(mvtnorm)

# data for group 1
dat1 <- rmvnorm(n=n, mean=mu1, sigma=Sigma2, method="chol")

# data for group 2
dat2 <- rmvnorm(n=n, mean=mu2, sigma=Sigma2, method="chol")

p.values <- vector(length=m) # m: number of hypotheses
for(i in 1:m){
  res <- t.test(dat1[,i], dat2[,i], mu=0, alternative =
    "two.sided")
  p.values[i] <- res$p.val
}
```

15.2.4 Focus on a Network Correlation Structure

The above way to generate multivariate normal data does not allow to control the causal structure among the covariates. It controls only the pairwise correlations. However, for many applications, it is necessary to use a specific correlation structure that is consistent with the underlying causal relations of the covariates. For instance, in biology, the causal relations among genes are given by underlying regulatory networks. In general, such a constrained covariance matrix is given by a Gaussian graphical model (GGM). The generation of a consistent covariance matrix is intricate, and the interested reader is referred to [148] for a detailed discussion.

To simulate multivariate normal data for constrained covariance matrices, one can use the R package *mvgraphnorm* [466]. An example is shown in Listing 15.2.

15.2.5 Application of Multiple Testing Procedures

For the correction of the *p*-values, one can use the function *p.adjust()*, which is part of core R package. This function includes the Šidák, Bonferroni, Holm, Hochberg, Hommel, Benjamini-Hochberg, and Benjamini-Yekutieli procedures. For the Benjamini-Krieger-Yekutieli and Blanchard-Roquain procedures, one can use the functions *multiple.down()* and *BlaRoq()* from the R package *mutoss* [44]. For the SD maxT and SD minP, the package *multtest* [387] can be used (see the Reference Manual for the complex setting of the functions' arguments). Recently,

Listing 15.2: Generating multivariate normal data with the R package *mygraphnorm*. An example is shown for a two-sample, two-sided t-test. Each population is defined by a mean vector of μ (called *mu1* and *mu2*) and a covariance matrix of Σ (called *Sigma1* and *Sigma2*). Furthermore, *g1* and *g2* are two causal structures (networks).

```
library(mvgraphnorm)

# data for group 1
dat1 <- rmvggm(n.samples=n1, net.str=g1, mean=mu1,
               method="htf")$dat

# data for group 2
dat2 <- rmvggm(n.samples=n2, net.str=g2, mean=mu2,
               method="htf")$dat

p.values <- vector(length=m) # m: number of hypotheses
for(i in 1:m){
  res <- t.test(dat1[,i], dat2[,i], mu=0, alternative =
               "two.sided")
  p.values[i] <- res$p.val
}
```

a much faster computational realization has been found for the Hommel procedure, and it is included in the package *hommel* [329].

Listing 15.3: Application of MTPs to raw *p*-values given by *p.values*.

```
library(mutoss)
library(multtest)
library(hommel)

# example usage
p.adj <- p.adjust(p.values, method="hochberg") # Hochberg
correction

p.adj <- multiple.down(p.values, alpha=0.05)$adjPValues # BYK
procedure

p.adj <- BlaRoq(p.values, alpha=0.05, silent=TRUE)$adjPValues #
BR-2S procedure

p.adj <- hommel(p.values) # very fast Hommel procedure
```

15.3 Motivation of the Problem

Before we discuss procedures for dealing with multiple testing corrections, we present motivations that demonstrate the need for such a correction. First, we present theoretical considerations that quantify formally the problem of testing multiple hypotheses and the accompanied misinterpretations of the significance level of a single hypothesis. Second, we provide an experimental example that demonstrates these problems impressively.

15.3.1 Theoretical Considerations

Suppose that we are testing three null hypotheses $H_0 = \{H_1, H_2, H_3\}$ independently, each for a significance level of $\alpha = 0.05$. That means for each hypothesis test H_i with $i \in \{1, 2, 3\}$, we are willing to make a false-positive decision of α where α is defined by

$$\alpha = Pr(\text{reject } H_i | H_i \text{ is true}). \quad (15.18)$$

For these three hypotheses, we would like to know our combined error, or our simultaneous error, in rejecting at least one hypothesis falsely, i.e., we would like to know

$$Pr(\text{reject at least one } H_0 | \text{all } H_0 \text{ are true}). \quad (15.19)$$

In order to obtain this error, we need some auxiliary steps. Assuming independence of the null hypotheses, from the α 's of each hypothesis test, it follows that the probability to accept all three null hypotheses, H_0 , is

$$Pr(\text{accept all three } H_0 | \text{all } H_0 \text{ are true}) = (1 - \alpha)^3. \quad (15.20)$$

The reason for this is that $1 - \alpha$ is the probability to accept H_i when H_i is true, i.e.,

$$1 - \alpha = Pr(\text{accept } H_i | H_i \text{ is true}). \quad (15.21)$$

Furthermore, because all three null hypotheses are independent from each other, $Pr(\text{accept all three } H_0 | \text{all } H_0 \text{ are true})$ is just the product of these hypotheses,

$$Pr(\text{accept all three } H_0 | \text{all } H_0 \text{ are true}) = \prod_{i=1}^3 Pr(\text{accept } H_i | H_i \text{ is true}) \quad (15.22)$$

$$= (1 - \alpha)^3. \quad (15.23)$$

From this, we can obtain the probability to reject at least one H_0 as follows:

$$\begin{aligned} Pr(\text{reject at least one } H_0 | \text{all } H_0 \text{ are true}) &= 1 - Pr(\text{accept all three } H_0 | \text{all } H_0 \text{ are true}) \\ &= 1 - (1 - \alpha)^3. \end{aligned} \quad (15.24)$$

This is just the complement of the probability in Eq. 15.20. For a significance level of $\alpha = 0.05$, we can now calculate

$$Pr(\text{reject at least one } H_0 | \text{all } H_0 \text{ are true}) = 0.14. \quad (15.25)$$

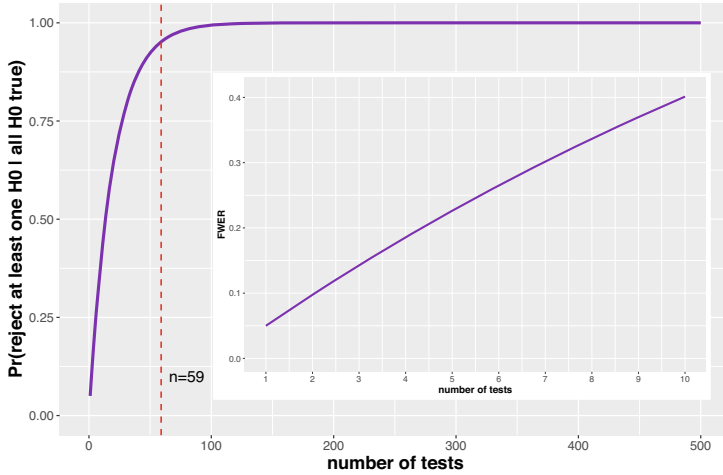


Fig. 15.2 Shown is the $\text{FWER} = \Pr(\text{reject at least one } H_0 | \text{all } H_0 \text{ are true})$ against the number of tests, with $\alpha = 0.05$ for all tests. The inlay highlights the first *ten* tests

That means although we are only making an error of 5% in falsely rejecting H_i for a single hypothesis, the combined error for all three tests is 14%.

In Fig. 15.2, we show the generalization of this result for m independent hypothesis tests given by

$$\text{FWER} = \Pr(\text{reject at least one } H_0 | \text{all } H_0 \text{ are true}) \quad (15.26)$$

$$= 1 - \Pr(\text{accept all } m \text{ } H_0 | \text{all } H_0 \text{ are true}) \quad (15.27)$$

$$= 1 - (1 - \alpha)^m. \quad (15.28)$$

As one can see, the probability to reject at least one H_0 falsely approaches quickly 1. Here, the dashed red line indicates the number of tests for which this probability is 95%. That means for testing 59 tests or more, we are almost certain to make such a false rejection.

The inlay in Fig. 15.2 highlights the first *ten* tests to show that even with a moderate number of tests, the FWER is much larger than the significance level of an individual test. Ideally, one would like a strong control of the FWER because this guarantees a control for all possible combinations of true null hypotheses.

These results demonstrate that the significance level of a single hypothesis can be quite misleading with respect to the error from testing many hypotheses. For this reason, different methods have been introduced to avoid this explosion in errors by controlling them.

15.3.2 *Experimental Example*

In order to demonstrate the practical importance of the problem, an experimental study was presented in [37]. In their study they used a post-mortem atlantic salmon as subject and showed “a series of photographs depicting human individuals in social situations with a specified emotional valence, either socially inclusive or socially exclusive. The salmon was asked to determine which emotion the individual in the photo must have been experiencing” [37]. Using fMRI neuroimaging to monitor the brain activity of the deceased salmon, they found out of 8064 voxels 16 to be statistically significant when testing 8064 hypotheses without any multiple testing correction.

Due to the fact that the physiological state of the fish is clear (it is dead), the measured activities correspond to Type 1 errors. They showed also that by applying multiple correcting procedures these errors can be avoided. The purpose of their experimental study was to highlight the severity of the multiple testing problem in general fMRI neuroimaging studies [36] and the need for applying MTC procedures [362].

Importantly, the above problems are not limited to neuroimaging, but similar problems have been reported in proteomics [111], transcriptomics [121], genomics [195], GWAS [344], finance [219], astrophysics [333], and high-energy physics [88].

15.4 Types of Multiple Testing Procedures

In general, MTPs can be categorized in three different ways.

1. Single-step vs stepwise approaches
2. Adaptive vs nonadaptive approaches
3. Marginal vs joint multiple testing procedures

In the following sections, we discuss each of these categories.

15.4.1 *Single-Step vs Stepwise Approaches*

Overall, there are three different types of MTPs commonly distinguished by the way they *conceptually* compare p -values with critical values [113].

1. Single-step (SS) procedure
2. Step-up (SU) procedure
3. Step-down (SD) procedure

The SU and SD procedures are commonly referred to as stepwise procedures.

Assuming that we have ordered p -values, as given by Eq. 15.2, then the procedures are defined as follows.

Definition 15.3 (Single-Step (SS) Procedure) An SS procedure tests the condition

$$p_{(i)} \leq c_i \quad (15.29)$$

and rejects the null hypothesis i if this condition holds.

For an SS procedure, there is no order required for testing the conditions. Hence, previous decisions are not taken into consideration. Furthermore, usually, the critical values c_i are constant for all tests, i.e., $c_i = c$ for all i .

Definition 15.4 (Step-up (SU) Procedure) Conceptually, an SU procedure starts from the least significant p -value, $p_{(m)}$, and goes toward the most significant p -value, $p_{(1)}$, by testing successively if the following condition holds:

$$p_{(i)} \leq c_i. \quad (15.30)$$

For the first index, i^* , such that this condition holds, the procedure stops and rejects all null hypothesis j with $j \leq i^*$, i.e., the procedure rejects the null hypotheses

$$H_{(1)}, H_{(1)}, \dots, H_{(i^*)}. \quad (15.31)$$

If such an index does not exist, the procedure does not reject any null hypothesis.

Formally, an SU procedure identifies the index

$$i^* = \max \{i \in \{1, \dots, m\} | p_{(i)} \leq c_i\} \quad (15.32)$$

for the critical values c_i . Usually, the c_i 's are not constant but change with the index, i .

Definition 15.5 (Step-Down (SD) Procedure) Conceptually, an SD procedure starts from the most significant p -value, $p_{(1)}$, and goes toward the least significant p -value, $p_{(m)}$, by testing successively if the following condition holds:

$$p_{(i)} \leq c_i. \quad (15.33)$$

For the first index $i^* + 1$ such that this condition **does not hold**, the procedure stops. Then, it rejects all null hypothesis j with $j \leq i^*$, i.e., it reject the null hypotheses

$$H_{(1)}, H_{(1)}, \dots, H_{(i^*)}. \quad (15.34)$$

If such an index does not exist, the procedure does not reject any null hypothesis.

Formally, an SD procedure identifies the index

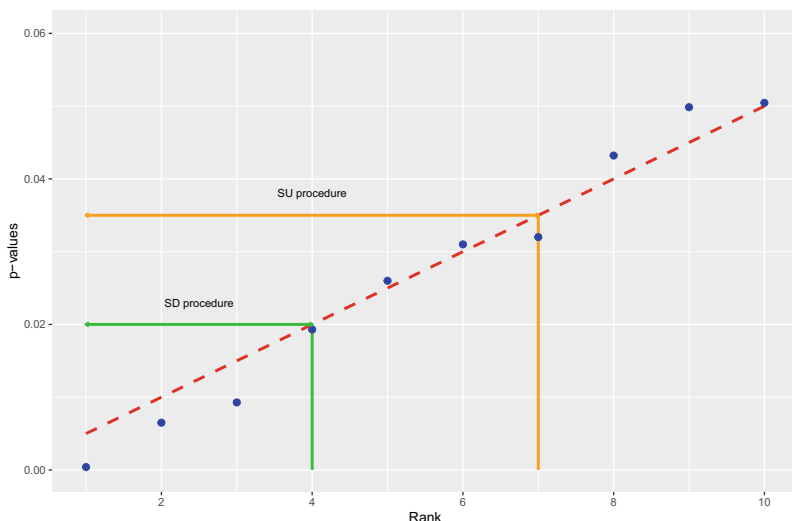


Fig. 15.3 An example visualizing the differences between an SU and an SD procedure. The dashed red line corresponds to the critical values c_i , and the blue points correspond to the rank ordered p -values. The green range indicates p -values identified using an SD procedure, whereas the orange range indicates p -values identified using an SU procedure

$$i^* = \max\{i \in \{1, \dots, m\} | p_{(j)} \leq c_j, \text{ for all } j \in \{1, \dots, i\}\} \quad (15.35)$$

for the critical values c_j .

Regarding the meaning of both procedures, we want to make two remarks. First, the direction, either “up” or “down,” is with respect to the significance of p -values. That means, an SU procedure steps toward more significant p -values (hence it steps up), whereas an SD procedure steps toward less significant p -values (hence it steps down).

Second, the crucial difference between an SU procedure and an SD procedure is that the latter is more strict, requiring all p -values below i^* to be significant as well, whereas the former does not require this.

In Fig. 15.3, we visualize the working mechanism of an SD and an SU procedure. The dashed red line corresponds to the critical values c_i , and the blue points correspond to the rank ordered p -values. Whenever a p -value is below the dashed red line, its corresponding null hypothesis is rejected, otherwise accepted. The green range indicates p -values identified using an SD procedure, whereas the orange range indicates p -values identified using an SU procedure. As one can see, an SU procedure is less conservative than an SD procedure because it does not have the monotonicity requirement.

15.4.2 *Adaptive vs Nonadaptive Approaches*

Another way to categorize MTPs is whether they estimate the number of null hypotheses m_0 from the data or not. The former type of procedures is called adaptive procedure (AD), and the latter are nonadaptive (NA) procedures [33, 414]. Specifically, adaptive MTPs estimate the number of null hypotheses m_0 from a given data set and then use this estimate for a multiple test procedure. In contrast, nonadaptive MTPs assume $m_0 = m$.

15.4.3 *Marginal vs Joint Multiple Testing Procedures*

A third way to categorize MTPs is whether they are using marginal or joint distributions of the test statistics. Multivariate procedures enable to take into account the dependency structure in the data (among the test statistics), and hence, such MTPs can be more powerful than marginal procedures because the latter just ignore this information. For instance, the dependency structure manifests as a correlation structure, which can have a noticeable effect on the results.

Usually, procedures using joint distributions are based on resampling approaches, e.g., bootstrapping or permutations [120, 501]. Hence, they are nonparametric methods, which require computational approaches.

15.5 Controlling the FWER

We start our presentation of MTPs with methods for controlling the FWER [426]. In the following, we will discuss procedures from Šidák, Bonferroni, Holm, Hochberg, Hommel, and Westfall-Young. This discussion emphasizes on the working mechanisms of these procedures. In Sect. 15.8, we present a summary of the underlying assumptions the procedures rely on.

15.5.1 *Šidák Correction*

The first MTP we discuss for controlling the family-wise error (FWER) was introduced by Šidák [434]. Let's say that we want to control the FWER at a level α . If we reverse Eq. 15.28, we obtain an adjusted significance level given by

$$\alpha_S = 1 - (1 - \alpha)^{1/m}. \quad (15.36)$$

This equation allows to calculate, for every FWER of α and every m (number of hypotheses), the corresponding adjusted significance level α_S of the individual hypotheses. A null hypothesis H_i is rejected if

$$p_i \leq \alpha_S \quad (15.37)$$

holds. Hence, using $\alpha_S(m)$, the FWER is controlled at level α .

The procedure given by Eq. 15.37 is called single-step Šidák correction. For completeness, we also want to mention that there is a step-down Šidák correction defined by

$$p_i \leq 1 - (1 - \alpha)^{1/m-i+1}. \quad (15.38)$$

From Eqs. 15.36 and 15.37, we can derive adjusted p -values for the single-step Šidák correction, which are given by

$$p_i^{adj} = \min\{1 - (1 - p_i)^m, 1\}. \quad (15.39)$$

These adjusted p -values can alternatively be used to test for significance by comparing them with the original significance level, i.e.,

$$p_i^{adj} \leq \alpha. \quad (15.40)$$

In Fig. 15.4a, we show the adjusted significance level α_S of the individual hypotheses for a single-step Šidák correction depending on the number of hypotheses m for $\alpha = 0.05$. As one can see, the adjusted significance level α_S becomes quickly more stringent for an increasing number of hypothesis tests m .

15.5.2 Bonferroni Correction

The Bonferroni correction controls the family-wise error (FWER) under general dependence. [50]. From a Taylor expansion of Eq. 15.36 up to the linear term, we obtain the following approximation

$$\alpha_B = \frac{\alpha}{m}. \quad (15.41)$$

Using Boole's inequality, one can elegantly show that this controls the FWER [195].

This is the adjusted Bonferroni significance level. We can use this adjusted significance level to test every p -value and reject the null hypothesis H_i if

$$p_i \leq \alpha_B. \quad (15.42)$$

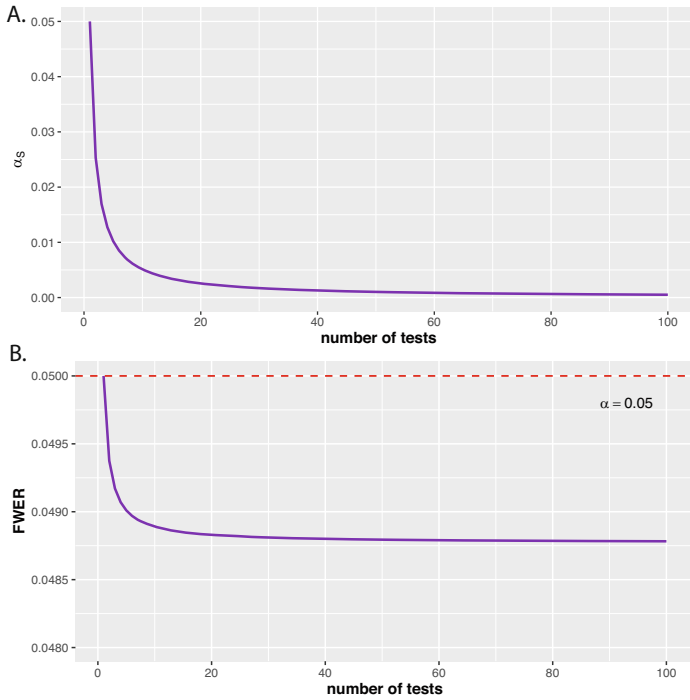


Fig. 15.4 A: Single-step Šidák correction. Shown is α_S in dependence on m for $\alpha = 0.05$. B: Bonferroni correction. Shown is the FWER against m for $\alpha = 0.05$

From Eqs. 15.41 and 15.42, we can derive adjusted p -values, which are given by

$$p_i^{adj} = \min\{mp_i, 1\}. \quad (15.43)$$

These adjusted p -values can alternatively be used to test for significance by comparing them with the original significance level, i.e.,

$$p_i^{adj} \leq \alpha. \quad (15.44)$$

The corresponding result is shown in Fig. 15.4. Specifically, the FWER in Eq. 15.28 is shown for the corrected significance level α_B given by

$$\text{FWER} = 1 - (1 - \alpha_B)^m. \quad (15.45)$$

As one can see, the FWER is controlled for all m because it is always below $\alpha = 0.05$. Here, it is important to emphasize that the y-axis range is only from 0.048 to 0.05 to see the effect.

15.5.3 Holm Correction

A modified Bonferroni correction, called the Holm correction, was suggested in [245]. In contrast with a Bonferroni or a Šidák correction, it is a sequential procedure that tests ordered p -values. For this reason, it was also called “the sequentially rejective Bonferroni test” [245].

Let's denote

$$p_{(1)} \leq p_{(2)} \cdots \leq p_{(m)} \quad (15.46)$$

the ordered sequence of p -values in increasing order. Then, the Holm correction tests the following conditions in a step-down manner:

$$\text{Step 1.: Reject } H_{(1)} \text{ if } p_{(1)} \leq \frac{\alpha}{m}; \quad (15.47)$$

$$\text{Step 2.: Reject } H_{(2)} \text{ if } p_{(2)} \leq \frac{\alpha}{m-1}; \quad (15.48)$$

$$\text{Step 3.: Reject } H_{(3)} \text{ if } p_{(3)} \leq \frac{\alpha}{m-2}; \quad (15.49)$$

$$\vdots$$

$$\text{Step } m\text{.: Reject } H_{(m)} \text{ if } p_{(m)} \leq \frac{\alpha}{1}. \quad (15.50)$$

If at any step the hypothesis, $H_{(i)}$, is not rejected, the procedure stops, and all other p -values, i.e., $p_{(i)}, p_{(i+1)}, \dots, p_{(m)}$ are accepted. The above testing criteria of the steps can be written in the following compact form:

$$p_{(i)} \leq \frac{\alpha}{m-i+1}, \quad (15.51)$$

for $i \in \{1, \dots, m\}$. As one can see, the first step, i.e., $i = 1$, is exactly a Bonferroni correction, and each following step is in the same spirit, but considering the changed number of remaining tests. The optimal cutoff index of this SD procedure can be identified as follows:

$$i^* = \max \left\{ i \in \{1, \dots, m\} \mid p_{(j)} \leq \frac{\alpha}{m-j+1}, \text{ for all } j \in \{1, \dots, i\} \right\}. \quad (15.52)$$

From this, the adjusted p -values of a Holm correction can be derived [127], and they are given by

$$p_{(i)}^{adj} = \max_{j \leq i} \left\{ \min_{k \in \{1, \dots, j\}} \left\{ (m-k+1)p_{(k)}, 1 \right\} \right\}. \quad (15.53)$$

The nested character of this formulation comes from the strict requirement of an SD procedure where all p -values, $p_{(i)}$, need to be significant with $j \leq i$ (see the j index in Eq. 15.52). An alternative, more explicit form to obtain the adjusted p -values is given by the following sequential formulation [114]:

$$p_{(i)}^{adj} = \begin{cases} \min\{mp_{(i)}, 1\} & \text{if } i = 1; \\ \max\{p_{(i-1)}^{adj}, (m - i + 1)p_{(i)}\} & \text{if } i = \{2, \dots, m\}. \end{cases} \quad (15.54)$$

A computational realization of a Holm correction is given by Algorithm 10.

Algorithm 10: SD Holm correction procedure.

Input: $p_{(1)} \leq p_{(2)} \cdots \leq p_{(m)}$
 $k = 0$
while $p_{(k+1)} \leq \alpha/(m - k)$ **do**
 $k = k + 1$
Reject $H_{(1)}, H_{(1)}, \dots, H_{(k)}$

Similar to a Bonferroni correction, a Holm correction also does not require the independence of the test statistics and provides a strong control of the FWER. In general, this procedure is more powerful than a Bonferroni correction.

15.5.4 Hochberg Correction

Another MTC that is formally very similar to the Holm correction is the Hochberg correction [238], shown in Algorithm 11. The only difference is that it is a step-up procedure.

Algorithm 11: SU Hochberg correction procedure

Input: $p_{(1)} \leq p_{(2)} \cdots \leq p_{(m)}$
 $k = m$
while $p_{(k)} > \alpha/(m - k + 1)$ **do**
 $k = k - 1$
Reject $H_{(1)}, H_{(1)}, \dots, H_{(k)}$

The adjusted p -values of the Hochberg correction are given by [114]:

$$p_{(i)}^{adj} = \begin{cases} p_{(i)} & \text{if } i = m, \\ \min\{p_{(i+1)}^{adj}, (m - i + 1)p_{(i+1)}\} & \text{if } i = \{m - 1, \dots, 2\}. \end{cases} \quad (15.55)$$

The Hochberg correction is an optimistic approach because it tests backwards and stops as soon as a p -value is significant at level $\alpha/(m-k+1)$. The SU character makes it more powerful, and hence, the SU Hochberg procedure is more powerful than the SD Holm procedure.

15.5.5 Hommel Correction

The next MTP we discuss, the Hommel correction [247], is far more complex than the previous procedures. The method evaluates the set

$$i^* = \max\{i \in \{1, \dots, m\} | p_{(m-i+k)} > \frac{k}{i}\alpha, \text{ for all } k \in \{1, \dots, i\}\}, \quad (15.56)$$

and determines the maximum index such that this condition holds [16]. If such an index does not exist, then we reject $H_{(1)}, H_{(1)}, \dots, H_{(m)}$; otherwise, we reject only the p -values such that

$$p < \frac{\alpha}{i^*}. \quad (15.57)$$

In Fig. 15.5, we visualize the testing condition for $m = 7$. In this figure, the scaling factor, $f(i, k) = k/I$, is shown as a function of the indices i and k . Each red point corresponds to an index pair (i, k) . Figure 15.5a shows $f(i, k)$ for fixed values of i , whereas Fig. 15.5b shows $f(i, k)$ for fixed values of k . This leads to k and i dependent curves shown in blue, for $f(i \text{ fixed}, k)$ and $f(i, k \text{ fixed})$, respectively. In both figures, the isolines shown as orange dashed lines connect points with a constant index $h = m - i + k$. For each blue line (in both figures), the points on these lines have indices $h = m, h = m - 1$, etc., going from high to low values of $f(i, k)$. These indices are used for the rank ordered p -values, i.e.,

$$P_{(h)} = P_{(m-i+k)}. \quad (15.58)$$

According to Eq. 15.56, for each index i , there are i different values of k . In Fig. 15.5a, these correspond to the points on the blue lines, whereas in Fig. 15.5b, these correspond to the points in vertical columns. In Fig. 15.5b, we highlight just one of these shown as a green dashed line.

The following array in Eq. 15.59 shows, for each step of the procedure, the corresponding conditions that need to be tested. At step 1, the index $i = m$. For simplicity, we use the notation $F(i, k) = f(i, k)\alpha = k\alpha/i$. The procedure stops only if all conditions for a step (for a column) are true, and i^* corresponds to the value of i for this step. Otherwise, the procedure continues to the next step.

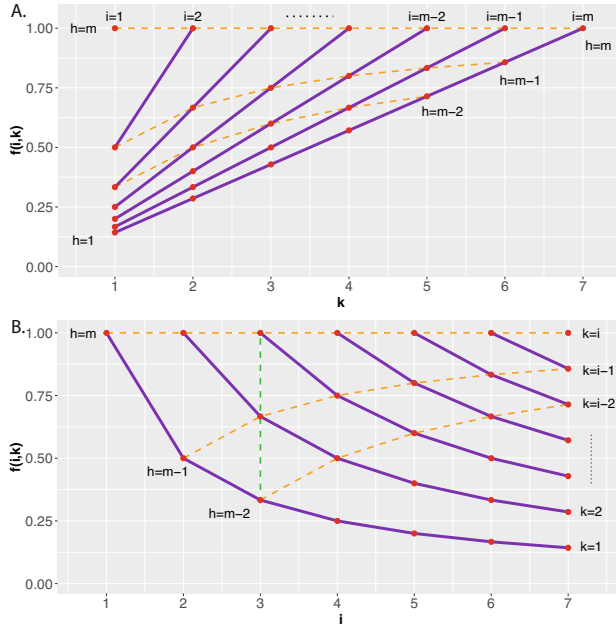


Fig. 15.5 The factor $f(i, k)$ of the Hommel correction is shown as a function of i and k . (a) The blue lines correspond to fixed values of i . (b) The blue lines correspond to fixed values of k . The index h is the argument of the ordered p -values, i.e., $p_{(h)}$

$$\begin{array}{ccccccc}
 \text{Step :} & 1 & 2 & \dots & m-1 & m \\
 i : & m & m-1 & \dots & 2 & 1 \\
 p_{(m)} & > F(m, m) & p_{(m)} > F(m-1, m-1) & \dots & p_{(m)} > F(2, 2) & p_{(m)} > F(1, 1) \\
 p_{(m-1)} & > F(m, m-1) & p_{(m-1)} > F(m-1, m-2) & \dots & p_{(m-1)} > F(2, 1) & \\
 \vdots & & \vdots & & & \\
 p_{(3)} & > F(m, 3) & p_{(3)} > F(m-1, 2) & & & \\
 p_{(2)} & > F(m, 2) & p_{(2)} > F(m-1, 1) & & & \\
 p_{(1)} & > F(m, 1) & & & &
 \end{array} \tag{15.59}$$

As one can see, from the triangular-shaped array, the number of conditions per step decreases by one. Specifically, always the smallest p -value from the previous step is dropped. This increased the probability for all conditions to hold from one step to the next since the corresponding values of $F(i, k)$ increase too. Specifically, $F(c, d) < F(c-1, d)$ hold for all c since

$$\frac{d\alpha}{c} < \frac{d\alpha}{c-1} \tag{15.60}$$

holds for all c . Hence, the smallest p -values tested per step are equivalent to stringent decreasing conditions.

In algorithmic form, one can write the Hommel correction as shown in Algorithm 12. In this form, the Hommel correction is less compact but easier to understand.

Algorithm 12: Hommel correction procedure

Input: $p_{(1)} \leq p_{(2)} \cdots \leq p_{(m)}$
 $i = m$
while $p_{(m-i+k)} \leq \frac{k}{i} \alpha$ for at least one $k \in \{1, \dots, i\}$ **do**
 $i = i - 1$
 $i^* = i$
if $i^* = 0$ **then**
 Reject $H_{(1)}, H_{(1)}, \dots, H_{(m)}$
else
 Reject $H_{(l)}$ with $p_l < \frac{\alpha}{i^*}$

It has been found that the Hommel procedure is more powerful than Bonferroni, Holm, and Hochberg [16]. Finally, we want to mention that recently, a much faster computational realization of the Hommel procedure has been found [329]. This algorithm has a linear time complexity and leads to an astonishing improvement allowing the application to millions of tests.

15.5.5.1 Examples

Let's consider some numerical examples for $m = 5$. In this case, the general array in Eq. 15.61 assumes the following numerical values.

$$\begin{array}{rccccc}
 \text{Step :} & 1 & 2 & 3 & 4 & 5 \\
 i : & 5 & 4 & 3 & 2 & 1 \\
 p_{(5)} & > 0.05 & p_{(5)} & > 0.05 & p_{(5)} & > 0.05 & p_{(5)} & > 0.05 & p_{(5)} & > 0.05 \\
 p_{(4)} & > 0.04 & p_{(4)} & > 0.037 & p_{(4)} & > 0.033 & p_{(4)} & > 0.025 & & \\
 p_{(3)} & > 0.03 & p_{(3)} & > 0.025 & p_{(3)} & > 0.016 & & & & \\
 p_{(2)} & > 0.02 & p_{(2)} & > 0.012 & & & & & & \\
 p_{(1)} & > 0.01 & & & & & & & &
 \end{array} \tag{15.61}$$

- Example 1: $p_{(1)} = 0.011, p_{(2)} = 0.021, p_{(3)} = 0.031, p_{(4)} = 0.41, p_{(5)} = 0.051$. In this case, $i^* = 5$ and $\alpha/i^* = 0.01$. From this, it follows that no hypothesis can be rejected.

- Example 2: $p_{(1)} = 0.009$, $p_{(2)} = 0.021$, $p_{(3)} = 0.031$, $p_{(4)} = 0.41$, $p_{(5)} = 0.051$. In this case, $i^* = 4$ and $\alpha/i^* = 0.0125$. From this, it follows that $H_{(1)}$ can be rejected.
- Example 3: $p_{(1)} = 0.009$, $p_{(2)} = 0.021$, $p_{(3)} = 0.024$, $p_{(4)} = 0.41$, $p_{(5)} = 0.051$. In this case, $i^* = 3$ and $\alpha/i^* = 0.016$. From this, it follows that $H_{(1)}$ can be rejected.

These examples should demonstrate that the application and outcome of a Hommel correction is nontrivial.

15.5.6 Westfall-Young Procedure

For most real-world situations, the joint distribution of the test statistics is unknown. Westfall and Young made seminal contributions by showing that, in this case, resampling-based methods can be used under certain conditions, to estimate p -values without many theoretical assumptions [501]. However, in order to do this, one needs (1) to access the data and (2) to be able to resample the data such that the resulting permutations allow to estimate the null hypotheses for the test statistics. The latter is usually possible for two-sample tests but may be more involved for other types of tests.

In particular, four such permutation-based methods have been introduced by Westfall and Young [501]. Two of these are *single-step* procedures, and two are *step-down* procedures. The single-step procedures are called *single-step minP*

$$\tilde{p}_j = \Pr \left(\min_{l \in \{1, \dots, m\}} P_l \leq p_j | H_0^C \right), \quad (15.62)$$

and *single-step maxT*

$$\tilde{p}_j = \Pr \left(\max_{l \in \{1, \dots, m\}} |T_l| \geq t_j | H_0^C \right), \quad (15.63)$$

and their adjusted p -values are given by Eqs. 15.62 and 15.63. Here, H_0^C is an intersection of all true null hypotheses, P_l denotes unadjusted p -values from permutations, and T_l denotes test statistics from permutations. The p_j and t_j are the p -values and test statistics from the non-permuted data, respectively.

Without additional assumptions, single-step maxT and single-step minP provide a weak control of the FWER. However, for subset pivotality, both procedures control the FWER strongly [501]. Here, subset pivotality is a property of the distribution of raw p -values, which holds if all subsets of p -values have an identical joint distribution under the complete null distribution [119, 500, 501] (for a discussion of an alternative and practically simpler sufficient condition, see [194]). Furthermore, the results from single-step maxT and single-step minP are similar when the test statistics are identically distributed [121].

From a computational perspective, the single-step minP is more demanding than the single-step maxT because it is based on p -values and not on test statistics. The difference is that one can get a resampled value of a test statistic from one resampled data set, whereas for a p -value, one needs a distribution of resampled test statistics, which can only be obtained from many resampled data sets. This has been termed *double permutation* [177].

The step-down procedures are called *step-down minP*

$$\tilde{p}_{r_j} = \max_{k \in \{1, \dots, j\}} \left\{ \Pr \left(\min_{l \in \{k, \dots, m\}} P_{r_l} \leq p_{r_k} | H_0^C \right) \right\}, \quad (15.64)$$

and *step-down maxT*

$$\tilde{p}_{s_j} = \max_{k \in \{1, \dots, j\}} \left\{ \Pr \left(\max_{l \in \{k, \dots, m\}} |T_{s_l}| \geq |t_{s_k}| | H_0^C \right) \right\}, \quad (15.65)$$

and their adjusted p -values are given by Eqs. 15.64 and 15.65. The indices r_k and s_k are the ordered indices, i.e., $|t_{s_1}| \geq |t_{s_2}| \geq \dots \geq |t_{s_m}|$ and $p_{r_1} \leq p_{r_2} \leq \dots \leq p_{r_m}$.

Interestingly, it can be shown that when the P_l are uniformly distributed in $[0, 1]$, the p -values in Eq. 15.64 correspond to those obtained from the Holm procedure [177]. That means, in general, the step-down minP procedure is less conservative compared to the Holm's procedure. Again, the step-down minP is computationally more demanding compared to the step-down maxT due to the required double permutations. Also, assuming the subset pivotality, both procedures have a strong control of the FWER [399].

The general advantage of using maxT and minP procedures over all the other procedures, discussed in this chapter, is their potential use of the dependency structure among the test statistics. That means, when such a dependency (correlation) is absent, there is no apparent need for using these procedures. However, most data sets may have some kind of dependency since the associated covariates are usually connected with each other. In such situations, the maxT and minP procedures can lead to an improved power.

In algorithmic form, the step-down maxT and step-down minP procedure can be formulated as shown in Algorithm 13 and 14. For Step 2 in Algorithm 14, it is of the raw p -value $p_{i,b}$ obtained using the same permutations from Step 1.

Algorithm 13: Westfall-Young step-down maxT procedure.

Input: $|t_{s_1}| \geq |t_{s_2}| \cdots \geq |t_{s_m}|$, and the data matrix X **for** $b \in \{1, \dots, B\}$ **do** Step 1. Permute the columns of the data matrix X Step 2: Calculate the test statistics $t_{i,b}$ for $i \in \{1, \dots, m\}$ Step 3: Estimate $u_{i,b}$ for $i \in \{1, \dots, m\}$ by

$$\begin{aligned} u_{m,b} &= |t_{s_m,b}| \\ u_{i,b} &= \max\{u_{i+1,b}, |t_{s_i,b}|\} \quad \text{for } i \in \{m-1, \dots, 1\} \end{aligned}$$

Estimate the adjusted p-values by

$$\hat{p}_{s_i}^* = \frac{\#\{b : u_{i,b} \geq |t_{s_i}|\}}{B} \quad \text{for } i \in \{1, \dots, m\} \quad (15.66)$$

Finally, monotonicity is enforced by

$$\begin{aligned} \hat{p}_{s_1} &= \hat{p}_{s_1}^* \\ \hat{p}_{s_i} &= \max\{\hat{p}_{s_{i-1}}, \hat{p}_{s_i}^*\} \quad \text{for } i \in \{2, \dots, m\} \end{aligned}$$

Algorithm 14: Westfall-Young step-down minP procedure.

Input: $p_{(1)} \leq p_{(2)} \cdots \leq p_{(m)}$, and the data matrix X **for** $b \in \{1, \dots, B\}$ **do** Step 1. Permute the columns of the data matrix X Step 2: Estimate the raw p-values $p_{i,b}$ for $i \in \{1, \dots, m\}$ Step 3: Estimate $q_{i,b}$ for $i \in \{1, \dots, m\}$ by

$$\begin{aligned} q_{m,b} &= p_{r_m,b} \\ q_{i,b} &= \min\{q_{i+1,b}, p_{r_i,b}\} \quad \text{for } i \in \{m-1, \dots, 1\} \end{aligned}$$

Estimate the adjusted p-values by

$$\hat{p}_{r_i}^* = \frac{\#\{b : q_{i,b} \leq p_{r_i}\}}{B} \quad \text{for } i \in \{1, \dots, m\} \quad (15.67)$$

Finally, monotonicity is enforced by

$$\begin{aligned} \hat{p}_{r_1} &= \hat{p}_{r_1}^* \\ \hat{p}_{r_i} &= \max\{\hat{p}_{r_{i-1}}, \hat{p}_{r_i}^*\} \quad \text{for } i \in \{2, \dots, m\} \end{aligned}$$

15.6 Controlling the FDR

Now, we come to a second type of correction methods. In contrast with the methods discussed so far, for controlling the FWER, the methods we are discussing in the next sections are for controlling the FDR. That means, these methods have a different optimization goal. In Sect. 15.8, we present a summary of the underlying assumptions the procedures rely on.

15.6.1 Benjamini-Hochberg Procedure

The first method from this category of procedures for controlling the FDR is called the Benjamini-Hochberg (BH) procedure [32]. The BH procedure can be considered as a breakthrough because it introduced a novel way of thinking to the community. The procedure assumes ordered p -values as in Eq. 15.46. Then, it identifies, using a step-up procedure, the largest index k such that

$$p_{(i)} \leq i \frac{\alpha}{m} \quad (15.68)$$

holds, and it rejects the null hypotheses $H_{(1)}, H_{(2)}, \dots, H_{(k)}$. This can be formulated in the following compact form:

$$k = \max \{i \in \{1, \dots, m\} | p_{(i)} \leq i \frac{\alpha}{m}\}. \quad (15.69)$$

If no such index exists, then no hypothesis is rejected.

Conceptually, the BH procedure utilizes Simes inequality [435]; see Sect. 15.2.1. In algorithmic form, the BH procedure can be formulated as shown in Algorithm 15.

Algorithm 15: SU Benjamini-Hochberg procedure

Input: $p_{(1)} \leq p_{(2)} \leq \dots \leq p_{(m)}$

$k = 0$

while $p_{(k+1)} \leq ((k+1)\alpha)/m$ **do**

$k = k + 1$

Reject $H_{(1)}, H_{(1)}, \dots, H_{(k)}$

The adjusted p -values of the BH procedure [121] are given by

$$p_{(i)}^{adj} = \min_{j \in \{1, \dots, m\}} \left\{ \frac{mp_{(j)}}{j}, 1 \right\}. \quad (15.70)$$

In general, the BH procedure makes a good trade-off between false positives and false negatives and works well for independent test statistics or positive regression dependencies (denoted PRDS), which is a weaker assumption compared to independence [34, 161, 414]. Generally, it is also more powerful than procedures for controlling the FWER. The correlation assumptions imply that in the presence of negative correlations, the control of the FDR is not always achieved. The BH procedure can also suffer from a weak power, especially when testing a relatively small number of hypotheses because in such a situation it is similar to a Bonferroni correction; see Fig. 15.6b.

15.6.1.1 Example

In Fig. 15.6, we show a numerical example for the Benjamini-Hochberg procedure. In Fig. 15.6a, we show the rank ordered p -values for $m = 100$. The dashed red line corresponds to a significance level of $\alpha = 0.05$, and the dashed green line corresponds to the testing condition in Eq. 15.68.

In Fig. 15.6b, we zoom into the first 30 p -values. Here, we added a Bonferroni correction, depicted by the dashed orange line at a value of $\alpha/m = 5e - 04$. One can see that the BH correction corresponds to a straight line that is always above the Bonferroni correction. Hence, a BH is always less conservative than a Bonferroni correction. As a result, for the shown p -values, we obtain 18 significant values for the BH correction but only 3 significant values for the Bonferroni correction. One can also see that using the uncorrected p -values with $\alpha = 0.05$ gives additional significant values in an uncontrolled manner beyond rank 18.

15.6.2 Adaptive Benjamini-Hochberg Procedure

A modified version of the BH procedure, which estimates the proportion of null hypothesis from data whereas the proportion of true null hypotheses is given by $\pi_0 = m_0/m$, was introduced in [33]. For this reason, this procedure is called adaptive Benjamini-Hochberg procedure (adaptive BH).

The adaptive BH procedure modifies Eq. 15.68 by substituting α with α/π_0 , which gives

$$p_{(i)} \leq i \frac{\alpha}{\pi_0 m} = i \frac{\alpha}{m_0}. \quad (15.71)$$

The procedure itself searches, in a step-up manner, the largest index k such that

$$k = \max \left\{ i \in \{1, \dots, m\} \mid p_{(i)} \leq i \frac{\alpha}{\pi_0 m} \right\}. \quad (15.72)$$

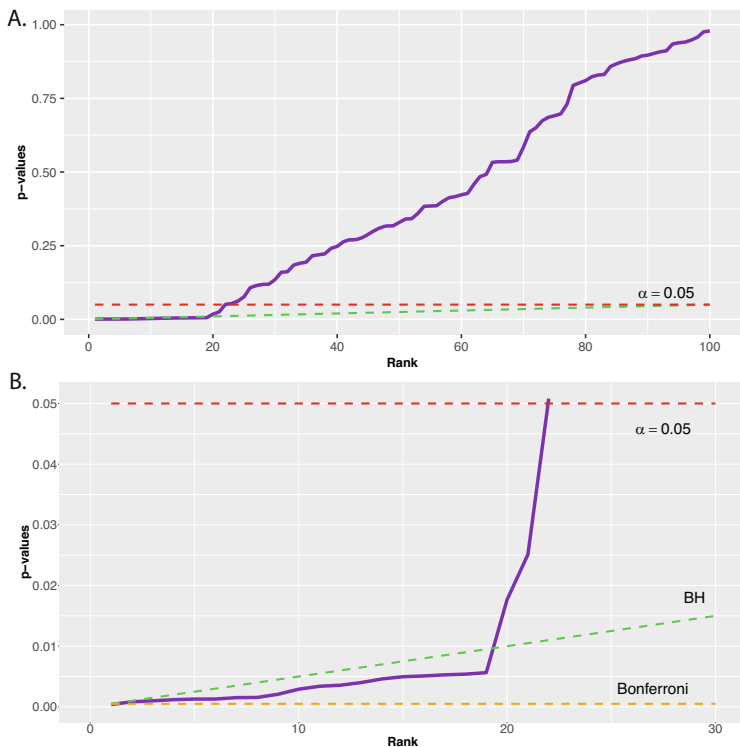


Fig. 15.6 Example for the Benjamini-Hochberg procedure. The dashed green line corresponds to the critical values given by Eq. 15.68. (a) Results for $m = 100$. (b) Zooming into the first 30 p -values

If no such index exists, then no hypothesis is rejected; otherwise, the null hypotheses, $H_{(1)}, H_{(2)}, \dots, H_{(k)}$, are rejected.

The estimator for π_0 is found as a result of an iterative search [309] based on

$$\hat{\pi}_0^{BH}(k) = \frac{m - k + 1}{(1 - p_{(k)})m}. \quad (15.73)$$

Specifically, the optimal index k is found from

$$k = \min \{i \in \{2, \dots, m\} | \hat{\pi}_0^{BH}(i) > \hat{\pi}_0^{BH}(i - 1)\}. \quad (15.74)$$

The importance of this study does not lie in its practical usage but in the inspiration it provided for many follow-up approaches that introduced new estimators for π_0 . In this chapter, we will provide some examples for this, e.g., when discussing the BR-2S procedure and in the summary Sect. 15.8.

15.6.3 Benjamini-Yekutieli Procedure

In order to improve the BH procedure that can deal with a dependency structure, a modification, called the Benjamini-Yekutieli (BY) procedure, was introduced in [34]. The BY procedure assumes also ordered p -values as in Eq. 15.46, and then it identifies, in a step-wise procedure, the largest index k such that

$$p_{(k)} \leq k \frac{\alpha}{mf(m)} \quad (15.75)$$

holds, and it rejects the null hypotheses $H_0^{(1)}, H_0^{(2)}, \dots, H_0^{(k)}$. It is important to note that here the factor $f(m) = \sum_{i=1}^m 1/i$, which depends on the total number of hypotheses, is introduced. This can be formulated in the following compact form:

$$k = \max \left\{ i \in \{1, \dots, m\} \mid p_{(i)} \leq i \frac{\alpha}{mf(m)} \right\}. \quad (15.76)$$

If no such index exists, then no hypothesis is rejected.

Since $f(m) > 1$ for all m , the product $mf(m)$ can be seen as an effective increase in the number of hypotheses to $m' = mf(m)$. Hence, the BY procedure is very conservative, and it can be even more conservative than a Bonferroni correction. For instance, for $m \in \{100, 1000, 10,000, 100,000\}$, we obtain $f(m) = \{5.8, 7.5, 9.8, 12.1\}$. The adjusted p -values of the BY procedure [121] are given by

$$p_{(i)}^{adj} = \min_{k \in \{i, \dots, m\}} \left\{ \frac{mf(m)p_{(k)}}{k}, 1 \right\}. \quad (15.77)$$

It has been proved that the BY procedure controls the FDR in the strong sense by

$$FDR \leq \frac{m_0}{m} \alpha = \pi_0 \alpha, \quad (15.78)$$

for any type of dependent data [34]. Since $m_0 \leq m$ always, the FDR is controlled either at level α (for $m_0 = m$) or even below that level. A disadvantage of the BY procedure is that it is less powerful than BH.

15.6.3.1 Example

In Fig. 15.7, we show a numerical example for the Benjamini-Yekutieli procedure. Here, the BY correction corresponds to the dashed red line, which is always below the BH correction (dashed green line) indicating that it is more conservative.

Interestingly, the line for the BY correction intersects with the Bonferroni correction (dashed orange line) at rank 5 (see inset). That means, below this

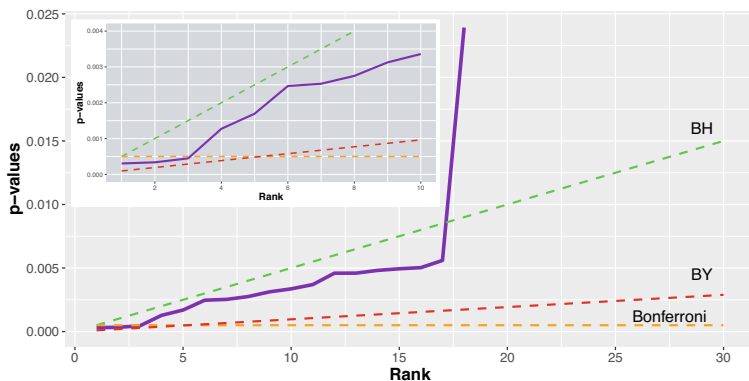


Fig. 15.7 Example of the Benjamini-Yekutieli procedure for $m = 100$. Both figures show only a subset of the results up to rank 30, respectively, 10 in order to see the effect of a BY correction

value, the BY correction is more conservative, and it is less conservative after the intersection. For the p -values in this example, the BY gives no significant results. This indicates the potential problem with the BY procedure in practice because its conservativeness can lead to no significant results at all.

15.6.4 Benjamini-Krieger-Yekutieli Procedure

Yet another modification of the BH procedure was introduced in [35]. This MTP is an adaptive two-stage linear step-up method, called BKY (Benjamini-Krieger-Yekutieli). Here, “adaptive” means that the procedure estimates the number of null hypothesis from the data and uses this information to improve the power. This approach is motivated by Eq. 15.78 and the dependency of the control on m_0 .

Step 1 Use a BH procedure with $\alpha' = \alpha/(1-\alpha)$. Let r be the number of hypotheses rejected. If $r = 0$, no hypothesis is rejected. If $r = m$, all m hypotheses are rejected. In both cases, the procedure stops. Otherwise proceed.

Step 2 Estimate the number of null hypotheses by $\hat{m}_0 = m - r$.

Step 3 Use a BH procedure with $\alpha'' = m\alpha'/(\hat{m}_0) = \alpha'/\hat{\pi}_0$.

The BKY procedure utilizes the BH procedure twice: to estimate the number of null hypotheses $\hat{m}_0$ in the first stage and to declare significance in the second stage.

The BKY procedure controls the FDR exactly at level α when tests are independent. In [35], it has been shown that this procedure has higher power than BH.

15.6.5 Blanchard-Roquain Procedure

A generalization of the Benjamini-Yekutieli procedure was introduced by Blanchard and Roquain [43].

15.6.5.1 BR-1S Procedure

The first procedure, introduced in [43], is a one-state adaptive step-up procedure, called BR-1S, independently proposed in [414]. Formally, the BR-1S procedure [43], first, defines an adaptive threshold by

$$t_{(i)} = \min \left\{ \lambda, \frac{i\alpha(1-\lambda)}{m-i+1} \right\}, \quad (15.79)$$

for $\lambda \in (0, 1)$ and for all $i \in \{1, \dots, m\}$. Then, the largest index k is determined as follows:

$$k = \max \{i \in \{1, \dots, m\} | p_{(i)} \leq t_{(i)}\}. \quad (15.80)$$

If no such index exists, then no hypothesis is rejected; otherwise, all the null hypotheses with p -values such that $p_{(i)} \leq t_{(k)}$ are rejected.

For the BR-1S procedure, it has been proved that the FDR is controlled by

$$\text{FDR} \leq \min \left\{ \lambda, \alpha(1-\lambda)m \right\}. \quad (15.81)$$

A brief calculation shows that both arguments of the above equations are equal, for

$$\lambda(m) = \frac{\alpha m}{1 + \alpha m}. \quad (15.82)$$

A further calculation shows that Eq. 15.82 is monotonously increasing for increased values of m , and for $m \geq 2$, we find $\lambda(m) > \alpha$. That means, one needs to choose λ values smaller than the value on the right-hand side, in Eq. 15.82, to be able to control the FDR [43]. Hence, a common choice for λ , in Eq. 15.81, is $\lambda = \alpha$ because this controls the FDR on the α level, i.e., $\text{FDR} \leq \alpha$.

For $\lambda = \alpha$, the adaptive threshold simplifies and becomes

$$t_{(i)} = \alpha \min \left\{ 1, \frac{i(1-\alpha)}{m-i+1} \right\}. \quad (15.83)$$

For $i \leq (m+1)/2$, the adaptive threshold simplifies even further to

$$t_{(i)} = \alpha \frac{i(1-\alpha)}{m-i+1}. \quad (15.84)$$

15.6.5.2 BR-2S Procedure

The second procedure, introduced in [43], is a two-state adaptive plug-in procedure, called BR-2S, given by:

Stage 1 Estimate $R(\lambda_1) = m_0$ by BR-1S(λ_1).

Stage 2 Use $\alpha' = \alpha/\hat{\pi}_0$ with

$$\hat{\pi}_0^{BR} = \frac{m - R(\lambda_1) + 1}{(1 - \lambda_2)m} \quad \text{for } \lambda_2 \in (0, 1) \quad (15.85)$$

in the step-up procedure given by Eq. 15.72. That means, the estimate for the proportion of null hypotheses is used to find the largest index k such that

$$k = \max \left\{ i \in \{1, \dots, m\} \mid p_{(i)} \leq i \frac{\alpha}{\hat{\pi}_0^{BR} m} \right\}. \quad (15.86)$$

If no such index exists, then no hypothesis is rejected; otherwise, the null hypotheses, $H_{(1)}, H_{(2)}, \dots, H_{(k)}$, are rejected.

The BR-2S procedure depends on two parameters denoted λ_1 and λ_2 . The first parameter is for BR-1S in stage one, whereas the second is used to estimate the proportion of null hypotheses in stage two. It has been proved in [43] that, setting $\lambda_1 = \alpha/(1 + \alpha + 1/m)$ in step 1 of the BR-2S procedure, one obtains $\text{FDR} = \lambda$. This suggests to set $\lambda = \alpha$ in step 2. The BR-1S and BR-2S procedures are proven to control the FDR for arbitrary dependence.

15.7 Computational Complexity

When performing MTCs for high-dimensional data, the computation time required by a procedure can have an influence on its selection. For this reason, we present in this section a comparison of the computation time for different methods depending on the dimensionality of the data.

In the following, we apply the seven MTPs, namely, Bonferroni, Holm, Hochberg, Hommel (two times), Benjamini-Hochberg, and Benjamini-Yekutieli, to p -values of varying size for $m \in \{100, 20000, 50000\}$. In Table 15.1 we show the mean computation times in seconds averaged over *ten* independent runs.

One can see that there are large differences in the computation times. By far, the slowest method is Hommel. For instance, correcting $m = 50,000$ p -values takes over 400 times longer compared to a Bonferroni correction. This method has also the worst scaling, which means practical applications need to take this into consideration. This computational complexity could already be anticipated from our discussion in Sect. 15.5.5 because the Hommel correction is much more involved compared to all the other procedures. However, the new algorithm in

Table 15.1 Computational analysis for seven MTPs. Shown are the average computation times in seconds for m tests

Method	Error control	$m = 100$	$m = 20,000$	$m = 50,000$
Bonferroni	FWER	4.997253e-05	6.375313e-04	0.003246069
Holm	FWER	8.559227e-05	1.908016e-03	0.004300451
Hochberg	FWER	7.801056e-05	1.903248e-03	0.005757761
Hommel	FWER	1.836157e-03	1.432532e+01	1.389162673
Benjamini-Hochberg	FDR	9.551458e-05	1.955366e-03	0.004550409
Benjamini-Yekutieli	FDR	8.130074e-05	2.041864e-03	0.004711628
Hommel*	FWER	2.599716e-04	1.967573e-03	0.004542589

[329] (indicated by *) leads to an astonishing improvement in the computational complexity for this method.

Furthermore, from Table 15.1, there are essentially two groups of computational times. In the group of the fastest methods, we have Bonferroni, Holm, Hochberg, Benjamini-Hochberg, and Benjamini-Yekutieli, and the group of slowest methods includes only Hommel. As one can see, applying MTPs to tens of thousands of hypothesis (p -values) is feasible without problems.

15.8 Comparison

In this chapter, we discussed many multiple testing procedures for controlling the FWER and the FDR, and we categorized them as:

1. Single-step vs stepwise approaches
2. Adaptive vs nonadaptive approaches
3. Marginal vs joint multiple testing procedures

When it comes to the practical application of a MTP, one needs to realize that to select a method, there is more than the control of an error measure. Specifically, while a given MTP may guarantee the control of an error measure, e.g., the FWER or the FDR, this does not inform us about the Type 2 error/power of the procedure. Especially, the latter is important for practical applications because if one cannot reject any null hypothesis, there is usually nothing to explore.

In order to find the optimal procedure for a given problem, the best approach is to conduct simulation studies and compare different MTPs. Specifically, for a given data set, one can diagnose its characteristics, e.g., by estimating the presence and the structure of correlations, and then simulate data following these characteristics. This ensures that the simulations are problem-specific and adopt the characteristics of the data as close as possible.

The advantage of this approach is that the selection of an MTP is not based on generic results from the literature but tailored to your problem. The disadvantage is

Table 15.2 Summary of MTC procedures. PRDS stands for positive regression dependencies

Method	Error control	Procedure type	Error control type	Correlation assumed
Šidák	FWER	Single-step	Strong	Non-negative
Šidák	FWER	Step-down	Strong	Non-negative
Bonferroni	FWER	Single-step	Strong	Any
Holm	FWER	Step-down	Strong	Any
Hochberg	FWER	Step-up	Strong	PRDS
Hommel	FWER	Step-down	Strong	PRDS
maxT	FWER	Single-step	Strong	Subset pivotality
minP	FWER	Single-step	Strong	Subset pivotality
maxT	FWER	Step-down	Strong	Subset pivotality
minP	FWER	Step-down	Strong	Subset pivotality
Benjamini-Hochberg	FDR	Step-up	Strong	PRDS
Benjamini-Yekutieli	FDR	Step-up	Strong	Any
Benjamini-Krieger-Yekutieli	FDR	Step-up	Strong	Independence
BR-1S	FDR	Step-up	Strong	Any
BR-2S	FDR	Two-stage	Strong	Any

the effort it takes to estimate, simulate, and compare the different procedures with each other.

If such a simulation approach is not feasible, one needs to revert to results from the literature. In Table 15.2, we show a summary of MTPs and some important characteristics. Furthermore, from a multitude of simulation studies, the following results have been found independently:

- Positive correlations (simulated data): BR is more powerful than BKY [43].
- General correlations (real data): BY has a higher PPV compared to BH [286].
- Positive correlations (simulated data): BKY is more powerful than BH [35].
- Positive correlations (simulated data): Hochberg, Holm, and Hommel do not control the PFER for high correlations [170].
- General correlations (real data): SS MaxT has higher power compared to SS minP [121, 306, 499].
- General correlations (real data): SS MaxT and SD MaxT can be more powerful than Bonferroni, Holm, and Hochberg [121].
- Random correlations (simulated data): SD minP is more powerful than SD maxT [306].

The abovementioned simulation studies considered all the correlation structure in the data because this is of practical relevance. Since there is more than one type of correlation structure, the possible number of different studies to consider

for all these different structures is huge. Specifically, one can assume homogenous or heterogeneous correlations. The former assumes that pair-wise correlations are equal throughout the different pairs, whereas the latter assumes inequality. For heterogeneous correlation structures, one can further assume a random structure or a particular structure. For instance, a particular structure can be imposed from an underlying network, e.g., a gene regulatory network among genes [106] (see also Chap. 5). Hence, for the simulation of such data, the covariance structure needs to be consistent with a structure of the underlying network [148].

15.9 Summary

In this chapter, we discussed multiple testing corrections (MTCs) because they provide important extensions to the framework of hypothesis testing (discussed in Chap. 10) [182, 331, 384, 402]. As we have seen, there is a large number of different methodological correction procedures allowing the control of the FWER or the FDR [60, 112, 386, 447]. However, especially for correlated data, many more methods can be expected in the coming years because high-dimensional data are usually correlated, at least to some extent, and there is a great potential to further improve those methods using tailored MTCs.

Learning Outcome 15: Multiple Testing Corrections

A multiple testing correction procedure aims to control the Type 1 error (FWER or FDR) and at the same time tries to maximize the power of the test.

In this chapter, we categorized MTPs as (1) single-step vs stepwise approaches, (2) adaptive vs nonadaptive approaches, and (3) marginal vs joint multiple testing procedures. While single-step procedures apply the same constant correction to each test, step-wise procedures are variable in their correction, and decisions depend also on previous steps. Furthermore, the latter procedures are based on rank-ordered p -values of the tests, and they inspect them in either decreasing (step-down) or increasing (step-up) order of their significance. To fully explain each of these concepts, we discussed procedures from all categories. Specifically, we discussed single-step corrections of Šidák [434], Bonferroni [50], and Westfall and Young [501]; step-wise procedures of Holm [245], Hochberg [238], Benjamini and Hochberg [32], Benjamini-Yekutieli [34], and Westfall and Young (maxT and minP) [501]; and the multistage procedure of Benjamini-Krieger-Yekutieli [35].

15.10 Exercises

1. Identify a problem in science or industry that requires the testing of multiple hypotheses. Discuss for this problem the difference of testing one and multiple hypotheses.
2. Reproduce the results shown in Fig. 15.2.
 - a. Calculate $\text{FWER} = Pr(\text{reject at least one } H_0 | \text{all } H_0 \text{ are true})$ for $\alpha = 0.05$.
 - b. Calculate $\text{FWER} = Pr(\text{reject at least one } H_0 | \text{all } H_0 \text{ are true})$ for $\alpha = 0.001$.
 - c. Calculate $\text{FWER} = Pr(\text{reject at least one } H_0 | \text{all } H_0 \text{ are true})$ for $\alpha = 0.0001$.
3. Discuss the Holm procedure summarized by Algorithm 10 by giving a numerical example.
4. Suppose we perform *ten* hypothesis tests (e.g., testing the effect of *ten* marketing campaigns compared to the current strategy) and obtain the following p -values:

0.0140, 0.2960, 0.9530, 0.0031, 0.1050, 0.6410, 0.7810, 0.9010, 0.0053, 0.4500
 - a. Which hypotheses are rejected when applying a Bonferroni correction?
 - b. Order the p -values.
 - c. Which hypotheses are rejected when applying a Holm correction?
5. Discuss the Benjamini-Hochberg procedure summarized by Algorithm 15 by giving a numerical example.

Chapter 16

Survival Analysis



16.1 Introduction

The term survival analysis comprises a collection of longitudinal analysis methods for studying *time-to-event data*. Here, the term “time” corresponds to the duration until the occurrence of a particular event, whereas an “event” is a special incident that assumes an application-specific meaning, e.g., death, heart attack, wear-out or failure of a product or equipment, divorce, violation of parole, or bankruptcy of a company—to name just a few. It is this diversity in the meaning of “event” that makes survival analysis widely applicable to many problems in various fields. For instance, survival analysis is frequently utilized in biology, medicine, engineering, marketing, social sciences, or behavioral sciences [8, 12, 61, 136, 210, 268, 351, 443, 522]. This interdisciplinary usage led also to many alternative names for the field used synonymously. For this reason, survival analysis is also known as *event history analysis* (social sciences), *reliability theory* (engineering), or *duration analysis* (economics).

There are two approaches that contributed crucially to the development of modern survival analysis. The pioneers of the first approach are Kaplan and Meier, who introduced an estimator for survival probabilities [271]. The second approach was put forward by Cox, who introduced what is nowadays called the Cox proportional hazard model (CPHM) [86], which is a regression model.

In this chapter, we discuss the theoretical basics for survival analysis, including estimators for survival and hazard functions, and the comparison of survival curves. We discuss the Cox proportional hazard model (CPHM) in detail, as well as approaches for testing the proportional hazard (PH) assumption. Furthermore, we discuss stratified Cox models for cases where the PH assumption does not hold. We will see that there are cross-links to previously discussed topics in this book, e.g., linear regression discussed in Chap. 11 and statistical hypothesis testing discussed in Chap. 10. In fact, the CPHM is a regression model, and the comparison of survival curves is conducted with the help of statistical hypothesis tests.

We start this chapter with a motivation for the need of survival analysis, by discussing two examples. The first is about the effect of chemotherapy on patients and second is about the effect of medication on schizophrenia patients.

16.2 Motivation

In order to develop an intuitive understanding of survival analysis, let us discuss its basic principles. When we speak about *survival*, we mean in fact probabilities, which means that survival can be “measured” as a probability. Importantly, survival relates to the membership in a group, where a group consists of a number of subjects, and the survival probability is associated with each subject in this group. The membership in a group is not constant; therefore it can change. Such a change of a membership is initiated by an event. Particular examples of events are:

- Death
- Relapse/recurrence
- Infection
- Suicide
- Agitation attack
- Crime
- Violation of parole
- Divorce
- Graduation
- Bankruptcy
- Malfunctioning or failure of a device
- Purchase

The event “death” is certainly the most severe example that can be given, which explains also intuitively the name survival analysis. Importantly, survival analysis is not limited to medical problems but can also be applied to problems in social sciences, engineering, or marketing, as illustrated by the examples in the above list.

16.2.1 *Effect of Chemotherapy: Breast Cancer Patients*

In [310], the authors investigated the effect of neoadjuvant chemotherapy on triple-negative breast cancer (TNBC) patients. TNBC is characterized by the lack of expression of three genes, namely, estrogen receptor (ER), progesterone receptor (PR), and human epidermal growth factor receptor 2 (HER2). To compare the survival time of TNBC patients with non-TNBC, the time was measured from surgery (mastectomy) to death. As a result, the authors found that patients with TNBC have a decreased survival compared to non-TNBC patients.

16.2.2 Effect of Medication: Agitation

In [303], the authors studied the effect of medications on individuals with schizophrenia. Due to the complexity of this neuronal disorder, it is rather difficult or even impossible to judge, from observing such individuals, how long the effect of a medication lasts or the onset of an attack. Hence, measuring “time to an attack” is, in this case, nontrivial because it is not directly observable. To accomplish this task, the authors used the following experimental design for the study. At a certain time, the patients are using either medication or a placebo administered by an inhaler. Then, the patients are not allowed to reuse the inhaler for 2 h. After the 2 h, everyone could use the inhaler as required. This allowed the easy measurement of the time between the first and second usage of the inhaler, which was then used as “time to event.” This was used to perform a survival analysis to assess the difference between the medication and the placebo.

In contrast to the breast cancer example above, the agitation example shows that the time to event is not for all problems easy to obtain but requires sometimes a clever experimental design that enables its measurement.

Taken together, survival analysis examines and models the time for events to occur, and changes of the survival probability over time. Practically, one needs to estimate these from data of subjects, which contains information about the time of events. A factor that further complicates the analysis is the incomplete information caused by *censoring*. Due to the central role of censoring for essentially all statistical estimators that will be discussed in the following, we discuss the problem associated with censoring in the next section.

16.3 Censoring

In order to perform a survival analysis, one needs to record the time to event t_i for the subjects $i \in \{1, \dots, N\}$ of a group. This establishes so-called **time-to-event data** (see Chap. 5) upon which a general survival analysis is based on. However, this is not always possible since we may have only partial information about the time to an event. In these cases, the data are called censored. Specifically, a patient has a censored survival time if the event has not yet occurred for this patient. This could happen when:

- A patient is a drop-out of a study, e.g., stops attending the clinic for follow-up examination.
- The study has a fixed timeline and the event occurs after the cutoff time.
- A patient withdraws from a study.

The above censoring instances are called *right censoring* [304]. In Fig. 16.1, we visualize the meaning of censoring. For instance, the subjects with the labels ID_A and ID_C experience the event within the duration of the study, and this is indicated

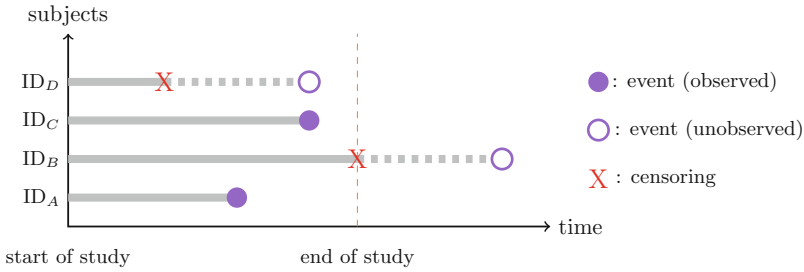


Fig. 16.1 A visualization for the meaning of right censoring

by a full blue circle. In contrast, the subject ID_B experiences the event after the end of the study, and this is indicated by an open blue circle. Therefore, this event is not observed during the study. The only useable (observable) information we have is that at the end of the study, subject ID_B did not yet experience the event. Hence, the survival time of subject ID_B is censored, as indicated by the red X. This means that until the censoring event occurred (indicated by the red X), subject ID_B did not experience the event. Also, for subject ID_D , we have a censored survival time; however, for a different reason, since the study did not end yet. A possible explanation for this censoring is that the subject did not attend follow-up visits after the censoring event occurred (indicated by the red X). Formally, the censoring for subject ID_B is termed a fixed right censoring, whereas the censoring for subject ID_D is called random, right censoring.

There are further censoring types that can occur. For instance, a left censoring occurs if the event is observed but not the beginning of the process. A typical example of left censoring is an infection, since it is usually diagnosed at some time, but it started before the diagnosis at an unknown time point. In the following, we will limit our focus to right-censored subjects. A summary of the different types of censoring is provided in [300]:

- Type I censoring: All subjects begin and end the study at the same time (fixed length of study). Examples of Type I censoring occur during laboratory experiments.
- Type II censoring: All subjects begin the study at the same time, but the study ends when a predetermined fixed number of subjects have experienced the event (flexible length of study). Examples of Type II censoring also occur during laboratory experiments.
- Type III censoring: The subjects enter the study at different times, but the length of the study is fixed. Examples of Type III censoring occur during clinical trials.

16.4 General Characteristics of a Survival Function

A survival curve, denoted $S(t)$, formulates the survival probability as a function of time (t). The function $S(t)$ is also referred to as the **survival function** or the survivor function. Formally, $S(t)$ is the probability that the random variable T is larger than a specified time t , i.e.,

$$S(t) = \Pr(T > t). \quad (16.1)$$

Since $S(t)$ is defined for a group of subjects, $S(t)$ can be interpreted as the proportion of subjects having survived till t . Therefore, a naive estimator for $S(t)$ is given by

$$S_{naive}(t) = \frac{\text{\#subjects surviving past } t}{N}, \quad (16.2)$$

whereas N is the total number of subjects. Equation 16.1 is the population estimate of a survival function.

Below, we will discuss various sample estimates of $S(t)$, which can be numerically evaluated from data. Put simply, the survival function gives the probability that a subject (represented by T) will survive past time t .

The survival function has the following properties:

- The range of time, t , is $[0, \infty)$.
- $S(t)$ is a non-increasing function, i.e., $S(t_1) \geq S(t_2)$ for $t_1 \leq t_2$.
- At time $t = 0$, $S(t) = 1$, i.e., the probability of surviving past time $t = 0$ is 1.

Since $S(t)$ is a probability distribution, then there exists a probability density f such that

$$S(t) = \int_t^{\infty} f(\tau) d\tau. \quad (16.3)$$

Hence, differentiating $S(t)$ with respect to t , we obtain

$$f(t) = -\frac{d S(t)}{dt}. \quad (16.4)$$

Furthermore, the expectation value of T is given by

$$\mu = \mathbb{E}[T] = \int_0^{\infty} t f(t) dt. \quad (16.5)$$

Using Eq. 16.4 and integrating by parts, it can be shown that the survival function, $S(t)$, can be used to obtain the mean life expectation:

$$\mu = \mathbb{E}[T] = \int_0^\infty S(t)dt. \quad (16.6)$$

16.5 Nonparametric Estimator for the Survival Function

In this section, we present two of the most popular nonparametric methods used to estimate the survival function $S(t)$.

16.5.1 Kaplan–Meier Estimator for the Survival Function

The Kaplan–Meier estimator [271] of a survival function, denoted $S_{KM}(t)$, is given by

$$S_{KM}(t) = \prod_{i:t_i < t} \frac{n_i - d_i}{n_i} = \prod_{i:t_i < t} \left(1 - \frac{d_i}{n_i}\right). \quad (16.7)$$

This estimator holds for all $t > 0$, and it only depends on the following two parameters:

- n_i : number of subjects at risk at time t_i
- d_i : number of events at time t_i

Here, n_i corresponds to the number of subjects present at time t_i . In contrast, subjects that experienced the event or censored are no longer present. The difficult part of this estimator is the argument of the product, which considers only events i that occur before time t , i.e., $t_i < t$. Hence, the survival curve $S_{KM}(t)$ at time t considers all events that happened before t .

It is important to realize that for evaluating the Kaplan–Meier estimator, only the events occurring at $\{t_i\}$ are important. That means, between two events, e.g., t_i and t_{i+1} , the survival curve is constant. This allows a simple reformulation of Eq. 16.7 to rewrite the Kaplan–Meier estimator using the following recursive relationship:

$$S_{KM}(t_k) = \frac{n_{k-1} - d_{k-1}}{n_{k-1}} S_{KM}(t_{k-2}). \quad (16.8)$$

In Fig. 16.2, we show an example of the evaluation of the Kaplan–Meier estimator using the recursive formulation given in Eq. 16.8. This example includes in total 5 subjects and 4 events.

H_0 : There is no difference in survival between (group 1) and (group 2).

H_1 : There is a difference in survival between (group 1) and (group 2).

The most popular tests for comparing survival curves are:

- log-rank test
- Wilcoxon (Gehan) test (a special case of a weighted log-rank test)

The difference between both tests is that a log-rank test has more power than a Wilcoxon test for detecting late differences in the survival curves, whereas a Wilcoxon test has more power than a log-rank test for detecting early differences.

16.6.1 Log-Rank Test

The log-rank test, sometimes called the Mantel-Haenszel log-rank test, is a nonparametric hypothesis test [324]. It makes the following assumptions.

- Censored and uncensored subjects have the same probability of the event (censoring is non-informative).
- Kaplan–Meier curves of the two groups must not intersect (proportional hazards assumption must hold).
- No particular distribution for the survival curve is assumed (distribution free).

Formally, the test is defined as follows. For each time t , estimate the expected number of events for (group 1) and (group 2) as follows:

$$e_{1t} = \left(\frac{n_{1t}}{n_{1t} + n_{2t}} \right) \times (m_{1t} + m_{2t}), \quad (16.12)$$

$$e_{2t} = \left(\frac{n_{2t}}{n_{1t} + n_{2t}} \right) \times (m_{1t} + m_{2t}), \quad (16.13)$$

where the indices “1” and “2” indicate group one and two, respectively; n_{it} and m_{it} are the number of subjects and number of events in group i at time t , respectively. The first term in the above equations can be interpreted as the probability to select a subject from a group of interest, i.e.,

$$e_{1t} = Pr(\text{group 1}) \times (m_{1t} + m_{2t}), \quad (16.14)$$

$$e_{2t} = Pr(\text{group 2}) \times (m_{1t} + m_{2t}). \quad (16.15)$$

Using the auxiliary terms

$$E_1 = \sum_t e_{1t}, \quad (16.16)$$

$$E_2 = \sum_t e_{2t}, \quad (16.17)$$

$$O_1 = \sum_t m_{1t}, \quad (16.18)$$

$$O_2 = \sum_t m_{2t}, \quad (16.19)$$

we can define the following test statistic

$$s = \sum_i^{\text{over groups}} \frac{(O_i - E_i)^2}{E_i}. \quad (16.20)$$

The statistic s follows a chi-square distribution with one degree of freedom.

16.7 Hazard Function

Next, we define the hazard function and its relationship with the survival function. The hazard function is given by

$$h(t) = \lim_{\Delta t \rightarrow 0} \frac{P(t \leq T < t + \Delta t | T \geq t)}{\Delta t}. \quad (16.21)$$

The hazard function $h(t)$, also called hazard rate, can be interpreted as an “instant probability” because the only individuals considered are those such that $T \geq t$ and $T < t + \Delta t$ for $\Delta t \rightarrow 0$. Put simply, $h(t)$ represents the chance you will succumb to the event in the next instant (because $\Delta t \rightarrow 0$) given that you survive up to time t .

The hazard function, $h(t)$, has the following properties:

- $h(t) \geq 0$ for all t .
- $h(t)$ has no upper bound.
- $h(t)$ can assume any shape.

When $h(t) = 0$, it means that no event occurred within the time interval Δt .

The cumulative hazard function, denoted $H(t)$, describes the accumulated risk up to time t , and it is given by

$$H(t) = \int_0^t h(\tau) d\tau. \quad (16.22)$$

The function $H(t)$ can also be seen as the total amount of risk that has been accumulated up to time t . The integration/summation over $h(t)$ makes the interpretation simpler, but some details are lost during this process.

There is an important relationship between $h(t)$, $f(t)$, and $S(t)$, given by

$$h(t) = \frac{f(t)}{S(t)}. \quad (16.23)$$

Therefore, the hazard, density, and survival functions are not independent from each other. In the following, we derive this relationship.

From the definition of the hazard function, we have

$$h(t) = \lim_{\Delta t \rightarrow 0} \frac{P(t \leq T < t + \Delta t | T \geq t)}{\Delta t} \quad (16.24)$$

$$= \lim_{\Delta t \rightarrow 0} \frac{P(t \leq T < t + \Delta t)}{P(T \geq t) \Delta t} \quad (16.25)$$

$$= \lim_{\Delta t \rightarrow 0} \frac{P(t \leq T < t + \Delta t)}{\textcolor{red}{S(t)} \Delta t} \quad (16.26)$$

$$= \frac{f(t)}{S(t)} \quad (16.27)$$

The first step followed from the property of a conditional probability, and the second from the definition of a survival function (in red). The last step shows the desired relationship.

In addition to the above relationship, there is another important connection between $h(t)$ (or $H(t)$) and $S(t)$, given by

$$S(t) = \exp\left(-\int_0^t h(\tau) d\tau\right) = \exp\left(-H(t)\right). \quad (16.28)$$

From this, the inverse relationship from $S(t)$ to $h(t)$, given by

$$h(t) = -\left[\frac{dS(t)/dt}{S(t)}\right], \quad (16.29)$$

is derived as follows. Starting from

$$S(t) = \int_t^\infty f(\tau) d\tau, \quad (16.30)$$

we have

$$1 - S(t) = \int_0^t f(\tau) d\tau. \quad (16.31)$$

Differentiating both side of Eqn 16.31 with respect to t gives

$$\frac{d(1 - S(t))}{dt} = -\frac{dS(t)}{dt} = f(t). \quad (16.32)$$

Using Eq. 16.23 for $f(t)$, we obtain the desired relation in Eq. 16.29.

The difference between a hazard function and a survival function can be summarized as follows:

- The hazard function focuses on failing.
- The survival function focuses on surviving.

By making some assumptions about $h(t)$, one can obtain a parametric model. The relationship between $h(t)$ and $S(t)$, given by Eq. 16.28, makes the survival distribution a parametric model. Specific models that are frequently used include:

- Weibull model
- Exponential model
- Log-logistic model
- Log-normal model
- Gamma model

Compared with a nonparametric model, parametric assumptions enable to model a survival function elegantly and in more detail. However, the major risk associated with such an approach is to make assumptions that are not justified by the data. In the following, we discuss four parametric models in detail.

16.7.1 Weibull Model

For the Weibull model, the hazard, survival, and density functions are respectively given by

$$h(t) = \lambda p (\lambda t)^{p-1}, \quad (16.33)$$

$$S(t) = \exp(-(\lambda t)^p), \quad (16.34)$$

$$f(t) = \lambda p (\lambda t)^{p-1} \exp(-(\lambda t)^p). \quad (16.35)$$

Here, $\lambda > 0$ is a rate parameter, and $p > 0$ is a shape parameter allowing to control the behavior of the hazard function. Specifically, one can observe the following:

- $h(t)$ is monotonously decreasing when $p < 1$.
- $h(t)$ is constant when $p = 1$.
- $h(t)$ is monotonously increasing when $p > 1$.

The expected life time and its variance are given by

$$\mathbb{E}[T] = \frac{1}{\lambda} \Gamma\left(1 + \frac{1}{p}\right), \quad (16.36)$$

$$\text{Var}(T) = \frac{1}{\lambda^2} \Gamma\left(1 + \frac{2}{p}\right) - \frac{1}{\lambda^2} \Gamma\left(1 + \frac{1}{p}\right)^2. \quad (16.37)$$

Here, Γ is the Gamma function defined by

$$\Gamma(x) = \int_0^\infty t^{x-1} \exp(-t) dt. \quad (16.38)$$

16.7.2 Exponential Model

For the exponential model, the hazard, survival, and density functions are respectively given by

$$h(t) = \lambda, \quad (16.39)$$

$$S(t) = \exp(-\lambda t), \quad (16.40)$$

$$f(t) = \lambda \exp(-\lambda t). \quad (16.41)$$

The exponential model depends only on the rate parameter $\lambda > 0$.

The expected life time and its variance are given by

$$\mathbb{E}[T] = \frac{1}{\lambda}, \quad (16.42)$$

$$\text{Var}(T) = \frac{1}{\lambda^2}. \quad (16.43)$$

16.7.3 Log-Logistic Model

For the log-logistic model, the hazard, survival, and density functions are respectively given by

$$h(t) = \lambda \alpha (\lambda t)^{\alpha-1} (1 + (\lambda t)^\alpha)^{-1}, \quad (16.44)$$

$$S(t) = (1 + (\lambda t)^\alpha)^{-1}, \quad (16.45)$$

$$f(t) = \lambda \alpha (\lambda t)^{\alpha-1} (1 + (\lambda t)^\alpha)^{-2} \quad (16.46)$$

The log-logistic model depends on two parameters: the rate parameter $\lambda > 0$ and the shape parameter $\alpha > 0$. Depending on α , one can distinguish the following different behaviors of the hazard function:

- $h(t)$ is monotonously decreasing from ∞ when $\alpha < 1$.
- $h(t)$ is monotonously decreasing from λ when $\alpha = 1$.
- $h(t)$ is first increasing and then decreasing when $\alpha > 1$.

Specifically

- $h(t = 0) = 0$.
- The maximum of $h(t)$ is at $t = (\alpha - 1)^{\frac{1}{\alpha}}$.

The expected life time and its variance are given by

$$\mathbb{E}[T] = \exp\left(\mu + \frac{\sigma^2}{2}\right), \quad (16.47)$$

$$\text{Var}(T) = (\exp(\sigma^2) - 1)\exp(2\mu + \sigma^2). \quad (16.48)$$

16.7.4 Log-Normal Model

For the log-normal model, the hazard, survival, and density functions are respectively given by

$$h(t) = \frac{\alpha}{\sqrt{2\pi}t} \exp\left(-\frac{\alpha^2(\ln(\lambda t))^2}{2}\right) \left(1 - \Phi(\alpha \ln(\lambda t))\right)^{-1}, \quad (16.49)$$

$$S(t) = 1 - \Phi(\alpha \ln(\lambda t)), \quad (16.50)$$

$$f(t) = \frac{\alpha}{\sqrt{2\pi}t} \exp\left(-\frac{\alpha^2(\ln(\lambda t))^2}{2}\right). \quad (16.51)$$

Since a normal distribution has two parameters, then the log-normal model has also two parameters: the mean $\mu \in \mathbb{R}$ and the standard deviation $\sigma > 0$. These parameters are obtained from the transformations $\mu = -\ln(\lambda)$ and $\sigma = \alpha^{-1}$. The behavior of the hazard function is similar to the log-logistic model for $\alpha > 1$.

The expected life time and its variance are given by

$$\mathbb{E}[T] = \exp\left(\mu + \frac{\sigma^2}{2}\right), \quad (16.52)$$

$$\text{Var}(T) = (\exp(\sigma^2) - 1)\exp(2\mu + \sigma^2). \quad (16.53)$$

16.7.5 Interpretation of Hazard Functions

In Fig. 16.3, we provide some examples for the four parametric models discussed previously. From this figure, one can see that the hazard function can assume a variety of different behaviors. The specific behavior of $h(t)$ suggests the parametric model to use for a particular problem. In the Table 16.1, we summarize some examples of characteristic hazard curves and their associated events.

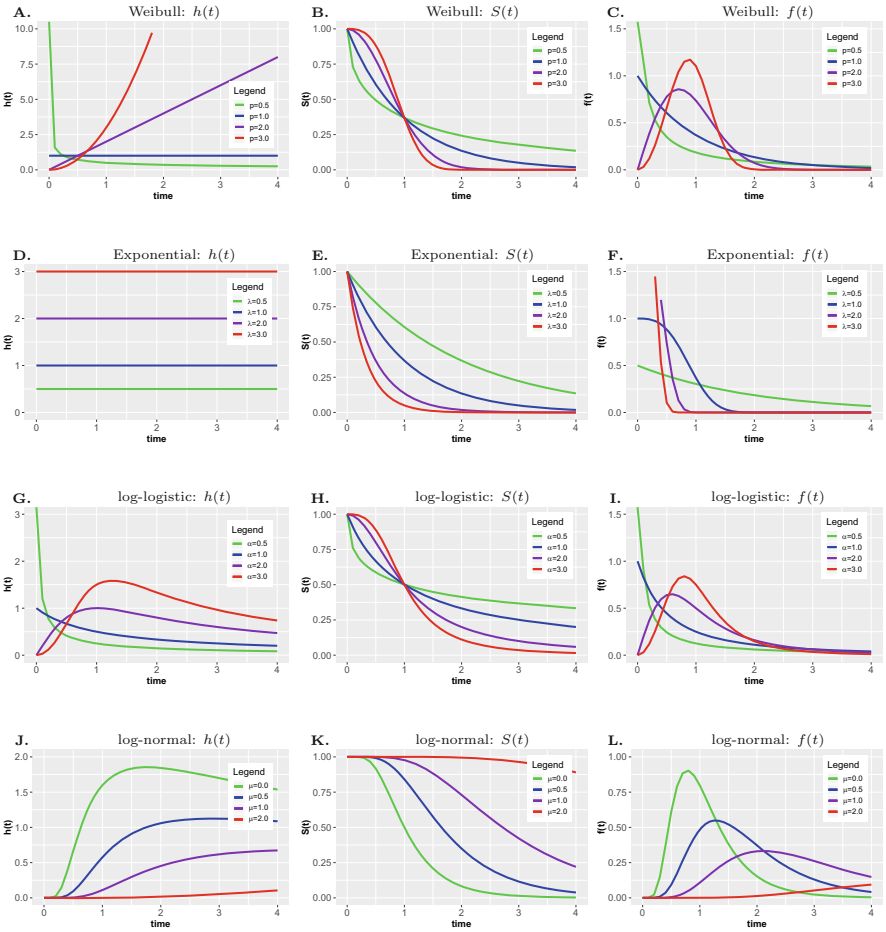


Fig. 16.3 Comparison of different parametric survival models

Table 16.1 Summary of characteristic hazard functions and their usage

Hazard function behavior	Event	Parametric model
Constant	Normal product	Weibull ($p = 1$)
Monotonous decreasing	Patient after surgery or stock market after crash or infant mortality	Log-logistic ($\alpha < 1$)
Monotonous (linear) increasing	Unsuccessful surgery or unsuccessful treatment or failure of a product	Weibull ($p = 2$)
Humped	Infection with tuberculosis (TB)	Log-normal
U-shaped	Heart transplant	

16.8 Cox Proportional Hazard Model

So far, we considered only models that did not include any covariates of the subjects. Now, we include such covariates, and the resulting model is called the **Cox proportional hazard model (CPHM)**. The CPHM is a semi-parametric regression model that defines the hazard function as follows:

$$h(t, X) = h_0(t)\exp(\beta_1 X). \quad (16.54)$$

Here, $h_0(t)$ is called the baseline hazard. The baseline hazard can assume any functional form. Examples of covariates are gender, smoking habit, or medication intake.

Equation 16.54 may look like a special case because no constant β_0 is included. However, the following calculation shows that it is actually included in $h_0(t)$ since

$$h(t, X) = h_0(t)' \exp(\beta_0 + \beta_1 X), \quad (16.55)$$

$$= h_0(t)' \exp(\beta_0) \exp(\beta_1 X). \quad (16.56)$$

We can generalize the above formulation to p covariates as follows:

$$h(t, X) = h_0(t) \exp\left(\sum_i^p \beta_i X_i\right). \quad (16.57)$$

For $X = 0$, we obtain

$$h(t, X) = h_0(t), \quad (16.58)$$

which is the hazard function defined in Eq. 16.21 without an influence of covariates.

The CPHM for n covariates does not make assumption about the baseline hazard $h_0(t)$. However, the model assumes:

- Time independence of the covariates X_i
- Linearity in the covariates X_i
- Additivity
- Proportional hazard

The Cox proportional hazard regression model is a semi-parametric model because it does not make assumption about $h_0(t)$. However, it assumes a parametric form for the effect of the predictors on the hazard.

In many situations, one is interested in the numerical estimates of the regression coefficients β_i rather than the shape of $h(t, X)$ because this provides a summary of the overall results.

To demonstrate this, let's take the logarithm of the hazard ratio

$$\log \left(\frac{h(t, X)}{h_0(t)} \right) = \sum_i \beta_i X_i, \quad (16.59)$$

which is linear in X_i and β_i . From this formulation, the connection to a linear regression model is apparent. In other terms, this can be summarized as follows:

$$\log \left(\text{HR}_0 \right) = \log \left(\frac{\text{group hazard}}{\text{baseline hazard}} \right) = \sum_i \beta_i X_i. \quad (16.60)$$

Here, the group hazard corresponds to all effects of the covariates X_i , whereas the baseline hazard excludes all such effects. Hence, the sum over all covariates is $\log \text{HR}_0$.

Let's consider just one covariate, i.e., $p = 1$, and suppose that this covariate is the gender, which can assume the values $X_1 = 1$ (female) and $X_1 = 0$ (male). Then, we obtain

$$\log \left(\frac{\text{hazard female}}{\text{baseline hazard}} \right) = \beta_1 \quad (16.61)$$

$$\log \left(\frac{\text{hazard male}}{\text{baseline hazard}} \right) = 0. \quad (16.62)$$

By taking the difference, we obtain

$$\log \left(\frac{\text{hazard female}}{\text{baseline hazard}} \right) - \log \left(\frac{\text{hazard male}}{\text{baseline hazard}} \right) = \log \left(\frac{\text{hazard female}}{\text{hazard male}} \right) \quad (16.63)$$

$$= \beta_1. \quad (16.64)$$

Hence, β_1 is the log hazard ratio of the hazard for females and males. This gives a direct interpretation of the regression coefficient β_1 . Transforming both sides of the above equation, we obtain the hazard ratio

$$\frac{\text{hazard female}}{\text{hazard male}} = \exp(\beta_1). \quad (16.65)$$

For the above evaluation, we used the binary covariate gender as an example. However, not all covariates are binary. In case of non-binary covariates, one can use a difference of one unit, i.e., $X_1 = x + 1$ and $X_1 = x$, to obtain a similar interpretation for the regression coefficients.

A major advantage of the CPHM framework is that we can estimate the parameters, β_i , without having to estimate the baseline hazard function $h_0(t)$. This implies that we also do not need to make parametric assumptions about $h_0(t)$, hence making the CPHM semi-parametric.

16.8.1 Why Is the Model Called a Proportional Hazard Model?

To appreciate why the model is called a proportional hazard model, let's consider two individuals m and n for the same model. Specifically, for individuals m and n , the hazards are given by:

$$h_m(t) = h_0(t) \exp \left(\sum_i \beta_i X_{mi} \right), \quad (16.66)$$

$$h_n(t) = h_0(t) \exp \left(\sum_i \beta_i X_{ni} \right). \quad (16.67)$$

Here, the covariates in blue are from individual m and the covariates in green are from individual n . Taking the ratio of both hazards, we obtain the following hazard ratio

$$\frac{h_m(t)}{h_n(t)} = \exp \left(\sum_i \beta_i (X_{mi} - X_{ni}) \right), \quad (16.68)$$

which is independent of the baseline hazard $h_0(t)$ because it cancels out. Here, it is important to note that the right-hand side is constant over time due to the time independence of the coefficients and covariates. Let's denote HR, the hazard ratio, i.e.,

$$\text{HR} = \exp \left(\sum_i \beta_i (X_{mi} - X_{ni}) \right). \quad (16.69)$$

A simple reformulating of Eq. 16.68 leads to

$$h_m(t) = \text{HR} \times h_n(t). \quad (16.70)$$

From this equation, it is clear that the hazard for the individual m is proportional to the hazard for the individual n , and the proportion, HR, is time independent.

16.8.2 Interpretation of General Hazard Ratios

The validity of the proportional hazard (PH) assumption allows a simple summarization for comparing time-dependent hazards. Specifically, instead of taking a hazard ratio between the hazards for two individuals, as in Eqs. 16.66 and 16.67, we can take a hazard ratio of arbitrary hazards for conditions we want to compare. Let's call these conditions "treatment" and "control," since these have an intuitive meaning in a medical context, and let's denote their corresponding hazards by

$$h(t, X^{\text{treatment}}) = h_0(t) \exp\left(\sum_i^p \beta_i X_i^{\text{treatment}}\right), \quad (16.71)$$

$$h(t, X^{\text{control}}) = h_0(t) \exp\left(\sum_i^p \beta_i X_i^{\text{control}}\right). \quad (16.72)$$

Regardless of the potential complexity of the individual hazards, assuming the PH holds, their hazard ratio is constant over time

$$\text{HR(T vs C)} = \frac{h(t, X^{\text{treatment}})}{h(t, X^{\text{control}})}. \quad (16.73)$$

Hence, the effect of treatment and control over time is given by one real valued number. Here, it is important to emphasize that the ratio of the two hazards is given by HR(T vs C) for any time point t , and hence, it is not the integrated ratio over time. Specifically, it gives the instantaneous relative risk, where the relative risk (RR) quantifies the cumulative risk integrated over time.

In contrast with the comparison of survival curves for treatment and control, e.g., using a log-rank test, which gives us only a binary distinction, the HR tells us something about the magnitude and the direction of this difference. Put simply, the HR has the following interpretation:

- $\text{HR(T vs C)} > 1$: The treatment group experiences a higher hazard over the control group $\Rightarrow$ control group is favored.
- $\text{HR(T vs C)} = 1$: No difference between the treatment and the control group.
- $\text{HR(T vs C)} < 1$: The control group experiences a higher hazard over the treatment group $\Rightarrow$ treatment group is favored.

For instance, for $HR(T \text{ vs } C) = 1.3$, the hazard of the treatment group is increased by 30% compared to the control group, and for $HR(T \text{ vs } C) = 0.7$, the hazard of the treatment group is decreased by 30% compared to the control group [24].

16.8.3 Adjusted Survival Curves

A Cox proportional hazard model can be used to modify estimates for a survival curve. Using Eq. 16.28, it follows that

$$S(t, X) = \exp\left(-\int_0^t h_0(\tau) \exp\left(\sum_i \beta_i X_i\right) d\tau\right) \quad (16.74)$$

$$= \exp\left(-\exp\left(\sum_i \beta_i X_i\right) \int_0^t h_0(\tau) d\tau\right) \quad (16.75)$$

$$= \exp\left(-\int_0^t h_0(\tau) d\tau\right)^{\exp(\sum_i \beta_i X_i)} \quad (16.76)$$

$$= S_0(t)^{\exp(\sum_i \beta_i X_i)}. \quad (16.77)$$

In general, one can show that

$$S(t, X) \leq S(t), \quad (16.78)$$

since the survival probability is always smaller than 1 and the exponent is always positive.

16.8.4 Testing the Proportional Hazard Assumption

In the above discussion, we assumed that the proportional hazard (PH) assumption holds. In the following, we discuss three ways (two graphical and one analytical) that can be used to evaluate the PH assumption.

16.8.4.1 Graphical Evaluation

The two graphical methods to assess the PH assumption perform a comparison for each variable/covariate one at a time [280]. The underlying idea of both methods is as follows:

- I. Comparison of estimated $\ln(-\ln)$ survival curves
- II. Comparison of observed and predicted survival curves

Graphical Method I To illustrate this method, we need to take the $\ln(-\ln)$ of Eq. 16.77. This leads to

$$\ln(-\ln S(t, \mathbf{X})) = \sum_{i=1}^p \beta_i X_i + \ln(-\ln S_0(t)). \quad (16.79)$$

Using the expression in Eq. 16.79 to evaluate two individuals, characterized by the following specific covariates

$$\mathbf{X}_1 = (X_{11}, X_{12}, \dots, X_{1p}), \quad (16.80)$$

$$\mathbf{X}_2 = (X_{21}, X_{22}, \dots, X_{2p}), \quad (16.81)$$

gives

$$\ln(-\ln S(t, \mathbf{X}_1)) - \ln(-\ln S(t, \mathbf{X}_2)) = \sum_{i=1}^p \beta_i (X_{1i} - X_{2i}). \quad (16.82)$$

From Eq. 16.82, we can see that the difference between $\ln(-\ln)$ of survival curves for two individuals, having different covariate values, is a constant given by the right-hand side.

To assess the PH assumption, we perform such a comparison for each covariate one at a time. In case of categorical covariates, all values will be assessed. For continuous covariates, we categorize them first, and then perform the comparison. The reason for using Eq. 16.82 for each covariate one at a time and not for all at once is that performing such a comparison covariate by covariate is more stringent.

From Eq. 16.82, follows that survival curves cannot cross each other if the hazards are proportional. Observation of such crossings lead to a clear violation of the PH assumption.

Graphical Method II The underlying idea of this approach, to compare the observed and expected survival curves to assess the PH assumption, is the graphical analog of the goodness-of-fit (GOF) testing.

Here, observed survival curves are obtained from stratified estimates of KM curves. The strata are derived from the categories of the covariates, and the expected survival curves are obtained from a CPHM with adjusted survival curves, as given by Eq. 16.77.

The comparison is performed similarly for the $\ln(-\ln)$ survival curves, i.e., for each covariate one at a time. Then, the observed and expected survival curves for each strata are plotted on the same graph for assessment. If, for each category of the covariates, the observed and expected survival curves are close to each other, then the PH assumption holds.

Kleinbaum [280] suggested to assume that the PH assumption holds unless there is a very strong evidence against it, namely:

- Survival curves cross and don't look parallel over time.
- Log cumulative hazard curves cross and don't look parallel over time.
- Weighted Schoenfeld residuals clearly increase or decrease overtime; see Sec. 16.8.4.2 (tested by a significant regression slope).

If the PH assumption doesn't hold for a particular covariate, then we are getting an average HR (averaged over the event times). In many cases, this is not necessarily a bad estimate.

16.8.4.2 Goodness-of-Fit Test

To test the validity of the PH assumption, several statistical tests have been suggested. However, the most popular one, introduced in [218], is a variation of a test originally proposed in [421] based on the so-called Schoenfeld residuals.

To carry out this test, the following steps are performed for each covariate one at a time:

1. Estimate a CPHM and obtain Schoenfeld residuals for each predictor/covariates.
2. Set up a reference vector containing the ranks of events. Specifically, the subject with the first (earliest) event receives a value of 1, the next subject receives a value of 2, and so on.
3. Perform a correlation test between the variables obtained in the first and second steps. The null hypothesis tested is that the correlation coefficient between the Schoenfeld residuals and the ranked event times is zero.

The Schoenfeld residual [421] for subject i and covariate k , experiencing the event at t_i , is given by

$$r_{ik} = X_{ik} - \bar{X}_k(\beta, t_i). \quad (16.83)$$

Here, X_{ik} is the individual value for subject i , and $\bar{X}_k(\beta, t_i)$ is the weighted average of the covariate values for the subjects at risk at time t_i , denoted $\mathcal{R}(t_i)$, and it is given by

$$\bar{X}_k(\beta, t_i) = \sum_{j \in \mathcal{R}(t_i)} X_{jk} w_j(\beta, t_i). \quad (16.84)$$

The weight function for all subjects at risk, given by $\mathcal{R}(t_i)$, is

$$\Pr(\text{subject } j \text{ fails at } t_i) = w_j(\beta, t_i) = \frac{\exp(\beta^T \mathbf{X}_j)}{\sum_{l \in \mathcal{R}(t_i)} \exp(\beta^T \mathbf{X}_l)}. \quad (16.85)$$

The Schoenfeld residual in Eq. 16.83 is evaluated for the parameter vector β from a fitted CPHM.

Overall, for each covariate k , this gives a vector

$$\mathbf{r}_k = (r_{1k}, r_{2k}, \dots, r_{nk}), \quad (16.86)$$

which is compared with the vector of rank values through a correlation test.

16.8.5 Parameter Estimation of the CPHM via Maximum Likelihood

So far, we formulated the CPHM and utilized it in a number of different settings. Now, we are dealing with the problem of estimating the regression coefficients β of the model.

Conceptually, the values of the regression coefficients are obtained via maximum likelihood (ML) estimates, i.e., by finding the parameters β of our CPHM that maximize $L(\beta|\text{data})$. Importantly, the CPHM does not specify the base hazard. This implies that without explicitly specifying it, the full likelihood of the model cannot be defined. For this reason, Cox proposed a *partial likelihood*.

The full likelihood for right-censored data, assuming no ties, would be composed of two contributions: one for individuals observed to fail at time t_i , contributing to the density $f(t_i)$, and the other for individuals censored at time t_i , contributing to the survival function $S(t_i)$. The product of both defines the full likelihood, denoted L_F , given by

$$L_F = \prod_i f(t_i)^{\delta_i} S(t_i)^{1-\delta_i}. \quad (16.87)$$

Here, δ_i indicates censoring. Using Eq. 16.23, we can rewrite L_F using the hazard function

$$L_F = \prod_i h(t_i)^{\delta_i} S(t_i). \quad (16.88)$$

16.8.5.1 Case Without Ties

Assuming that there are no ties in the data, i.e., event times t_i are unique, formally, the Cox partial likelihood function [86, 87] is given by

$$L(\beta) = \prod_{t_i \text{ uncensored}} \frac{h_0(t) \exp(\beta^T \mathbf{X}_i)}{\sum_{j \in \mathcal{R}(t_i)} h_0(t) \exp(\beta^T \mathbf{X}_j)}, \quad (16.89)$$

where $\mathcal{R}(t_i)$ is again the set containing the subjects at risk at t_i . Here, again the baseline hazard $h_0(t)$ is not needed because it cancels out.

The solution of Eq. 16.89 is given by the coefficients β that maximize the function $L(\beta)$, i.e.,

$$\beta_{ML} = \underset{\beta}{\operatorname{argmax}} L(\beta). \quad (16.90)$$

To obtain the coefficients β_k , we solve the following system of equations:

$$\frac{\partial L}{\partial \beta_k} = 0, \quad \forall k. \quad (16.91)$$

Usually, this needs to be carried out numerically using computational optimization methods. Practically, the log likelihood can be used to simplify the numerical analysis because it converts the product term of the partial likelihood function into a sum.

16.8.5.2 Case with Ties

As mentioned previously, the above given Cox partial likelihood is only valid for data without ties. However, in practice, ties of events can occur. For this reason, extensions are needed to deal with this problem. Three of the most widely used extensions are exact methods [86, 97, 270], the Breslow approximation [59], and the Efron approximation [123].

There are two types of exact methods. One assumes that the time is discrete while the other assumes continuous time. Due to the discrete nature of the time, the former model is called exact discrete method [86]. This method assumes that occurring ties are true ties and there exists no underlying ordering that would resolve the ties. Formally, it has been shown that this can be described by a conditional logit model that considers all possible combinations obtained for d_i tied subjects drawn from all subjects at risk at t_i . In contrast, Kalbfleisch and Prentice suggested an exact method assuming continuous times. In this model, ties arise as a result of imprecise measurement, e.g., due to scheduled doctor visits. Hence, this model assumes that there exists an underlying true ordering for all events, and the partial likelihood needs to consider all possible orderings for resolving ties. This involves considering all possible permutations (combinations) of tied events leading to an average likelihood [270, 461].

A major drawback of both exact methods is that they are very computationally expensive due to the high number of combinations to be considered when there are many ties. This means that the methods can even become computationally infeasible. For this reason, the following two methods, providing approximations of the exact partial likelihood and computationally much faster, are preferred.

The first method is the Breslow approximation [59] given by

$$L_B(\boldsymbol{\beta}) = \prod_{t_i \text{ uncensored}} \frac{\exp(\boldsymbol{\beta}^T \mathbf{S}_i)}{\left[\sum_{j \in \mathcal{R}(t_i)} \exp(\boldsymbol{\beta}^T \mathbf{X}_j) \right]^{d_i}}. \quad (16.92)$$

This approximation utilizes, $\mathcal{D}(t_i)$, the set of all subjects experiencing their event at the same time t_i , where d_i is the number of subjects given by $d_i = |\mathcal{D}(t_i)|$ and

$$\mathbf{S}_i = \sum_{j \in \mathcal{D}(t_i)} \mathbf{X}_j. \quad (16.93)$$

That means the set $\mathcal{D}(t_i)$ provides information about the tied subjects at time t_i . It is interesting to note that using the following simple identification

$$\exp(\boldsymbol{\beta}^T \mathbf{S}_i) = \prod_{k \in \mathcal{D}(t_i)} \exp(\boldsymbol{\beta}^T \mathbf{X}_k) \quad (16.94)$$

leads to an alternative formulation of the Breslow approximation

$$L_B(\boldsymbol{\beta}) = \prod_{t_i \text{ uncensored}} \frac{\prod_{k \in \mathcal{D}(t_i)} \exp(\boldsymbol{\beta}^T \mathbf{X}_k)}{\left[\sum_{j \in \mathcal{R}(t_i)} \exp(\boldsymbol{\beta}^T \mathbf{X}_j) \right]^{d_i}}. \quad (16.95)$$

Overall, the Breslow approximation looks similar to the Cox partial likelihood with minor adjustments. One issue with the Breslow method is that it considers each of the events, at a given time, as distinct from each other and it allows all failed subjects to contribute with the same weight to the risk set.

In contrast with the Breslow approximation, the Efron approximation allows each of the members that fail at time t_i to contribute partially (in a weighted way) to the risk set. The Efron approximation [123] is given by

$$L_E(\boldsymbol{\beta}) = \prod_{t_i \text{ uncensored}} \frac{\prod_{k \in \mathcal{D}(t_i)} \exp(\boldsymbol{\beta}^T \mathbf{X}_k)}{\prod_{j=1}^{d_i} \left[\sum_{k \in \mathcal{R}(t_i)} \exp(\boldsymbol{\beta}^T \mathbf{X}_k) - \frac{j-1}{d_i} \sum_{k \in \mathcal{D}(t_i)} \exp(\boldsymbol{\beta}^T \mathbf{X}_k) \right]}.$$

Overall, when there are no ties in the data, all approximations give the same results. Also, for a small number of ties, the differences are usually small. The Breslow approximation works well when there are few ties, but it is problematic for a large number. In general, the Efron approximation almost always works better; hence, it is the preferred method. For this reason, it is the default in the function *coxph()* available in R. Both the Breslow and Efron approximations give coefficients that are biased toward zero.

16.9 Stratified Cox Model

In Sect. 16.8.4, we discussed approaches for testing the PH assumption. In this section, we show that a stratification of the Cox model is a way to deal with covariates for which the PH assumption does not hold.

Let's assume that we have p covariates for which the PH assumption holds except for one covariate. Furthermore, we assume that the violating covariate assumes values in S different categories. If this variable is continuous, we need to define S discrete categories and discretize it.

For this, we can specify a hazard function, for each strata s , given by

$$h_s(t, X(s)) = h_{0,s}(t) \exp(\beta^T X(s)). \quad (16.96)$$

Here, $X(s) \in \mathbb{R}^p$ are the covariates for which the PH assumption holds, and $\beta \in \mathbb{R}^p$ and $s \in \{1, \dots, S\}$ are the different strata. We wrote the covariate as a function of strata s to indicate that only subjects having values within this strata are used. Put simply, the categories s are used to stratify the subjects into S groups for which a Cox model is fitted.

For each of these strata-specific hazard function, one can define a partial likelihood function, $L_s(\beta)$, in the same way as for the ordinary CPHM. The overall partial likelihood function, for all strata, is then given by the product of the individual likelihoods as follows:

$$L(\beta) = \prod_{s=1}^S L_s(\beta). \quad (16.97)$$

We want to emphasize that the parameters β are constant across the different strata, i.e., we are fitting S different models but the covariate-dependent part is identical for all of these models, and only the time-dependent baseline hazard function is different. This feature of the stratified Cox model is called the *no-interaction* property. This implies that the hazard ratios are the same for each stratum.

16.9.1 Testing No-Interaction Assumption

A question that arises is whether it is justified to assume a no-interaction model for a given data set. This question can be answered with a likelihood ratio (LR) test. To achieve this, we need to specify the interaction model given by

$$h_s(t, X(s)) = h_{0,s}(t) \exp(\beta_s^T X(s)). \quad (16.98)$$

Practically, this can be done by introducing dummy variables. For $S = 2$ strata, we need one dummy variable $Z^* \in \{0, 1\}$, leading to the following interaction model

$$h_s(t, \mathbf{X}(s)) = h_{0,s}(t) \exp(\boldsymbol{\beta}^T \mathbf{X}(s) + \beta_{11}(Z^* \times X_1) + \beta_{21}(Z^* \times X_2) + \cdots + \beta_{p1}(Z^* \times X_p)).$$

For $Z^* = 0$, we have

$$\text{Coefficient for } X_1 : \beta_1, \quad (16.99)$$

$$\text{Coefficient for } X_2 : \beta_2, \quad (16.100)$$

$$\vdots \quad (16.101)$$

$$\text{Coefficient for } X_p : \beta_p, \quad (16.102)$$

and for $Z^* = 1$, we have

$$\text{Coefficient for } X_1 : \beta_1 + \beta_{11}, \quad (16.103)$$

$$\text{Coefficient for } X_2 : \beta_2 + \beta_{21}, \quad (16.104)$$

$$\vdots \quad (16.105)$$

$$\text{Coefficient for } X_p : \beta_p + \beta_{p1}. \quad (16.106)$$

This shows that the coefficients differ for the two strata.

For $S > 2$ strata, we need to introduce $S - 1$ dummy variables Z_j^* , $j \in \{1, \dots, S - 1\}$ with $Z_j^* \in \{1, \dots, S\}$. This gives

$$\begin{aligned} h_s(t, \mathbf{X}(s)) = h_{0,s}(t) \exp(\boldsymbol{\beta}^T \mathbf{X}(s) + \beta_{11}(Z_1^* \times X_1) + \beta_{21}(Z_1^* \times X_2) + \cdots + \beta_{p1}(Z_1^* \times X_p) \\ + \beta_{11}(Z_2^* \times X_1) + \beta_{21}(Z_2^* \times X_2) + \cdots + \beta_{p1}(Z_2^* \times X_p) \\ \vdots \\ + \beta_{11}(Z_{S-1}^* \times X_1) + \beta_{21}(Z_{S-1}^* \times X_2) + \cdots + \beta_{p1}(Z_{S-1}^* \times X_p)). \end{aligned}$$

In this way, we obtained, from the no-interaction model (NIM) and the interaction model (IM), the likelihoods to be used for the test statistic $LR = -2 \log L_{\text{NIM}} + 2 \log L_{\text{IM}}$. The statistic, LR, follows a chi-square distribution with $p(S - 1)$ degrees of freedom.

16.9.2 Case of Many Covariates Violating the PH Assumption

In the case where there are more than one covariate violating the PH assumption, there is no elegant extension. Instead, the approach is usually situation-specific, requiring the combination of all these covariates into a single covariate X^* having S strata. An additional problem is imposed by the presence of continuous covariates, which requires discrete categorization. Both issues (large number of covariates violating the PH assumption and continuous covariates) lead to a complicated situation making such an analysis very laborious. This is especially true for the testing of the no-interaction assumption.

16.10 Survival Analysis Using R

In this section, we show how to perform a survival analysis using R. We provide some scripts that enable to obtain numerical results for different problems. To demonstrate such an analysis, we use data from lung cancer patients provided in the package *survival* [460].

16.10.1 Comparison of Survival Curves

In Listing 16.1, we show an example, based on the lung cancer data, to compare the survival curves of female and male using the packages *survival* and *survminer* [273, 460] available in R. The result of this analysis is shown in Fig. 16.4. From the total number of available patients (228), we select 175 randomly. For the selected patients, we estimate the Kaplan–Meier survival curves and compare them using a log-rank test. The p -value from this comparison is $p < 0.0001$, which means that, e.g., based on a significance level of $\alpha = 0.05$, we need to reject the null hypothesis “there is no difference in the survival curves for male and female.”

By setting options in the function *ggsurvplot*, we added to Fig. 16.4 information about the number of subjects at risk in interval steps of 100 days (middle figure) and the number of censoring events (bottom figure). This information is optional, but one should always complement survival curves with this table because it provides additional information about the data upon which the estimates are based.

Usually, it would be also informative to add confidence intervals to the survival curves. This can be accomplished by setting the option *conf.int* to TRUE (not used to avoid an overloading of the presented information).

Listing 16.1: An example for comparing the survival curves of female and male patients using lung cancer data.

```
library("survival")
library("survminer")

data(cancer, package="survival")
ind <- sample(1:228, 175)
lung <- lung[ind,]

fit <- survfit(Surv(time, status) ~ sex, data = lung)

d <- data.frame(time = fit$time,
                 n.risk = fit$n.risk,
                 n.event = fit$n.event,
                 n.censor = fit$n.censor,
                 surv = fit$surv,
                 upper = fit$upper,
                 lower = fit$lower
                )

ggsurvplot(
  fit,                                     # survfit object with calculated
    statistics.                           # statistics
  pval = TRUE,                            # show p-value of log-rank test.
  conf.int = FALSE,                      # show confidence intervals for
                                           # point estimates of survival curves.
  #conf.int.style = "step",               # customize style of confidence
    intervals
  xlab = "Time in days",                  # customize X axis label.
  break.time.by = 100,                   # break X axis in time intervals by
    200.
  ggtheme = theme_light(),               # customize plot and risk table with
    a theme.
  risk.table = "abs_pct",                 # absolute number and percentage at
    risk.
  risk.table.y.text.col = TRUE,           # colour risk table text
    annotations.
  risk.table.y.text = TRUE,               # show bars instead of names in text
    annotations
                                           # in legend of risk table.
  ncensor.plot = TRUE,                   # plot the number of censored
    subjects at time t
  #surv.median.line = "hv",               # add the median survival pointer.
  font.x = 18,
  font.y = 17,
  font.tickslab = 13,
  legend.labs =
    c("Male", "Female"),                 # change legend labels.
  palette =
    c("#239b56", "#2E9FDF")             # custom color palettes.
)
```

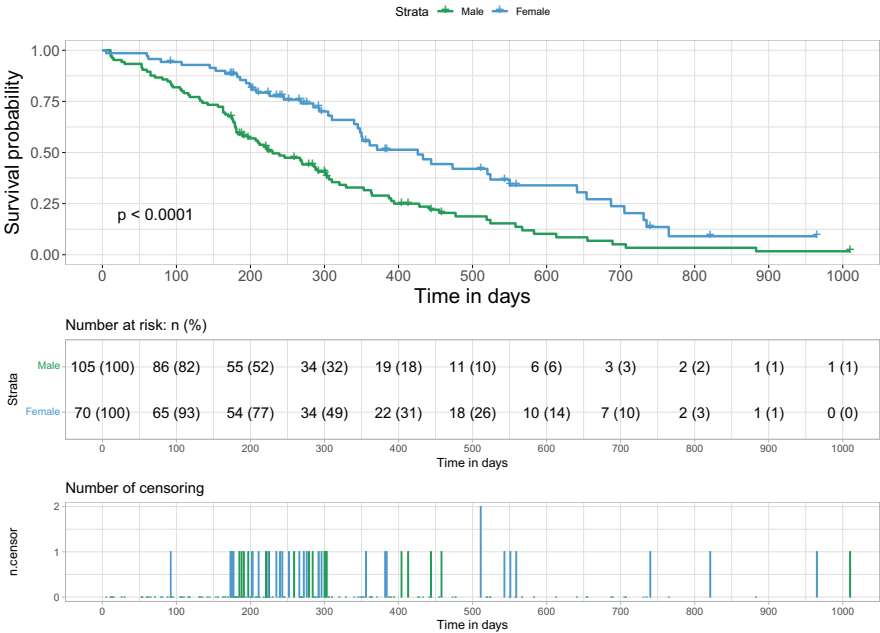


Fig. 16.4 The result of Listing 16.1. Top: The two survival curves for male (green) and female (blue) are shown for a duration of 1000 days. Middle: The number of subjects at risk is shown in interval steps of 100 days. Bottom: The number of censoring events is shown for the same interval steps

16.10.2 Analyzing a Cox Proportional Hazard Model

Here, we will illustrate how to perform the analysis of a CPHM. We will use again the lung cancer data, and as covariate, we will use the sex of the patients. Listing 16.2 provides the steps of the analysis as well as the corresponding outputs. In this model, the p -value of the regression coefficient is 0.000126, indicating a statistical significance of this coefficient. Hence, the covariate sex has a significant contribution to the hazard.

The hazard ratio of female/male is 0.4788. That means the hazard for group female is by a factor 0.4788 reduced compared to group male, or it is reduced by 52.12%.

Finally, the outputs of the statistical tests at the end of the listing provide information about the overall significance of the model. The three tests assessed the null hypothesis “all the regression coefficients are zero,” and they are asymptotically equivalent. All the three tests are significant, and this indicates that the null hypothesis needs to be rejected.

Listing 16.2: An example for a CPHM with one covariate for the lung cancer data.

```

> res.cox <- coxph(Surv(time, status) ~ sex, data = lung)
>
> summary(res.cox)
Call:
coxph(formula = Surv(time, status) ~ sex, data = lung)

    n = 175, number of events = 128

      coef exp(coef) se(coef)      z Pr(>|z|)
sex -0.7364    0.4788   0.1903 -3.869 0.000109 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

      exp(coef) exp(-coef) lower .95 upper .95
sex    0.4788      2.088    0.3297    0.6953

Concordance= 0.604 (se = 0.023 )
Likelihood ratio test= 15.94 on 1 df,  p=7e-05
Wald test               = 14.97 on 1 df,  p=1e-04
Score (logrank) test  = 15.61 on 1 df,  p=8e-05

```

16.10.3 Testing the PH Assumption

Using the fitted model above, we can now test the PH assumption for sex. Listing 16.3 provides the corresponding script and its outputs. Here, the null hypothesis tested is “whether the correlation between the Schoenfeld residuals and the ranked failure time is zero.” The test is not statistically significant for the covariate sex. Hence, we can consider that the proportional hazard assumption holds.

Listing 16.3: Testing the PH assumption for a CPHM with one covariate for the lung cancer data.

```

> test.ph.a <- cox.zph(res.cox)
> test.ph.a
      chisq df    p
sex      2.35  1 0.13
GLOBAL   2.35  1 0.13

```

In Listing 16.4, we show how to obtain the Schoenfeld residuals, as discussed in Sect. 16.8.4.2, and we also provide a visualization of the resulting residuals.

Listing 16.4: Schoenfeld residuals for a CPHM with one covariate for the lung cancer data.

```
ggcoxzph(test.ph.a)
```

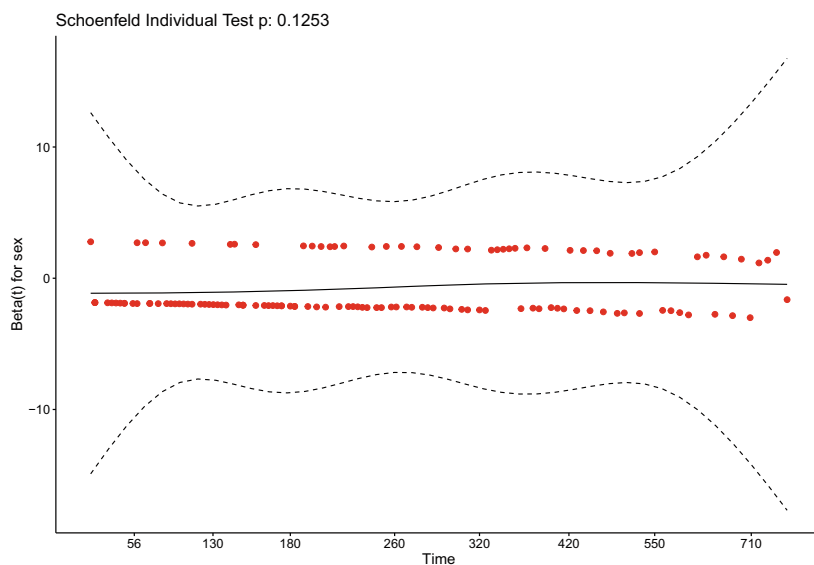


Fig. 16.5 Visualization of the scaled Schoenfeld residuals of sex against the transformed time

Figure 16.5 depicted the result of Listing 16.4. In this figure, the solid line is a smoothing spline fit of the scaled Schoenfeld residuals against the transformed time, and the dashed lines indicate ± 2 standard errors. A systematic deviation from a straight horizontal line would indicate a violation of the PH assumption, since for a valid assumption, the coefficient(s) do not vary over time. Overall, the solid line is sufficiently straight to assume that the PH holds. Figure 16.5 shows also the p -value of the result obtained using Listing 16.3.

16.10.4 Hazard Ratios

Finally, we present results for the full multivariate CPHM using all *seven* available covariates, in the lung cancer dataset, as input for the model. Listing 16.5 gives the corresponding code.

Listing 16.5: An example for a CPHM using 7 covariates for the lung cancer data.

```
res.cox <- coxph(Surv(time, status) ~ age + sex + ph.ecog +
  ph.karno +
```

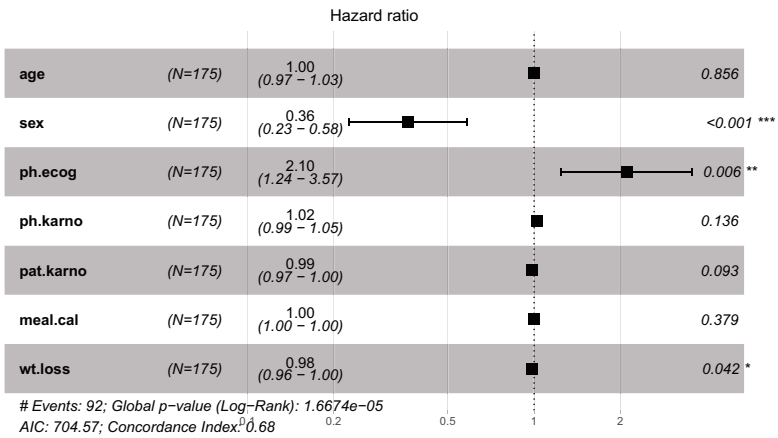



Fig. 16.6 Forest plot of hazard ratios for a multivariate CPHM

A convenient way to summarize the results is by using a forest plot, shown in Fig. 16.6. This figure shows the hazard ratios for the *seven* covariates, where the mean is represented by the square symbol and the confidence interval of the estimates is represented by the horizontal line. The right-hand side shows the *p*-values of the corresponding regression coefficients, which can also be obtained using the function `summary(res.cox)`. Overall, the covariate *sex* reduces the hazard, whereas *ph.ecog* (measure of well-being according to ECOG performance score) increases it. All other covariates are located around 1, i.e., their effect is marginal.

16.11 Further Reading

As we have seen in this chapter, survival analysis is a very broad topic. In order to gain a deeper understanding, we refer to [81, 129, 193, 280, 319].

16.12 Summary

Survival analysis, also known as event history analysis, is a very interesting topic because the type of data, which can be analyzed with such methods, are different from the other types of data discussed in this book. Although *time-to-event data* may appear dull at first, the information, which can be extracted from them, is quite rich and informative. A reason behind this is that the outcome variable of a survival analysis, namely, the “event,” assumes usually a role of great interest, e.g., death, bankruptcy, failure, or crime. This enables a direct connection of a

statistical analysis with events of apparent real-world interest. For this reason, it is not surprising that survival analysis is used in diverse fields ranging from medicine and biology to economy and marketing. Hence, every data scientist should be familiar with survival analysis.

Learning Outcome 16: Survival Analysis

Survival analysis is a collection of methods for analyzing time-to-event data that can be flexibly applied to a wide range of problems.

As we have seen in this chapter, for multivariate data, a survival analysis can become quickly complex and the interpretation of the results can be nontrivial, e.g., with respect to hazard ratios. Moreover, the testing of the proportional hazard assumption or the analysis of the stratified Cox model can become complex and labor intensive. The latter means that a diagnosis of the models, which demands a thorough understanding by the analyst, is required. Overall, while the very basics of survival analysis are rather intuitive, more advanced approaches are generally quite involved.

16.13 Exercises

1. Discuss right censoring according to the information provided in Fig. 16.1.
2. Calculate the Kaplan–Meier estimator manually for the data provided in Fig. 16.2.
3. Load the `pbcc` data from the library *survival*. Install and load the library *survival* and then use the following command to load the data:
`data(pbc, package="survival")`
 - a. Compare the survival curves of female and male patients from the “pbc” data.
 - b. Perform an analysis of the CPHM using the covariate “sex” for the “pbc” data.
 - c. Test the PH assumption for the CPHM using the covariate “sex” for the “pbc” data.
 - d. Visualize the results of the scaled Schoenfeld residuals of sex against the transformed time.
 - e. Perform an analysis of the CPHM using all the covariates for the “pbc” data.

Chapter 17

Foundations of Learning from Data



17.1 Introduction

So far, we pursued a practical approach to data science. This included the presentation of many worked examples using R, allowing to reproduce the analysis of essentially all topics discussed. Now, we are going a step back to present a theoretical perspective on learning from data.

In this chapter, we discuss two fundamental aspects of learning from data. The first addresses the computational learning theory, and the second is about different learning paradigms. In general, computational learning theory studies questions of “learnability.” That means given a task, natural questions are: How many samples are needed to learn? Under what conditions is successful learning possible? Is there a best learning algorithm under all conditions? Such questions can be formalized with the *probably approximately correct (PAC)* learning framework, which defines *learnability* formally. We will see that PAC can provide learnability bounds for a finite hypothesis space, and by using the *Vapnik-Chervonenkis (VC) dimension*, such results can be extended to an infinite hypothesis space.

These results will give us theoretical characterizations of the difficulty of machine learning problems and the capabilities of certain models. The main results of this part are summarized by the *fundamental theorem of statistical learning*, which provides conditions for the PAC learnability of an infinite hypothesis space.

The second fundamental aspect, addressed in this chapter, is about the definition of different learning paradigms in machine learning. So far, we focused on supervised learning and unsupervised learning methods. Examples for supervised learning are classification or regression methods (see Chaps. 9 and 11), whereas clustering or dimension reduction methods are unsupervised (see Chaps. 7 and 8). However, there are data sets (or combinations thereof) that have characteristics, which cannot be described by those two learning paradigms but, instead, require advanced learning paradigms. In this chapter, we discuss seven modern learning paradigms: semi-supervised learning, one-class classification, positive-unlabeled

learning, few/one-shot learning, transfer learning, multi-task learning, and multi-label learning. We will see that these learning paradigms open a new universe of novel methods and approaches.

17.2 Computational and Statistical Learning Theory

The field of *computational learning theory (COLT)* studies whether a given problem is efficiently learnable or not. That means it is not only important to know under what circumstances a problem is learnable but that there is an efficient computational realization of a learning algorithm. Hence, computational considerations are also of importance. Most importantly, computational learning theory will provide us with a formal definition of *learnability* due to Valiant. While there have been earlier attempts to define *learning* formally, e.g., [197], Valiant’s seminal work in [474] is generally considered the starting point of computational learning theory.

A field that is closely related to COLT is *statistical learning theory (SLT)*. Compared to COLT, SLT is less concerned with computational aspects of learning. Furthermore, it allows infinite hypothesis spaces, which can be dealt with the VC dimension due to Vapnik and Chervonenkis. Interestingly, compared to statistics, SLT does not assume that there exists a correct model which should be estimated. Instead, SLT presumes that the correct model is entirely unknown and the goal is to find a model with similar prediction capabilities. Despite such differences, in the literature, the terms computational learning theory and statistical learning theory are often used synonymously.

17.2.1 Probabilistic Learnability

Before we formalize the problem, let’s consider two particular classification problems. These examples will teach us valuable lessons regarding the feasibility of *learning*.

Classification of Tumor Tissues Let’s assume that we have a number of images of tumor tissues, e.g., from lung cancer. The tumors are labeled by “b” if the tumor sample (TS) is benign or “m” if it is malignant. Let’s denote the data set that contains all of these tumor tissues by D , i.e., $D = \{(TS_1, f_1), \dots, (TS_n, f_n)\}$ with $f_i \in \{‘b’, ‘m’\}$. Suppose that we are presented with an additional tumor tissue, TS_x . What is the label, f_x , of this tumor?

Usually, such a question is answered by a pathologist. A pathologist is a clinician with an additional education in histology, which enables to distinguish tissues based on their morphological differences. In this sense, a pathologist serves as a human classifier for tumor tissue. Now, suppose we present the above outlined problem to 1000 pathologists all over the world asking them to classify the unlabeled

tumor tissue. Since becoming a pathologist requires years of training, this problem is generally considered as difficult. Empirical studies show that, in general, the agreement among pathologists is usually not 100% but much less; see, e.g., [446]. How much less depends on the particular problem/disease.

Classifying Boolean Patterns The next example is of mathematical nature. Suppose that we have a training data set D consisting of three Boolean patterns of the form $(x = (x_1, x_2); y)$, as shown on the left-hand side in Fig. 17.1, i.e., $D = \{((0, 0); 0), ((0, 1); 1), ((1, 0); 1)\}$. That means, for each of these three Boolean patterns, we know its corresponding class label y . Now, the question is what Boolean function f , with

$$f : \mathcal{X} \rightarrow \mathcal{Y};$$

(17.1)

$$x = (x_1, x_2) \in \mathcal{X}, y \in \mathcal{Y};$$

(17.2)

$$\mathcal{X} = \{0, 1\}^2, \mathcal{Y} = \{0, 1\}.$$

(17.3)

provides the best classification of two-dimensional Boolean patterns?

Let’s further assume that we have a finite set of 16 functions, $f_1, \dots, f_{16}$, available as a potential solution to this problem. Each of these 16 functions is fully defined by its truth table, as shown on the right-hand side in Fig. 17.1. Using the training data D , we see that only the functions f_7 and f_{15} allow an error-free learning of the training data because both functions lead to the correction classification of the three input data points.

It is interesting to note that, based on the above assumptions, we have no information, which would allow us to distinguish f_7 from f_{15} . Hence, both functions are reasonable solutions to our classification problem. Nevertheless, only one of them will be correct because the unknown input pattern $x = (1, 1)$ will result in either $y = 0$ or $y = 1$ (shown in yellow in Fig. 17.1), but not in both! As a

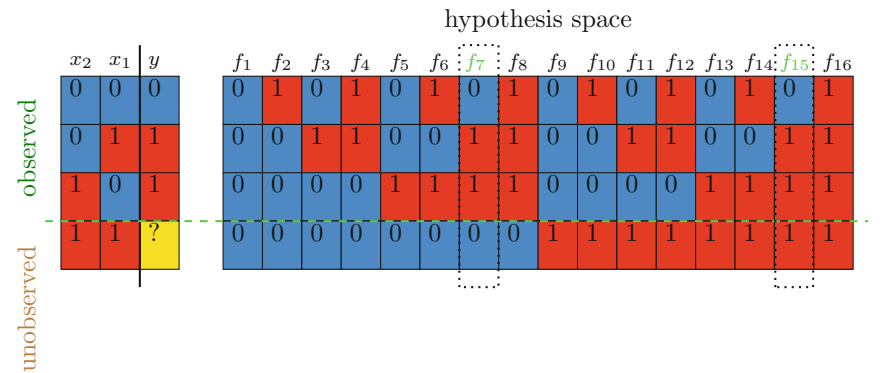


Fig. 17.1 Classification of two-dimensional Boolean patterns

consequence, we cannot guarantee that we can select the correct function from the provided information.

Summary From the first (real world) example, we can learn that there are classification problems humans cannot solve. From the second (mathematical) example, we can learn that there are classification problems mathematics cannot solve. The first observation may not surprise you; however, the latter one does because it shows that despite the simplicity of the problem (two-dimensional Boolean patterns, no noise, deterministic mapping from the input pattern to the class label), there is no approach that could guard us against selecting the wrong function.

The dilemma from the above consideration comes from our *implicit* assumption that “learning a problem” means that for every data set $\mathcal{D}$ and hypothesis space, we can always find an error-free solution. This is not possible. However, if we amend our expectations in a probabilistic manner, we can find quantitative approaches. Hence, the nature of computational and statistical learning theory is to provide probabilistic statements in the form of bounds about the *learnability* of a problem. Below, we will quantify the term learnability using appropriate error measures.

17.2.2 Probably Approximately Correct (PAC) Learning

In this section, we provide a specific quantitative definition of *learnability*, called *probably approximately correct (PAC)* or PAC learnability, due to Valiant [474]. There are also other definitions of learnability, which will be briefly discussed in the following section.

The PAC model is based on the following definitions [339]: Let X be the space of all possible instances or data points that can be observed, also called the input space. A function

$$c : X \rightarrow \mathcal{Y} \quad (17.4)$$

is called a *concept* mapping from X to the output space $\mathcal{Y}$. For simplicity, the output space shall be binary labels, i.e., $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{-1, +1\}$. Put simply, a concept function provides a classification for the data points $x \in X$. The (possibly infinite) set C containing all allowed concept functions $c \in C$ is called *concept space*.

We assume that all instances $x \in X$ are independently and identically distributed (i.i.d.) according to a fixed but unknown distribution P , i.e., $x \sim P$. Furthermore, a data point x and its class label are fully defined by a distribution P over X and a target concept $t \in C$ by $(x, t(x))$.

The learning task is defined as follows: For a set of i.i.d. samples given by $S(n) = \{x_1, \dots, x_n | x_i \sim P \ \forall i\}$ and labels $\{t(x_1), \dots, t(x_n)\}$ for a target concept t , what is the concept $h \in H$ that minimizes the generalization error? Here H is the set of hypotheses, called hypothesis space, accessible to the learning algorithm. Possible cases for a hypothesis space are $H = C$ or $C \subset H$.

Definition 17.1 (Generalization Error) The *generalization error* or *risk* of a concept $h \in H$ for a target concept $t \in C$ and distribution P is given by

$$R(h) = \text{error}_P(h, t) = \Pr_{x \sim P}(h(x) \neq t(x)) = \mathbb{E}_{x \sim P}[I_{h(x) \neq t(x)}]. \quad (17.5)$$

This error measure is the probability of misclassifications of concept h for all possible data x sampled from P given the true classifier t .

Alternatively, one can formulate the generalization error for the zero-one loss

$$L(h(x), t(x)) = I_{h(x) \neq t(x)} \quad (17.6)$$

by

$$R(h) = \text{error}_P(h, t) = \mathbb{E}_{x \sim P}[L(h(x), t(x))]. \quad (17.7)$$

Due to the fact that in reality we have only a finite data sample to assess the quality of h , the generalization error needs to be approximated by the *empirical risk*, also called *empirical error*, given by

$$\widehat{R}(h) = \widehat{\text{error}}_{S(n)}(h, t) = \frac{1}{n} \sum_{i=1}^n I_{h(x_i) \neq t(x_i)}. \quad (17.8)$$

Here, $S(n) = \{x_1, \dots, x_n | x_i \sim P \forall i\}$ is a data set of sample size n .

Taking the expectation value of Eq. 17.8 with respect to $S(n) \sim P$, i.e., for all data sets $S(n)$ of size n sampled from P , one can show that this results in the generalization error [339],

$$\text{error}_P(h, t) = \mathbb{E}_{S(n) \sim P}[\widehat{\text{error}}_{S(n)}(h, t)]. \quad (17.9)$$

Now, we can give the definition of learnability according to a PAC model [336, 339].

Definition 17.2 (Probably Approximately Correct (PAC) Learning) Let C be a concept class. We call C *probably approximately correct (PAC)* learnable if there exists a prediction model M that outputs h_S given $S(n)$, H , and such that for every target concept $t \in C$, for every distribution P over X , and for all $0 < \delta, \varepsilon < 1/2$, M can learn a concept $h_S \in H$ such that

$$\Pr_{S(n) \sim P}(\text{error}_P(h_S, t) \leq \varepsilon) \geq 1 - \delta \quad (17.10)$$

for all sample sizes n with $n \geq n_0$.

Here, n_0 is a polynomial that grows in $1/\varepsilon, 1/\delta$.

In the above definition, the prediction model M is also called *learning algorithm* because, given a data set $S(n)$ and hypothesis space H as input, it can *learn* the concept $h_S \in H$, i.e.,

$$M(S(n), H) = h_S, \quad (17.11)$$

such that Eq. 17.10 holds. By “learn,” we mean a procedure that selects h_S from all available concepts in H . It is important to note that because the result of M depends on $S(n)$, which is a random sample from P , h is a random variable.

We would like to note that there is no assumption regarding the underlying distribution P from which the data are sampled. For this reason, PAC learning is a *distribution-free* model.

For clarity, we would like to highlight that PAC learning has its name from the two-step estimation of the error given by

$$\underbrace{\Pr_{S(n) \sim P} \left(\underbrace{\text{error}_P(h_S, t) \leq \varepsilon}_{\text{approximately correct}} \right)}_{\text{probably}} \geq 1 - \delta. \quad (17.12)$$

In case there are additional properties of the run time of the prediction model M , the above definition can be extended.

Definition 17.3 (Efficient PAC Learnability) Suppose that all the conditions in Definition 17.2 hold. If, in addition, the time complexity of M is $\text{poly}(1/\delta, 1/\varepsilon)$, for a fixed polynomial “poly,” we call C efficiently PAC learnable.

In case the hypothesis space, H , is finite and a learning algorithm M can find a hypothesis h with $\widehat{\text{error}}_{S(n)}(h, t) = 0$, then h is called consistent hypothesis and M consistent learner. Based on this, one can define the set of all consistent hypothesis, called version space $VS_{H, S(n)}$.

Definition 17.4 (Version Space) For a target concept t , a hypothesis space H , and a sample $S(n)$, the following set is called a version space,

$$VS_{H, S(n)} = \{h \in H \mid h(x) = t(x) \forall x \in S(n)\}. \quad (17.13)$$

For this special case of a consistent learner M , one can derive the following bound.

Theorem 17.1 (Fine Hypothesis Space with h Consistent) Let H be a finite hypothesis space and M a learning algorithm that finds, for any target concept $t \in H$ and $S(n)$, a consistent hypothesis $h_S \in VS_{H, S(n)}$. Then, for any $\varepsilon, \delta > 0$, Eq. 17.10 holds if $n \geq n_0$ with

$$n_0 = \frac{1}{\varepsilon} \left(\log |H| + \log \frac{1}{\delta} \right). \quad (17.14)$$

Here, $|H|$ is the cardinality of the hypothesis space. This characterizes the *sample complexity*.

Proof Let's derive a bound for the probability that the true error, given a consistent hypothesis h_S , is larger than ε .

$$\Pr_{S(n) \sim P} \left(\widehat{\text{error}}_{S(n)}(h_S, t) = 0 \wedge \text{error}_P(h_S, t) > \varepsilon \right) \leq \quad (17.15)$$

$$\Pr_{S(n) \sim P} \left(\exists h \in H : \widehat{\text{error}}_{S(n)}(h, t) = 0 \wedge \text{error}_P(h, t) > \varepsilon \right) \leq \quad (17.16)$$

$$\sum_{i=1}^{|H|} \Pr_{S(n) \sim P} \left(\widehat{\text{error}}_{S(n)}(h_i, t) = 0 \wedge \text{error}_P(h_i, t) > \varepsilon \right) \leq \quad (17.17)$$

$$\sum_{i=1}^{|H|} \Pr_{S(n) \sim P} \left(\widehat{\text{error}}_{S(n)}(h_i, t) = 0 \right) = |H| \Pr_{S(n) \sim P} \left(\widehat{\text{error}}_{S(n)}(h, t) = 0 \right) \quad (17.18)$$

Now, we derive a bound for the last probability.

$$\Pr_{S(n) \sim P} \left(\widehat{\text{error}}_{S(n)}(h, t) = 0 \right) = \Pr_{S(n) \sim P} \left(h(x_i) = t(x_i) | \forall x_i \in S(n) \right) \quad (17.19)$$

$$= \prod_{i=1}^n \Pr_{S(n) \sim P} \left(h(x_i) = t(x_i) \right) \quad (17.20)$$

$$= \Pr_{S(n) \sim P} \left(h(x) = t(x) \right)^n \quad (17.21)$$

$$= \left(1 - \Pr_{S(n) \sim P} \left(h(x) \neq t(x) \right) \right)^n \quad (17.22)$$

$$= \left(1 - \text{error}_P(h, t) \right)^n \quad (17.23)$$

$$\leq (1 - \varepsilon)^n \quad (17.24)$$

$$\leq \exp(-\varepsilon n) \quad (17.25)$$

The second last step follows from

$$\text{error}_P(h, t) > \varepsilon \Rightarrow \quad (17.26)$$

$$1 - \text{error}_P(h, t) \leq 1 - \varepsilon \quad (17.27)$$

In summary, this leads to the following bound according to Haussler

$$\Pr_{S(n) \sim P} \left(\widehat{\text{error}}_{S(n)}(h_S, t) = 0 \wedge \text{error}_P(h_S, t) > \varepsilon \right) \quad (17.28)$$

$$\leq |H|\exp(-\varepsilon n) \quad (17.29)$$

□

From the above proof, one can see that the probability in Eq. 17.28 is for a consistent hypothesis h_S . By extending the definition of the version space, we can obtain a more intuitive understanding of this probability.

Definition 17.5 (ε -Exhausted Version Space) The version space $VS_{H,S(n)}$ is called ε -exhausted with respect to a target concept t and P , if every hypothesis h in $VS_{H,S(n)}$ has a generalization error less than ε , i.e., $\forall h \in VS_{H,S(n)}$ we have

$$\text{error}_P(h, t) < \varepsilon. \quad (17.30)$$

From this definition, one can see that the probability in Eq. 17.28 is the probability that the version space is not ε -exhausted. This means that *at least* one consistent hypothesis has an error larger than ε . Furthermore, one can see that by varying ε and n one can control the size of the ε -exhausted version space. For instance, by increasing n for a fixed ε , one can shrink this space by removing hypotheses that violate the assumptions because they have a larger error than allowed.

Let's study an example for a PAC-learnable concept class to see how to apply the above definitions.

17.2.2.1 Example: Rectangle Learning

Suppose that the space X is two-dimensional (x, y) , with $x, y \in \mathbb{R}_+$. The concept class C consists of all axis-aligned rectangles defined by

$$f(x) = \begin{cases} +1 \equiv \bullet & \text{if } x \text{ is inside the rectangle;} \\ -1 \equiv \bullet & \text{if } x \text{ is outside the rectangle.} \end{cases} \quad (17.31)$$

for $x = (x, y)$ with $x, y \in \mathbb{R}_+$.

The learning algorithm L is defined as follows: For a sample of size n given by $S(n) = \{(x_1, y_1), \dots, (x_n, y_n)\}$, determine

$$x_l = \min\{x_1, \dots, x_n\}, \quad (17.32)$$

$$x_u = \max\{x_1, \dots, x_n\}, \quad (17.33)$$

$$y_l = \min\{y_1, \dots, y_n\}, \quad (17.34)$$

$$y_u = \max\{y_1, \dots, y_n\}. \quad (17.35)$$

These coordinates define the lower, (x_l, y_l) , and upper, (x_u, y_u) , boundaries of a rectangle.

Hence, the complementary event has the probability

$$\Pr_{S(1)|P}(\text{one data point falls outside } R') = 1 - \text{error}_P(R'(x'_2), h) = 1 - \frac{\varepsilon}{4}. \quad (17.38)$$

Finally, the probability that R' does not include any data point from a sample of size n is

$$\Pr_{S(n) \sim P}(R' \text{ does not include any data point from } S(n)) = \left(1 - \frac{\varepsilon}{4}\right)^n \quad (17.39)$$

thanks to Eq. 17.38 and the fact that the n data points are i.i.d samples. This probability is important, and it will be utilized below.

Now, we need to distinguish two cases: (1) $\text{error}_P(f, h) \leq \varepsilon$ and (2) $\text{error}_P(f, h) > \varepsilon$. If (1) holds, there is nothing to show, and we are done. So let's assume that (2) holds. However, in this case, at least one R' must be included in the corresponding R so that

$$\begin{aligned} \Pr_{S(1)|P}(\text{one data point falls inside } R) &> \\ \Pr_{S(1)|P}(\text{one data point falls inside } R') &= \frac{\varepsilon}{4} \end{aligned} \quad (17.40)$$

in order to enable $\text{error}_P(f, h) > \varepsilon$. From this, follows the bound for the negative events, given by

$$\begin{aligned} \Pr_{S(n) \sim P}(R \text{ does not include any data point from } S(n)) &\leq \\ \Pr_{S(n) \sim P}(R' \text{ does not include any data point from } S(n)) & \end{aligned}$$

Putting everything together for the four rectangles R'_i , we obtain

$$\begin{aligned} \Pr_{S(n) \sim P}(\text{error}_P(f, h) > \varepsilon) &\leq \\ \Pr_{S(n) \sim P}(f \text{ does not include } \cup_{i=1}^4 \{R_i\}) &= \\ \Pr_{S(n) \sim P}(\cup_{i=1}^4 \{R_i \text{ does not include any data point from } S(n)\}) &\leq \\ \Pr_{S(n) \sim P}(\cup_{i=1}^4 \{R'_i \text{ does not include any data point from } S(n)\}) &\leq \\ &\Pr(R'_i \cap R'_j \neq \emptyset) \geq 0 \\ \sum_{i=1}^4 \Pr_{S(n) \sim P}(R'_i \text{ does not include any data point from } S(n)) &= \\ &4 \left(1 - \frac{\varepsilon}{4}\right)^n \leq \\ 4 \exp\left(-\frac{n\varepsilon}{4}\right) &\leq \delta \end{aligned}$$

The penultimate inequality is justified because $1 - x \leq \exp(-x)$ for all $x \in \mathbb{R}$.

From the last inequality, it follows that

$$n \geq \frac{4}{\varepsilon} \ln\left(\frac{4}{\delta}\right). \quad (17.41)$$

This allows us to state the final result as follows. If Eq. 17.41 holds, then

$$\Pr(\text{error}_P(f, h) \leq \varepsilon) \geq 1 - \delta. \quad (17.42)$$

Overall, this proves that rectangle learning is an example of a PAC-learnable problem.

17.2.2.2 General Bound for a Finite Hypothesis Space H : The Inconsistent Case

Above we gave a bound for a finite hypothesis space H and the consistent case. The consistent case implies $\widehat{\text{error}}_{S(n)}(h, t) = 0$ and $t \in H$. Now, we generalize this for a finite hypothesis space H for the inconsistent case. This means that $t \notin H$, and hence, the learning algorithm M can only find a hypothesis $h \in H$ that minimizes the training error. In this case, we call M an *agnostic learner*.

Theorem 17.2 (Finite Hypothesis Space H with h Inconsistent) *Let H be a finite hypothesis space, i.e., $|H| < \infty$, and M a model that returns h_S for a given target concept $t \in H$ and $S(n)$. Then for any $\delta > 0$, any P over X , the following inequality holds at least with probability $1 - \delta$:*

$$\text{error}_P(h_S, t) \leq \widehat{\text{error}}_{S(n)}(h_S, t) + \varepsilon \quad (17.43)$$

with

$$\varepsilon = \sqrt{\frac{\ln|H| + \ln\frac{2}{\delta}}{2n}} \quad (17.44)$$

Alternatively, Theorem 17.2 can be formulated as follows:

$$\Pr_{S(n) \sim P} \left(\left| \text{error}_P(h_S, t) - \widehat{\text{error}}_{S(n)}(h_S, t) \right| \leq \varepsilon \right) \geq 1 - \delta. \quad (17.45)$$

The sample complexity for the above bound is

$$n_0 = \frac{1}{2\varepsilon^2} \left(\log|H| + \log\frac{2}{\delta} \right), \quad (17.46)$$

which is now quadratic in ε in contrast with Eq. 17.14 (consistent case).

Proof Since the hypothesis space H is finite, let $h_1, \dots, h_{|H|}$ denote all possible hypotheses/concepts. Then, we can write

$$\begin{aligned}
 & \Pr_{S(n) \sim P} \left(\left| \text{error}_P(h_S, t) - \widehat{\text{error}}_{S(n)}(h_S, t) \right| > \varepsilon \right) \leq \\
 & \Pr_{S(n) \sim P} \left(\exists h \in H \text{ with } \left| \text{error}_P(h, t) - \widehat{\text{error}}_{S(n)}(h, t) \right| > \varepsilon \right) = \\
 & \Pr_{S(n) \sim P} \left(\left| \text{error}_P(h_1, t) - \widehat{\text{error}}_{S(n)}(h_1, t) \right| > \varepsilon \cup \dots \right. \\
 & \quad \left. \cup \left| \text{error}_P(h_{|C|}, t) - \widehat{\text{error}}_{S(n)}(h_{|C|}, t) \right| > \varepsilon \right) \leq \\
 & \sum_{i=1}^{|H|} \Pr_{S(n) \sim P} \left(\left| \text{error}_P(h_i, t) - \widehat{\text{error}}_{S(n)}(h_i, t) \right| > \varepsilon \right) \leq \\
 & 2|H| \exp(-2n\varepsilon^2) = \delta.
 \end{aligned}$$

□

For the last step, we used Hoeffding's inequality given by

$$\Pr_{S(n) \sim P} \left(\left| \text{error}_P(h_i, t) - \widehat{\text{error}}_{S(n)}(h_i, t) \right| > \varepsilon \right) \leq 2 \exp(-2n\varepsilon^2). \quad (17.47)$$

We would like to note that neither Theorem 17.2 nor its proof makes any assumption about the empirical risk $\widehat{\text{error}}_{S(n)}(h, t)$, hence allowing the values $\widehat{\text{error}}_{S(n)}(h, t) \geq 0$. As mentioned above, the case $\widehat{\text{error}}_{S(n)}(h, t) = 0$ is called *consistent*, and the case $\widehat{\text{error}}_{S(n)}(h, t) > 0$ is called *inconsistent*. Put simply, the inconsistent case means that the hypothesis space H does not include a concept $h = t$ that would lead to a zero empirical risk. Nevertheless, as shown above, Theorem 17.2 provides a bound for the inconsistent case.

Theorem 17.2 assumes, however, a finite hypothesis space H . This is indeed a severe limitation because many classification algorithms are able to realize an infinite number of concepts, and hence, they have an infinite hypothesis space. The general question is, whether learning, in the sense of Definition 17.2, from a finite data sample $S(n)$ with $n < \infty$, is also possible for $|H| = \infty$? The answer is yes, and the axis-aligned rectangle problem, in Sect. 17.2.2.1, provides again an example for this. The general answer to this question will be given in the next section.

17.2.3 Vapnik Chervonenkis (VC) Theory

A problem with the bounds derived so far is that these depend on the size of the hypothesis space $|H|$; see Eqs. 17.14 and 17.46. Hence, they cannot be used for an infinite hypothesis space. For this reason, in this section, we will extend the above

results to the case of an infinite hypothesis space. We will do this by using two purely combinatorial entities, the *growth function* and the *VC dimension*. This will lead to generalization bounds based on the VC dimension instead of $|H|$.

We would like to remark that, alternatively, this could be also achieved using the Rademacher complexity, which quantifies the richness of a family of functions to fit random data; however, this can be NP-hard.

Before we can provide a formal definition of the VC dimension, we need to introduce some auxiliary definitions that will clarify the following discussion. First, we will formalize the meaning of the effective number of hypothesis.

Definition 17.6 (Dichotomies Generated by H) Let $S(n) = \{x_1, \dots, x_n | x_i \sim P \forall i\}$. The dichotomies generated by a hypothesis space, H , on $S(n)$ are defined by

$$H(S(n)) = \{(h(x_1), \dots, h(x_n)) | h \in H\}. \quad (17.48)$$

That means, the set $H(S(n))$ contains all the different N -tuples that are realizable by the hypothesis space H .

Definition 17.7 (Growth Function) Let H be a hypothesis space. The mapping $\Pi_H(n) : \mathbb{N} \rightarrow \mathbb{N}$, defined for any $n \in \mathbb{N}$, by

$$\Pi_H(n) = \max_{S(n) \sim P} |H(S(n))| \quad (17.49)$$

is called the *growth function*.

The growth function gives, for each n , the maximum number of dichotomies that can be generated by H . Since for any hypothesis space H , $H(S(n)) \subseteq \{-1, +1\}^n$, the total number of different N -tuples is given by

$$\Pi_H(n) \leq 2^n. \quad (17.50)$$

Definition 17.8 The data sample $S(n) = \{x_1, \dots, x_n\}$ is called shattered by H if $\Pi_H(n) = 2^n$.

In this case, the hypothesis space, H , realizes all possible dichotomies of $S(n) \sim P$.

Definition 17.9 (VC Dimension) The *Vapnik-Chervonenkis (VC) dimension* of hypothesis space H is the size of the largest set that can be shattered by H , i.e.,

$$\text{VC-dimension}(H) = \max_{n \in \mathbb{N}} \left\{ n : \Pi_H(n) = 2^n \right\}. \quad (17.51)$$

In the following, we denote the VC dimension by d_{VC} . If sets of arbitrary size can be shattered, then $d_{VC} = \infty$. It is important to remark that a VC dimension of d_{VC}

does not mean that *any* set can be shattered by H but that there exists *at least one* set such that this is possible.

17.2.3.1 Example: One-dimensional Intervals

Let $\mathcal{X} = \mathbb{R}$ be the one-dimensional real axis, and a concept h is a closed interval $[a, b]$ defined by

$$h(x) = \begin{cases} 0 & \text{if } x \in [a, b]; \\ 1 & \text{if } x \notin [a, b]. \end{cases} \tag{17.52}$$

This defines the hypothesis space H of the problem. The points for $m = 2$, shown in Fig. 17.3, can be shattered by the shown concepts. For this reason, $d_{VC} \geq 2$. However, there are no three data points with label order $(0, 1, 0)$ that could be shattered by h because that would require two intervals. However, our hypothesis space, as define above, does not allow this. In Fig. 17.3, we show the general problem for $m = 3$ and this label order. Note that the absolute position of the three data points is not important for this argument. The only argument that matters is the order of the labels. Hence, for this problem, $d_{VC} = 2$.

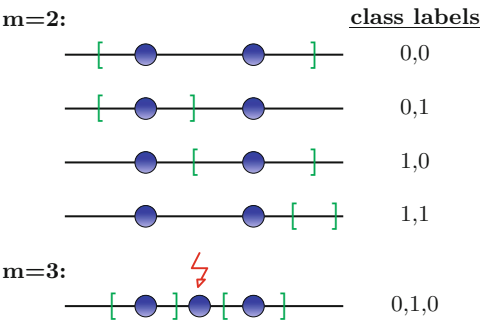
17.2.3.2 Example: Axis-Aligned Rectangles

The following two results are important because they connect the growth function with the VC dimension.

Theorem 17.3 (Sauer’s Lemma) *Let H be a hypothesis space with VC dimension d_{VC} . Then, for all $n \in \mathbb{N}$, the growth function is bound by the following inequality*

$$\Pi_H(n) \leq \sum_{i=0}^d \binom{n}{i}. \tag{17.53}$$

Fig. 17.3 Examples for $m = 2$ and $m = 3$ for $\mathcal{X} = \mathbb{R}$ with intervals as concept functions h . For the case $m = 3$, no two intervals, capable to realize the labeling 0, 1, 0, exist



Corollary 17.1 *Let H be a hypothesis space with VC dimension d_{VC} . Then, for all $n \geq d_{VC}$, we have*

$$\Pi_H(n) \leq \left(\frac{en}{d_{VC}}\right)^{d_{VC}}. \quad (17.54)$$

Here, e corresponds to Euler's number.

17.3 Importance of Bias for Learning

Another important aspect we want to discuss concerns the need of bias for learning. This may sound like a contradiction, at first, because bias will restrict our hypothesis space and, hence, remove flexibility of our model or learning algorithm. In order to understand this better, we discuss first the no free lunch theorem.

The *no free lunch (NFL)* theorem [427], by Wolpert, states that in the lack of prior knowledge (or inductive bias), any learning algorithm may fail on some learnable task. This implies the impossibility to obtain meaningful bounds on the error of a learning algorithm without prior assumptions and modeling.

Theorem 17.4 (No Free Lunch) *Let M be a learning algorithm for a binary classification task, with respect to a zero-one loss over a domain X . Let the sample size, n , be any number smaller than $|X|/2$, and let $S(n)$ be the training data. Then, there exists a distribution P such that:*

1. *There exists a concept $h = t \in H$ with $\text{error}_P(h, t) = 0$.*
2. *With probability of at least $1/7$ over $S(n)$, we have $\text{error}_P(M(S(n), H), t) \geq 1/8$.*

This theorem states that although the hypothesis space, H , contains the target concept t , given a finite training sample $S(n)$, the learning algorithm M cannot find t for a particular distribution P .

Wolpert and others provided additional theorems proving, e.g., the following results [503]:

- For any equally weighted overall measure, each algorithm will perform equally well.
- Averaged over all target concepts t , two algorithms will perform equally.
- Averaged over all distributions P , two algorithms will perform equally.

From all these results, it follows that there is no learning algorithm better than any other for minimizing the expected error averaged over all possible tasks. Hence, no universally best learning algorithm exists. This provides one explanation for the plurality of methods one can find in data science.

We would like to highlight that the proof of the NFL theorems works because we are averaging over all possible tasks. That means, a possible improvement may result from excluding unfavorable distributions that prohibit PAC learning.

However, this requires us to introduce a bias in learning. One form of bias frequently used is *Occam's razor*, also called the bias of simplicity [502]. In general, this means that when there are multiple possible explanations/models given the same data, we should choose the simplest one.

Another formulation of this problem can be given using the generalization capabilities of a model. For instance, Schaffer argued that generalization (to unseen data) is only possible if we have additional information about the problem besides the training data [418]. This information could be domain knowledge about the problem or knowledge about characteristics of different learning algorithms. However, regardless of the nature of the bias, it is generally believed that generalization without bias is not possible [335].

In order to introduce a bias (or prior knowledge) for learning, one has, in general, the following possibilities [483]:

- Change the hypothesis space H .
- Change the probability distribution P .
- Change the input space $\mathcal{X}$.
- Change the loss function.

It is important to emphasize that none of the above measures will be able to eliminate the results of the NFL theorems, but by introducing a bias in learning, the average practical generalization accuracy can be increased for a particular case, rather than for all cases [502].

In general, one can formalise a machine learning problem either as a parameter optimization problem or a hypothesis search problem. In 1995, Vapnik proposed another minimization criterion: the risk minimization principle. Algorithms like support vector machines, instead of minimizing a function of the error, minimize the margin, which is defined as the distance between the nearest examples (the support vectors) and the decision surface.

17.4 Learning as Optimization Problem

So far we neglected the problem how a learning algorithm M selects a hypothesis from the hypothesis space H . In this section, we present the empirical risk minimization (ERM) and the structural risk minimization (SRM) as induction principles for inferring an optimal hypothesis. We will see that both induction principles formulate learning as an optimization problem.

17.4.1 Empirical Risk Minimization

Above, we discussed that the generalization error (risk) is not accessible to the learning algorithm. For this reason, it cannot be directly used. Instead, we use the

empirical risk $\widehat{R}(h)$ as an approximation for the risk. Specifically, by using $\widehat{R}(h)$, we try to find the hypothesis, which minimizes the empirical risk, i.e.,

$$h_S = \operatorname{argmin}_{h \in H} \widehat{R}(h). \quad (17.55)$$

This induction principle is called the *empirical risk minimization (ERM)*.

When introducing the empirical risk, we mentioned that one can show that its expectation value corresponds to the generalization error; see Eq. 17.9. Furthermore, using Hoeffding's inequality (see Exercise 2), one can show that the empirical risk provides indeed a good approximation for the generalization error. Hence, the empirical risk minimization corresponds to the minimization of the risk in the limit.

17.4.2 Structural Risk Minimization

In Chap. 13, we discussed regularization as a principle to penalize the optimization functional of (regression) models in order to perform an implicit form of model selection. It is possible to extend this to the ERM, which is intended to deal with a large sample size, by introducing a regularized ERM. This is called structural risk minimization (SRM).

SRM, introduced by Vapnik and Chervonenkis, is an inductive principle for model selection used for learning from finite training data. It provides a trade-off between the quality of fitting the training data (empirical risk) and the complexity of the hypothesis space (model complexity) via the VC dimension.

Let's consider a sequence of nested hypothesis spaces $\mathcal{H} = \cup_{i=1}^m H_i$,

$$H_1 \subset H_2 \subset H_3 \cdots \subset H_m \quad (17.56)$$

with a growing VC dimension

$$d_{VC}(1) < d_{VC}(2) < \cdots < d_{VC}(n) \quad (17.57)$$

and a training data set $S(n)$. For instance, $\mathcal{H}$ may be the class of all polynomial classifiers, where each H_i corresponds to the class of polynomial classifiers of degree i . This way, H_{i+1} represents more complex models than H_i , and the nested hypotheses allow to express prior knowledge, by specifying preferences over hypotheses within $\mathcal{H}$.

In general, the structural risk minimizes the empirical risk for each hypothesis space H_i using a regularization term $J(h, n)$ that considers the complexity (VC dimension) of H_i as follows:

$$h_S = \operatorname{argmin}_{h \in H} \left\{ \widehat{R}(h) + \lambda J(h, n) \right\}. \quad (17.58)$$

This is called the *structural risk minimization (SRM)*.

For binary classification, the SRM is given by

$$h_S = \operatorname{argmin}_{h \in \mathcal{H}} \left\{ \widehat{R}(h) + \min_{h \in H_i} \sqrt{\frac{d_{VC}(i) \log(2n/d_{VC}(i)) + \log(4/\delta)}{n}} \right\}. \quad (17.59)$$

On a general note, we would like to remark that our discussion on the ERM and SRM showed that a learning problem can be converted into an optimization problem. The objective (or cost) function of the optimization problem is the empirical risk (and the regularization term), and the domain of the learning algorithm M is the hypothesis space H .

17.5 Fundamental Theorem of Statistical Learning

Finally, we are in a position to state the central result of statistical learning theory. The fundamental theorem of statistical learning shows that the VC dimension completely characterizes PAC learnability of hypothesis classes of binary classifiers. The fundamental theorem states that a hypothesis class is PAC learnable if and only if its VC dimension is finite. The theorem also shows that if a problem is PAC learnable, then uniform convergence holds, and therefore the problem is learnable using the ERM rule [45, 427].

Theorem 17.5 (Fundamental Theorem of Statistical Learning) *Let H be the hypothesis space of functions from X to $\{0, 1\}$ and let the loss function be the zero-one loss. Then there exist constants C_1 and C_2 such that the following statements are equivalent:*

- H is PAC learnable with sample complexity

$$C_1 \frac{d_{VC} + \log(1/\delta)}{\varepsilon^2} \leq m_H \leq C_2 \frac{d_{VC} \log(1/\varepsilon) + \log(1/\delta)}{\varepsilon^2} \quad (17.60)$$

- The VC dimension of H , denoted by d_{VC} , is finite.
- ERM is a successful PAC learner for H .
- H has the uniform convergence property.

Lemma 17.1 (Sauer's Lemma) *If $VC - d(H) = d$, then even though H might be infinite, when restricting it to a finite set CX , its "effective" size is only $O(|C|^d)$.*

The VC dimension determines the general outline of the growth function, which in turn determines whether a class satisfies uniform convergence. This is equivalent to agnostic PAC learning, which implies PAC learning. This implies a finite VC dimension thanks to the no free lunch theorems.

17.6 Discussion

Computational learning theory addresses the problem to find optimal generalization bounds for supervised learning. We presented two formalisms of this framework: probably approximately correct (PAC) learning and the VC theory. Both approaches are nonparametric and distribution-free.

We have seen that, based on certain assumptions, one can derive error bounds and sample complexities to characterize learning in various situations. This could be done for a finite hypothesis space (PAC learning) and an infinite hypothesis space (VC dimension). A somewhat sobering result has been contributed by the NFL theorem because it shows that there is no universally best learning algorithm. Furthermore, we have seen that the ERM and SRM allow to convert the learning problem into an optimization problem for finding an optimal hypothesis. Overall, this led to the formulation of the fundamental theorem of statistical learning, which summarizes and interconnects all those results.

The insights of the NFL theorem are intimately related to the bias of learning to improve generalization abilities of a learning algorithm for unseen data. Put simply, bias can be seen as a specialization in a model to obtain a generalization in learning. This motivates also data-driven models to achieve a better generalization.

On a practical note, we would like to mention that boosting is a PAC-inspired method and SVM is based on minimizing the VC dimension. Hence, the fingerprints of both frameworks can be found in practical methods, although neither is intended to provide practical tools for analyzing data but, instead, provides formal verification approaches.

17.7 Modern Machine Learning Paradigms

The second fundamental aspect we will discuss in this chapter is about learning paradigms. So far, we discussed in this book many methods and algorithms from data science. Interestingly, from the perspective of the underlying learning paradigm, all of them fall into the realm of supervised learning (including the first part of this chapter) and unsupervised learning. However, modern data sets (or combinations thereof) can have characteristics that cannot be described by those two learning paradigms. For this reason, advanced machine learning paradigms have been introduced. In the following, we discuss seven modern learning paradigms.

1. Semi-supervised learning
2. One-class classification
3. Positive-unlabeled learning
4. Few/one-shot learning
5. Transfer learning
6. Multi-task learning
7. Multi-label learning

For clarity reasons, we want to mention that according to Kuhn [67, 293] a *paradigm* is generally characterized as follows:

A scientific paradigm is a set of concepts, patterns, or assumptions to which those in a particular professional community are committed and which forms the basis of further research.

In order to further highlight the importance of a paradigm in science, the term *worldview* has been suggested as a synonym to describe “a way of thinking about and making sense of the complexities of the real world” [276, 377]. Since machine learning is a scientific field, the above definition can be applied directly to define the *machine learning paradigms* (or briefly called learning paradigms) as used in this chapter.

We will see that each of the modern learning paradigms has different requirements for the underlying data. Hence, they do not merely provide alternative algorithmic or computational approaches for existing data characteristics but, instead, establish new conceptual approaches.

17.7.1 *Semi-supervised Learning*

The idea of semi-supervised learning is to use both labeled and unlabeled data for performing a supervised learning task [71].

Definition 17.10 A domain $\mathcal{D}$ consists of a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \{X_1, \dots, X_n\} \in \mathcal{X}$, i.e., $\mathcal{D} = \{\mathcal{X}, P(X)\}$.

Definition 17.11 A task $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a prediction function $f(X)$ with $f : \mathcal{X} \rightarrow \mathcal{Y}$, i.e., $\mathcal{T} = \{\mathcal{Y}, f(X)\}$.

The definition of a domain is similar to supervised learning; however, the resulting data are different. Specifically, for semi-supervised learning, there are two parts of the data: a labeled part, $D_L = \{(x_i, y_i)\}_{i=1}^{n_L}$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, and an unlabeled part $D_U = \{(x_j)\}_{j=1}^{n_U}$. This means that the available data are of the form $D = D_L \cup D_U$ (see Fig. 17.4).

Formally, semi-supervised learning can be defined as follows.

Definition 17.12 Given a domain, $\mathcal{D}$, with task, $\mathcal{T}$, labeled data $D_L = \{(x_i, y_i)\}_{i=1}^{n_L}$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ and unlabeled data $D_U = \{(x_j)\}_{j=1}^{n_U}$, *semi-supervised learning* is the process of improving the prediction function, f , by utilizing the labeled and unlabeled data.

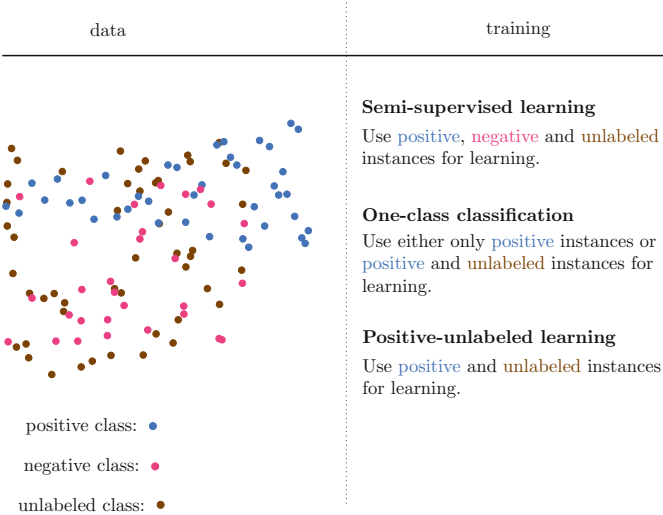


Fig. 17.4 Characterization of semi-supervised learning, one-class classification, and positive-unlabeled learning. The class label of instances are distinguished by the color, i.e., a positive class is blue, a negative class is red, and an unlabeled class is brown

17.7.1.1 Methodological Approaches

For semi-supervised learning, a broad variety of methods have been proposed. However, they can be distinguished based on the following two key concepts [526]:

- 1. Inductive methods
- 2. Transductive methods

Both concepts are fundamentally different from each other, and the training and prediction parts of such methods are largely different [176]. Put simply, these concepts can be formulated as follows:

Induction is reasoning from observed training cases to general rules, which are then applied to the test cases.

In contrast, transduction has the following meaning:

Transduction is reasoning from observed, specific (training) cases to specific (test) cases.

It is important to note that this implies that transductive learning does not distinguish between a training and testing step of a model. Instead, it uses both the training and testing data for training the model, in contrast with inductive learning. Consequently, transductive learning does not build a predictive model. For this reason, to test a new instance, the model needs to be trained again for all the available data. This is not necessary for inductive learning, because it leads to a predictive model that can be used for new instances without retraining the model.

It is interesting to note that the transductive learning is either explicitly or implicitly graph-based because information has to be propagated between different data points, which can be seen as nodes in a graph [313, 477]. A recent comprehensive review of semi-supervised learning, including details about algorithmic realizations, can be found in [477].

17.7.2 *One-Class Classification*

The idea of one-class classification (OCC) is to distinguish instances from one particular class from those outside this class [346, 401, 457]. This is quite different from ordinary classification, and for this reason, OCC was also called outlier detection, novelty detection, anomaly detection, or concept learning [261, 408]. Hence, OCC focuses on one particular class only.

17.7.2.1 **Methodological Approaches**

According to [277], one-class learning approaches can be categorized with respect to the way they are using the training data. This allows to distinguish approaches utilizing only positive data from approaches that learn from positive and unlabeled data. The latter has found widespread interest, and it is called positive-unlabeled learning. Due to the importance of such methods, we discuss this subcategory of one-class learning in the next section.

From a methodological point of view, there are two key concepts for one-class classification that use only positive-labeled data [26].

1. Density estimation
2. Boundary estimation

Density estimation methods estimate the density of the data points with a positive label. A new instance is classified according to a threshold [456]. Boundary estimation methods focus on setting boundaries around a small set of points, called target points. Some examples of methods, from this category, utilize support vector machines or neural networks [321, 417].

It is interesting to note that one-class classification using only positive-labeled data for density estimation is conceptually similar to statistical hypothesis testing [141]. However, methodologically, these approaches are different because OCC is not based on the concept of a sampling distribution, which specifies not only the estimation precisely but also the statistical interpretations thereof. In contrast, OCC approaches to density estimation are more broad, and for this reason, they vary considerably in their interpretation.

17.7.3 Positive-Unlabeled Learning

For positive-unlabeled learning, we are facing a classification problem when only labeled instances of one class are available. In addition, we have unlabeled data, which can come from any class, but their labels are unknown. For this reason, we have labeled data from one class (termed as “positive”) complemented with unlabeled data. The goal is to utilize these data for a classification task.

To obtain the data, we assume that n_p positive samples are randomly drawn from the marginal distribution $P(x|Y = +1)$ and n_u unlabeled samples are randomly drawn from $P(x)$ [366], resulting in the two data sets $D_p = \{(x_i, y_i)\}_{i=1}^{n_p}$ with $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$, and $D_u = \{(x_i)\}_{i=1}^{n_u}$ with $x_i \in \mathcal{X}$. Hence, in total we have the data set $D = D_p \cup D_u$ with $n = n_p + n_u$ samples. Furthermore, we assume that also for $x_i \in D_u$, labels in $\mathcal{Y}$ exist, but they are not observed.

Due to the lack of observable instances for the entire label space $\mathcal{Y}$, the problem is limited to a binary label space (simplifying the complexity).

Definition 17.13 The task $\mathcal{T}$ of positive-unlabeled learning consists of a label space $\mathcal{Y}$ and a prediction function $f(X)$ with $f : \mathcal{X} \rightarrow \mathcal{Y}$, i.e., $\mathcal{T} = \{\mathcal{Y}, f(X)\}$, where the label space $\mathcal{Y}$ is binary, i.e., $|\mathcal{Y}| = 2$.

Based on this definition and the above assumptions, positive-unlabeled learning can be formally defined as follows.

Definition 17.14 Given $D = D_p \cup D_u$, *positive-unlabeled learning* is the process of improving the prediction function f of the binary task $\mathcal{T}$ utilizing D_p and D_u .

Such approaches exploit inductive and transductive learning approaches, both of which adopt an iterative procedure to obtain reliable negative training data from the unlabeled data [375]. An example of such an inductive PU (positive-unlabeled) learning algorithm using bagging SVM to infer a GRN (gene regulatory network) is presented in [342].

17.7.3.1 Methodological Approaches

The main methodological approaches for positive-unlabeled learning can be distinguished as follows:

1. Two-step methods
2. Weighting methods

The two-step methods use the unlabeled data in step one to identify negative instances and then use a traditional classifier in step two. The weighting methods estimate real valued weights for the unlabeled data and then learn a classifier based on these weights. The weights represent the likelihood, or conditional probability, that an unlabeled instance belongs to a certain class. Hence, the problem is converted into a (constrained) regression problem.

Recently, a generative adversarial network (GAN) has been introduced for PU learning called GenPU [251]. GenPu consists of a number of generators and discriminators similar to a minimax game. These components generate simultaneously positive and negative samples with realistic properties, which can then be used with a standard classifier. For comprehensive reviews of positive-unlabeled learning, the reader is referred to [28, 262, 521].

17.7.4 Few/One-Shot Learning

The idea of few/one-shot learning is to utilize a (large) training set for learning a similarity function, which is then used in combination with a very small data set containing only one or a few instances about unknown classes to make predictions about these unknown classes [159, 269, 491].

Hence, few/one-shot learning utilizes semantic information from the training data to deal with few/one instances for new classes that are unknown from the training data. In Fig. 17.5, we summarize the idea of few/one-shot learning.

Few/one-shot learning utilizes three key components: (1) a labeled data set D , (2) a support set D_{su} , and (3) a query q representing a new instance for which a class label should be predicted. The labeled data D is given by $D = \{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ and $i \in \{1, \dots, n\}$, where n is the sample size, $\mathcal{X}$ feature space, and $\mathcal{Y}$ label space. If the cardinality of the label space is larger than two, i.e., $|\mathcal{Y}| > 2$, then we have a multiclass classification problem, otherwise a binary classification. The data set D serves as the training data to learn a similarity function g . This similarity function will then be used for evaluating the similarity of a query q to instances given in the support set D_{su} . The support set D_{su} is defined as follows.

Definition 17.15 A support set D_{su} is a labeled data set $D_{su} = \left\{ \{(x_i, y_i)\}_{i=1}^{n_s} \right\}_{s=1}^S$ providing information about labeled instances of S classes with $y_i \in \mathcal{Y}'$. For $n_1 = \dots = n_S = 1$, one obtains one-shot learning, and for $n_i > 1$ for all $i \in \{1, \dots, S\}$ with $|n_i|$ small, few-shot learning is obtained. For $n_1 = \dots = n_S = n$, this is called n -shot, S -way learning.

It is important to note that the label space of the support set D_{su} and the training data D are different, i.e., $\mathcal{Y} \neq \mathcal{Y}'$. Hence, the semantic transfer from the training data is accomplished via the similarity function, and the support set serves as a dictionary to *look up* the similarity with the query q . In this way, it is possible to make predictions about new classes, which were not in the training data.

The task that is important for few/one-shot learning is to learn a prediction function, $f_{su} : \mathcal{X} \rightarrow \mathcal{Y}'$, which maps into the classes given by $\mathcal{Y}'$, instead of $\mathcal{Y}$.

Definition 17.16 The task $\mathcal{T}_{su}$ for few/one-shot learning consists of outcome space $\mathcal{Y}'$ and prediction function $f_{su}(X)$ with $f_{su} : \mathcal{X} \rightarrow \mathcal{Y}'$, i.e., $\mathcal{T}_{su} = \{\mathcal{Y}', f_{su}(X)\}$.

training

$$D = \{(x_i, y_i)\}_{i=1}^n \text{ with } x_i \in \mathcal{X} \text{ and } y_i \in \mathcal{Y}$$

→

Learning **similarity function** $g : (x_i, x_j) \rightarrow \mathbb{R}^+$
with $x_i, x_j \in \mathcal{X}$

Use g and D_{su} to construct a **prediction function** f_{su}

$$f_{su} : \underset{j}{\operatorname{argmin}} \{g(x_{su,i}, x)\} \rightarrow \mathcal{Y}'$$

$$\text{with } D_{su} = \left\{ \{(x_{su,i}, y_{su,i})\}_{i=1}^{n_s} \right\}_{s=1}^S \text{ and } x_{su,i} \in \mathcal{X}, y_{su,i} \in \mathcal{Y}'$$

Importantly: $\mathcal{Y} \neq \mathcal{Y}'$

testing

$$f_{su} : \underset{j}{\operatorname{argmin}} \{g(x_{su,j}, q)\} \rightarrow \mathcal{Y}'$$

with $q \in \mathcal{X}$ new instance

Fig. 17.5 Overview of few/one-shot learning. There are three key components: (1) labeled data set D , (2) support set D_{su} with $\mathcal{Y}' \neq \mathcal{Y}$, and (3) query q representing a new instance for which a class label should be predicted. For the testing, the prediction function f_{su} is used to evaluate the similarity between q and the instances in the support set D_{su}

The distinction between $\mathcal{Y}'$ and $\mathcal{Y}$ may appear strange at first, because it means the classes of the training data and the testing data are different. So how can one learn from the instances provided by the training data for the testing data, when the outcome spaces are entirely different? The trick of few/one-shot learning is to assume that the similarity among instances in the training data and the testing data are identical. Hence, learning such a similarity function in the form of the function g allows to learn from the training data for the testing data despite the fact that $\mathcal{Y}' \neq \mathcal{Y}$.

We would like to remark that the above assumption about the similarity among instances in the training data and the testing data determines the quality of the outcome. Specifically, for infinitely large training data, it should be possible to learn the similarity function g with high accuracy. However, in the case where the similarity in the testing data is not captured by g , the prediction function f_{su} will not be able to provide meaningful results. Strictly, this is true irrespective of the sample

size of the training data and the number of instances in the support set. Hence, if the similarity assumption is violated, no learning occurs even in the limit of infinitely large sample sizes.

Based on the above definitions, few/one-shot learning can now be defined as follows.

Definition 17.17 Given a training data set D and a support set D_{su} , *few/one-shot learning* is the process of improving a prediction function, $f_{su} : \mathcal{X} \rightarrow \mathcal{Y}'$, for task $\mathcal{T}_{su}$ by utilizing D and D_{su} .

17.7.4.1 Methodological Approaches

In order to establish a few/one-shot learning model, there are essentially two main conceptual approaches:

1. Semantic transfer via similarities
2. Semantic transfer via features

The semantic transfer via similarities means that knowledge extracted from the training data is utilized for unknown classes via learning similarity concepts. An example for this is the Siamese network used in [282]. There, the authors learn an image verification task instead of predicting the classes of instances directly. Conceptually, this means to learn the similarity (or lack thereof) between pairs of instances. This network is trained for D and then utilized with D_{su} , where an instance from D_{su} , i.e., $x_{su,i}$, is used together with a query x . If x is similar to $x_{su,i}$, then the predicted class is $y_{su,i}$. The semantic transfer via features was suggested in [25]. The authors showed that the similarity, of novel features with existing features, learned from training data can help in feature adaptation.

Recently, deep learning approaches have been used, for instance, in [482], where a neural architecture, called Matching Networks, utilizing an augmented memory including an attention kernel, was introduced. Another example is Relation Network (RelNet) introduced in [451]. RelNet learns an embedding and a deep nonlinear distance metric with a convolutional neural network for comparing query and sample items.

17.7.5 Transfer Learning

The basic idea of transfer learning (TL) is to utilize information from one task to improve the learning of a second one. In order to distinguish the two tasks from each other, the former is called source task and the latter target task [372, 498, 527]. For each task, we distinguish the corresponding domain and data. In Fig. 17.6, we show a visualization of the underlying idea of transfer learning.

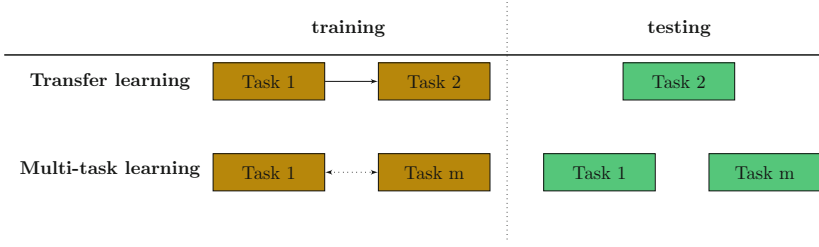


Fig. 17.6 Visualization of training and testing for transfer learning (top) and multi-task learning (bottom). For transfer learning, task 1 is usually called source task and task 2 target task. A crucial difference between transfer learning and multi-task learning is that for the latter all tasks are equal, whereas the former focuses only on task 2 (the target task). Furthermore, it is important to note that for multi-task learning, all tasks are evaluated independently from each other

Similar to supervised learning (see above) for transfer learning, we also need the definition of a domain, $\mathcal{D}$, and a task, $\mathcal{T}$.

Definition 17.18 A domain $\mathcal{D}$ consists of a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \{X_1, \dots, X_n\} \in \mathcal{X}$, i.e., $\mathcal{D} = \{\mathcal{X}, P(X)\}$.

Definition 17.19 A task $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a prediction function $f(X)$ with $f : \mathcal{X} \rightarrow \mathcal{Y}$, i.e., $\mathcal{T} = \{\mathcal{Y}, f(X)\}$.

The prediction function $f(X)$ is learned from a data set $D = \{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ for $i \in \{1, \dots, n\}$, where n is the sample size. Some machine learning methods provide explicitly probabilistic estimates of f in the form of conditional probability distributions, i.e., $f(X) = P(Y|X)$. Hence, this is a generalized form of a prediction function because in the deterministic case this reduces to a delta distribution $\delta_{x,y}$ given by

$$\delta_{x,y_i} = \begin{cases} 1 & \text{if } x = x_i \text{ with } (x_i, y_i) \\ 0 & \text{otherwise} \end{cases} \quad (17.61)$$

For transfer learning, one needs to distinguish between two kinds of domains and tasks which are called source domain, $\mathcal{D}_S$, and source task, $\mathcal{T}_S$, as well as target domain, $\mathcal{D}_T$, and target task, $\mathcal{T}_T$, with corresponding source data, D_S , and target data, D_T . From these, one can now formally define transfer learning.

Definition 17.20 Given a source domain, $\mathcal{D}_S$, with source task, $\mathcal{T}_S$, and target domain, $\mathcal{D}_T$, with target task, $\mathcal{T}_T$, *transfer learning* is the process of improving the prediction function, f_T , of the target task using $\mathcal{D}_S$ and $\mathcal{T}_S$.

The above definition is quite general in the sense that it does not specify various aspects. Hence, specifying these leads to different subtypes of transfer learning. In the following, we distinguish various subtypes from each other.

- Case $\mathcal{D}_S = \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$: This corresponds to the traditional machine learning setting when we learn f_S from source data D_S and continue the learning process with target data D_T , where the resulting prediction function is renamed to f_T . From this, it follows that transfer learning is obtained from $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$. Here, it is important to emphasize the “or” between the two conditions yields three different cases.
- Case $\mathcal{D}_S \neq \mathcal{D}_T$: Given that $\mathcal{D}_S = \{\mathcal{X}_S, P_S(X)\}$ and $\mathcal{D}_T = \{\mathcal{X}_T, P_T(X)\}$, this can either correspond to $\mathcal{X}_S \neq \mathcal{X}_T$ or $P_S(X) \neq P_T(X)$.
 - Homogeneous transfer learning: The case where the feature space of the source domain and target domain are the same, i.e., $\mathcal{X}_S = \mathcal{X}_T$, is called homogeneous transfer learning.
 - Heterogeneous transfer learning: The case where the feature space of the source domain and target domain are different, i.e., $\mathcal{X}_S \neq \mathcal{X}_T$, is called heterogeneous transfer learning.
 - $P_S(X) \neq P_T(X)$.
- Case $\mathcal{T}_S \neq \mathcal{T}_T$: Given that $\mathcal{T}_S = \{\mathcal{Y}_S, f_S(X)\}$ and $\mathcal{T}_T = \{\mathcal{Y}_T, f_T(X)\}$, this can correspond to either $\mathcal{Y}_S \neq \mathcal{Y}_T$ or $f_S(X) \neq f_T(X)$.
 - $\mathcal{Y}_S \neq \mathcal{Y}_T$: This case means that the label space of the source task and the target task is different. For instance, this can be due to a different number of classes in the source task and target task.
 - $f_S(X) \neq f_T(X)$: Given that the prediction functions generalize to conditional probability distributions, this means $P_S(Y|X) \neq P_T(Y|X)$.

17.7.5.1 Methodological Approaches

For transfer learning, a variety of different perspectives have been suggested for the categorization of this learning paradigm. For instance, one could assume a view with respect to traditional paradigms, distinguishing between inductive, transductive, and unsupervised transfer learning [372] or a model-based view [527]. However, the most common categorization is based on “what to transfer” [372]:

1. Feature-based TL
2. Parameter-based TL
3. Instance-based TL
4. Relational-based TL

1. For feature-based TL, good feature representations are learned from the source task, and they are assumed to be useful for the target task as well. Hence, in this case, the knowledge transfer between source task and target task is via learning feature representations. 2. For parameter-based TL, some parameters or prior distribution of hyperparameters are transferred from the source task to the target task. This assumes a similarity of the source model and the target model. Unlike multi-task learning, where both the source and target tasks are learned simultaneously, for

transfer learning, we may apply additional weightage to the loss of the target domain to improve overall performance. 3. The idea of instance-based TL is to reuse parts of the instances from the source task for the target task. Usually, instances cannot be used directly; instead, this is accomplished via instance weighting. 4. Relational-based TL assumes that instances are not independent and identically distributed, but they are dependent. This implies that the underlying data form a sort of network, e.g., a transcription regulatory network or a social network.

17.7.6 Multi-Task Learning

The idea of multi-task learning (MTL) compared to transfer learning is threefold. First, instead of considering exactly *two* tasks, the source and target task, in multi-task learning there can be $m > 2$ tasks. Second, these m tasks do not have one or more dedicated targets, but all tasks are equally important. That means there are m source tasks and m target tasks [69]. Third, MTL learns multiple related tasks jointly by sharing useful information among related tasks.

Formally, multi-task learning can be described as follows.

Definition 17.21 Given m learning tasks, $\{\mathcal{T}_k\}_{k=1}^m$, where all tasks or a subset of tasks are related, *multi-task learning* aims to improve each learning task $\mathcal{T}_k$ using information from some or all the other models.

For clarity, we would like to emphasize that for each learning task $\mathcal{T}_k$, there is a corresponding domain $\mathcal{D}_k = \{\mathcal{X}_k, P(X_k)\}$ and data set D_k given, from which information can be utilized. In the following, we denote the data set of task k by $D_k = \{(x_{ki}, y_{ki})\}_{i=1}^{n_k}$ with $x_{ki} \in \mathcal{X}_k$ and $y_{ki} \in \mathcal{Y}_k$ with $i \in \{1, \dots, n_k\}$, where n_k is the sample size.

For MTL there is an important special case one needs to distinguish from the general setting.

The case where $x_{ki} = x_{li}$ and $n_k = n_l = n$ for all $k, l \in \{1, \dots, m\}$ and $i \in \{1, n\}$, is called *multi-view learning*. Therefore in this case the x -values of the data D_k for all tasks are identical but can have different labels, i.e., $\mathcal{Y}_k \neq \mathcal{Y}_l$ for all $k, l \in \{1, \dots, m\}$.

17.7.6.1 Methodological Approaches

For multi-task learning, there are three key methodological approaches used to study such problems [517].

1. Feature-based MTL
2. Parameter-based MTL
3. Instance-based MTL

Feature-based MTL models assume that different tasks share the same or at least similar features. This includes also methods that perform feature selection or transformation of the original features.

Parameter-based MTL models utilize parameters between different models to relate the learning between different tasks. Examples for this include methods based on regularization or priors on model parameters. In general, this conceptual approach is very diverse with many different realizations.

Instance-based MTL models estimate weights for the membership of instances in tasks and then use all instances to learn all tasks in a weighted manner. For comprehensive reviews of multi-task learning, we refer the reader to [407, 444, 517].

17.7.7 Multi-Label Learning

The idea of multi-label learning (MLL) is to generalize the class labels of a traditional classification, having single-valued entities, to variable set sizes [190, 467, 520]. Therefore, the number of labels, as the outcome of a prediction function, is variable.

In order to formally define multi-label learning, we need to modify the definition of the data set D . Specifically, for multi-label learning, D is defined as $D = \{(x_i, Y_i)\}_{i=1}^n$ with $x_i \in \mathcal{X}$ and $Y_i \subseteq \mathcal{Y}$ where $\mathcal{Y} = \{L_1, \dots, L_q\}$. Here, Y_i can assume any subset of $\mathcal{Y}$, which makes the size, of such a set, variable.

The goal of multi-label learning is to find a prediction function, f , that maps the elements of D correctly. Formally, the task is defined as follows.

Definition 17.22 For multi-label learning, a *task* $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a prediction function $f(X)$ with $f : \mathcal{X} \rightarrow 2^{\mathcal{Y}}$, i.e., $\mathcal{T} = \{\mathcal{Y}, f(X)\}$.

Here, $2^{\mathcal{Y}}$ corresponds to the power set of $\mathcal{Y}$, which is the set of all subsets of $\mathcal{Y}$.

From the above definition, one may wonder why $2^{\mathcal{Y}}$ is not mapped to a multi-class problem. The reason for this can be visualized for $\mathcal{Y} = \{y_1, \dots, y_{20}\}$. In this case, the size of the power set is 1048576 ($= 2^{20}$). Hence, if we would map the multi-label problem to a multiclass classification, one would have 1048576 different classes. However, this results in severe learning problems for such a classifier. For this reason multi-label learning tries to be more resourceful.

17.7.7.1 Methodological Approaches

For multi-label learning, there are two key conceptual approaches, allowing a categorization of available methods as follows:

1. Problem transformation
2. Algorithm adaptation

Approaches based on problem transformation can be further subdivided into (i) transformation to binary classification, (ii) transformation to label ranking, and (iii) transformation to multiclass classification [190]. Such approaches convert a multi-label learning problem, by means of transformations, into well-established problem settings. Examples of these approaches include classifier chains [398], which transforms a multi-label learning problem into a binary classification task; calibrated label ranking [175], which maps MLL into the task of label ranking; and random k-labelsets [469], which transforms multi-label learning into the task of multiclass classification. If the mapping is performed, this is called label powerset [468].

From the definition of multi-label learning and the description of transformation methods, one may wonder why $2^{\mathcal{Y}}$ is not always directly mapped to a multiclass classification problem, because theoretically, such a mapping is always possible. However, there is a practical problem with this for large $|\mathcal{Y}|$ as discussed in the previous section.

Methods based on algorithm adaptation modify existing learning methods to adopt them to the multi-label case. In [520], four approaches are distinguished: (i) lazy learning (e.g., ML-kNN [519]), (ii) decision tree (e.g., ML-DT [78]), (iii) kernel learning (e.g., Rank-SVM [130]), and (iv) information-theoretic methods (e.g., CML [189]).

17.8 Summary

In this chapter, we discussed two fundamental aspects of learning from data: the first concerned computational learning theory and the second different definitions of learning paradigms. We have seen that computational learning theory provides a quantification of “learnability” and, hence, allows to derive bounds on the capabilities of learning algorithms. In contrast, the learning paradigms provided new frameworks for embedding particular data types with specific properties.

Learning Outcome 17: Machine Learning Paradigms

Machine learning is not a closed field, where everything has been discovered. Instead, this field is under rapid development not only with respect to novel methods but even entirely new learning paradigms.

Both of these frameworks are useful for providing an overview of learning from data beyond particular models. While computational learning theory is important for obtaining definite answers about principle learning capabilities, e.g., error bounds, the definition of learning paradigms provides clarity about tasks and the usage of data beyond a supervised learning paradigm. It can be expected that the latter, especially, will lead to the development of many new learning algorithms in the years to come because such definitions can spur the creativity to design novel

learning algorithms. Overall, the concepts discussed in this chapter provide valuable thought patterns for learning from data.

17.9 Exercises

1. Show that Eq. 17.9 holds. Hint: Utilize the linearity of the expectation value and the fact that the samples are drawn i.i.d.
2. Let $X_1, \dots, X_n$ be n independent and identically distributed (i.i.d) random variables bounded by the intervals $[a_i, b_i]$, i.e., $a_i \leq X_i \leq b_i$ for all i , and $\bar{X} = 1/n \sum_{i=1}^n X_i$. Then the following inequality holds according to Hoeffding [242]

$$Pr\left(|\bar{X} - \mathbb{E}[\bar{X}]| \geq \varepsilon\right) \leq 2\exp\left(-\frac{2n^2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right). \quad (17.62)$$

Show that this corresponds to the inequality in Eq. 17.47 by identifying the terms.

3. Compare transfer learning with supervised learning and discuss the differences. How can one convert a transfer learning problem to a supervised learning problem?
4. Compare multi-task learning with transfer learning and discuss the differences. How can one convert a multi-task learning problem to a transfer learning problem?

Chapter 18

Generalization Error and Model Assessment



18.1 Introduction

This chapter provides the conceptual roof for all the previous chapters. We started this book with a discussion on general topics, e.g., error measures and resampling methods, which allow on the one hand an intuitive understanding and on the other hand a general application to a large variety of different problems. Then, we discussed important core methods that enable to conduct different forms of data analysis. Now, we return to a general topic, however, this time of a more abstract nature.

The central topic of this chapter is the *generalization error*. Put simply, the generalization error provides a theoretical quantification of the performance of a prediction model. Furthermore, it enables to understand the bias-variance trade-off, error-complexity curves, learning curves, and the connection of these concepts with model selection and model assessment. Model selection and model assessment have been already discussed in Chap. 12; however, in this chapter, we approach these problems from a more fundamental view. Furthermore, other topics discussed in this chapter, e.g., learning curves, are applicable to the methods discussed in Parts II and III of this book. This underlines the fundamental character of the concepts around the generalization error. As we will see, the concepts discussed in this chapter can be used as guiding principles for general data science projects.

For the analysis of supervised learning models, e.g., regression or classification methods [56, 79, 139, 217, 222, 416], where prediction errors can be estimated, model selection and model assessment are the key concepts to find the best model for a given data set. Interestingly, regarding the definition of best model, there are two complementary approaches with a different underlying philosophy [108, 168]. One is defining best model based on its predictiveness and the other based on its descriptiveness. The latter approach aims to identify the true model, whose interpretation leads to a deeper understanding of the generated data and the underlying processes, which generated them.

18.2 Overall View on Model Diagnosis

Regardless of the statistical model under investigation, e.g., for classification or regression, there are two basic questions one needs to address: (1) How to choose between competing models and (2) how to evaluate them? Both questions aim to diagnose the models.

The above informal questions are formalized by the following two statistical concepts [222]:

Model selection: Estimate the performance of different models in order to choose the best model.

Model assessment: For the best model, estimate its generalization error.

Briefly, model selection refers to the process of optimizing a model family or a model candidate. This includes the selection of a model itself from a set of potentially available models and the estimation of its parameters. The former can relate to deciding which regularization method (e.g., ridge regression, LASSO, or elastic net) should be used, whereas the latter corresponds to estimating the parameters of the selected model. On the other hand, model assessment means the evaluation of the generalization error (also called test error) of the finally selected model, for an independent data set. This task aims to estimate the “true prediction error” as could be obtained from an infinitely large test data set. Both concepts are based on the utilization of data to quantify numerically the properties of models.

For simplicity, let’s assume we are given a very large (or arbitrarily large) data set D . The best approach for both problems would be to randomly divide the data into three non-overlapping sets:

1. Training data set: D_{train}
2. Validation data set: D_{val}
3. Test data set: D_{test}

By “very large data set,” we mean a situation where the sample sizes, i.e., n_{train} , n_{val} , and n_{test} , for all three data sets are large without necessarily being infinite, but an increase in their sizes would not lead to changes in the model evaluation.

Formally, the relation between the three data sets can be written as follows:

$$D = D_{\text{train}} \cup D_{\text{val}} \cup D_{\text{test}} \quad (18.1)$$

$$\emptyset = D_{\text{train}} \cap D_{\text{val}} \quad (18.2)$$

$$\emptyset = D_{\text{train}} \cap D_{\text{test}} \quad (18.3)$$

$$\emptyset = D_{\text{val}} \cap D_{\text{test}} \quad (18.4)$$

Based on these data, the training set would be used to estimate or learn the parameters of the models. This is called model fitting. The validation data would be used to estimate a selection criterion for model selection, and the test data would be used to estimate the generalization error of the final chosen model.

In practice, the situation is more complicated since D is typically not arbitrarily large. In the following sections, we discuss, first, model assessment and then model selection, in detail. The order of our discussion is reversed to the order one would perform a practical analysis. However, to facilitate the understanding of the concepts, this order is beneficial.

18.3 Expected Generalization Error

Let's assume that we have a general model of the form

$$y = f(x, \beta) + \varepsilon \quad (18.5)$$

mapping the input x to an output y , as defined by the function f . The mapping varies by a noise term $\varepsilon \sim N(0, \sigma^2)$ representing, e.g., measurement errors. We want to approximate the true (but unknown) mapping function f by a model g that depends on parameters β , i.e.,

$$\hat{y} = g(x, \hat{\beta}(D)) = \hat{g}(x, D). \quad (18.6)$$

Here, the parameters β are estimated from a training data set D (strictly denoted by D_{train}); hence, the parameters β are functions of the training set, i.e., $\hat{\beta}(D)$. The “hat” indicates that the parameters β are estimated using the data D . As a shortcut, we write $\hat{g}(x, D)$ instead of $g(x, \hat{\beta}(D))$.

Based on these entities, we can define the following measures to evaluate models:

$$SST = TSS = \sum_{i=1}^n (y_i - \bar{y})^2 = \|Y - \bar{Y}\|^2; \quad (18.7)$$

$$SSR = ESS = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2 = \|\hat{Y} - \bar{Y}\|^2; \quad (18.8)$$

$$SSE = RSS = \sum_{i=1}^n (\hat{y}_i - y_i)^2 = \sum_{i=1}^n e_i^2 = \|\hat{Y} - Y\|^2. \quad (18.9)$$

Here, $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$ is the mean value of the predictor variable, and $e_i = \hat{y}_i - y_i$ are the residuals, whereas

- SST is the **sum of squares total**, also called **total sum of squares** (TSS).
- SSR is the **sum of squares due to regression** (variation explained by linear model), also called the **explained sum of squares** (ESS).

- SSE is the **sum of squares due to errors** (unexplained variation)—also called **residual sum of squares** (RSS).

There is a remarkable property for the sum of squares, which is given by:

$$\underbrace{\text{SST}}_{\text{total deviation}} = \underbrace{\text{SSR}}_{\text{deviation of regression from mean}} + \underbrace{\text{SSE}}_{\text{deviation of regression}} \quad (18.10)$$

This relationship is called *partitioning of sum of squares* [428].

Furthermore, to summarize the overall predictions of a model, the **mean squared error** (MSE), given by

$$\text{MSE} = \frac{\text{SSE}}{n}, \quad (18.11)$$

is a useful quantity.

The general problem, when dealing with predictions, is that we would like to know about the generalization abilities of our model. Specifically, for a given training data set D_{train} , we can estimate the parameters of our model β leading to estimates $g(x, \hat{\beta}(D_{\text{train}})) = \hat{g}(x, D_{\text{train}})$. Ideally, we would like to have that $y \approx \hat{g}(x, D_{\text{train}})$ for every data point (x, y) drawn from the underlying population, i.e., $(x, y) \sim P$. In order to assess this quantitatively, a *loss function*, or simply a loss, is defined. Frequent choices for a loss are:

- The absolute error, defined by

$$L(y, \hat{g}(x, D_{\text{train}})) = |y - \hat{g}(x, D_{\text{train}})| \quad (18.12)$$

- The squared error, defined by

$$L(y, \hat{g}(x, D_{\text{train}})) = (y - \hat{g}(x, D_{\text{train}}))^2. \quad (18.13)$$

If one would use only data points from a training set, i.e., $(x, y) \in D_{\text{train}}$, to assess the loss, these estimates are usually overly optimistic and lead to much smaller errors as if data points are used from all possible values, i.e., $(x, y) \sim P$ where P is the distribution of all possible values. Formally, we can write this as an expectation value with respect to distribution P , as follows:

$$E_{\text{test}}(D_{\text{train}}, n_{\text{train}}) = \mathbb{E}_P \left[L(y, \hat{g}(x, D_{\text{train}})) \right]. \quad (18.14)$$

The expectation value in Eq. 18.14 is called the *generalization error* of the model. This error is also called *out-of-sample error* or simply *test error*. The latter name emphasizes the important fact that test data are used for the evaluation of the

prediction error (as represented by the distribution P) of the model, but training data are used to learn its parameters (as indicated by D_{train}).

From Eq. 18.14, one can see that we have an unwanted dependency on the training set D_{train} . In order to remove this, we need to assess the generalization error of the model, given by $\hat{\beta}(D_{train})$ by expressing the expectation value with respect to all training sets, i.e.,

$$E_{test}(n_{train}) = \mathbb{E}_{D_{train}} \mathbb{E}_P [L(y, \hat{g}(x, D_{train}))]. \quad (18.15)$$

This is the *expected generalization error* of the model, also called *expected out-of-sample error* [3], which is no longer dependent on any particular estimates of $\hat{\beta}(D_{train})$ via D_{train} . Hence, this error provides the desired assessment of a model for its generalization capability.

It is important to emphasize that the training sets, D_{train} , are not infinitely large but they all have the same finite sample size n_{train} . Hence, the expected generalization error in Eq. 18.15 is only independent of a particular training set but still depends on the size of these sets. In Sect. 18.6, we will explore this dependency when discussing learning curves.

From the above derivation, one can see that the expected generalization error in Eq. 18.15 is a *population estimate*. That means, its evaluation is based on expectation values over populations, namely, the population of all data points, P , and the population of all training data, D_{train} , of size n_{train} . Hence, this is a theoretical entity. When working with data, one requires an approximation of the population estimate via a *sample estimate*. For such an approximation, the resampling methods discussed in Chap. 4 can be used. This implies also that for model assessment, one wants to estimate the expected generalization error. Hence, theoretically, the expected generalization error is the desired measure for model assessment.

Although the expected generalization error cannot be evaluated in general, it can be used to derive theoretical insights. Specifically, in the next section, we will use the expected generalization error to derive a decomposition known as bias-variance trade-off [179, 188, 283, 497].

18.4 Bias-Variance Trade-Off

In this section, we show how the expected generalization error, of the model in Eq. 18.15, can be used to derive an error decomposed into different components. This decomposition is known as bias-variance trade-off [179, 188, 283, 497], and the three components are denoted bias, variance, and noise. We will see that the result provides valuable insights for understanding the influence of the model complexity on the prediction error.

In the following, we denote the training set by D to simplify the notation. Furthermore, we write the expectation value with respect to the distribution P by $\mathbb{E}_{x,y}$, and not as $\mathbb{E}_P$ as in Eq. 18.15, as a short form for $\mathbb{E}_{P(x,y)}$. This allows to apply the probability rule of the form

$$P(x, y) = P(y|x)(x), \quad (18.16)$$

making the derivation more explicit, as we will see below in Eqs. 18.26 and 18.23.

We start the derivation from the expected generalization error with the squared error as the loss, i.e.,

$$\mathbb{E}_{D_{\text{train}}} \mathbb{E}_P [L(y, \hat{g}(x, D_{\text{train}}))] = \mathbb{E}_D \mathbb{E}_{x,y} [(y - \hat{g}(x, D))^2] \quad (18.17)$$

$$\begin{aligned} & \mathbb{E}_D \mathbb{E}_{x,y} [(y - \hat{g}(x, D))^2] \\ &= \underbrace{\mathbb{E}_D \mathbb{E}_{x,y}}_{\text{independent}} [(y - \mathbb{E}_D[\hat{g}(x, D)] + \mathbb{E}_D[\hat{g}(x, D)] - \hat{g}(x, D))^2] \end{aligned} \quad (18.18)$$

$$\begin{aligned} &= \mathbb{E}_{x,y} \mathbb{E}_D [(y - \mathbb{E}_D[\hat{g}(x, D)])^2] + \mathbb{E}_{x,y} \mathbb{E}_D [(\mathbb{E}_D[\hat{g}(x, D)] - \hat{g}(x, D))^2] \\ &\quad + 2\mathbb{E}_{x,y} \mathbb{E}_D [(y - \mathbb{E}_D[\hat{g}(x, D)])(\mathbb{E}_D[\hat{g}(x, D)] - \hat{g}(x, D))] \end{aligned} \quad (18.19)$$

$$\begin{aligned} &= \mathbb{E}_{x,y} \mathbb{E}_D [(y - \mathbb{E}_D[\hat{g}(x, D)])^2] + \mathbb{E}_{x,y} \mathbb{E}_D [(\mathbb{E}_D[\hat{g}(x, D)] - \hat{g}(x, D))^2] \\ &\quad + 2\mathbb{E}_{x,y} \left[\underbrace{(y - \mathbb{E}_D[\hat{g}(x, D)])}_{\text{independent of } D} \mathbb{E}_D [(\mathbb{E}_D[\hat{g}(x, D)] - \hat{g}(x, D))] \right] \end{aligned} \quad (18.20)$$

$$= \mathbb{E}_{x,y} [(y - \mathbb{E}_D[\hat{g}(x, D)])^2] + \mathbb{E}_{x,y} \mathbb{E}_D [(\mathbb{E}_D[\hat{g}(x, D)] - \hat{g}(x, D))^2] \quad (18.21)$$

$$= \mathbb{E}_{x,y} [(y - \bar{g}(x))^2] + \mathbb{E}_{x,y} \mathbb{E}_D [(\bar{g}(x) - \hat{g}(x, D))^2] \quad (18.22)$$

$$= \mathbb{E}_{x,y} [(y - \bar{g}(x))^2] + \mathbb{E}_D \mathbb{E}_x \mathbb{E}_{y|x} [\underbrace{(\bar{g}(x) - \hat{g}(x, D))^2}_{\text{independent of } y}] \quad (18.23)$$

$$= \mathbb{E}_{x,y} [(y - \bar{g}(x))^2] + \mathbb{E}_D \mathbb{E}_x [(\bar{g}(x) - \hat{g}(x, D))^2] \quad (18.24)$$

$$= \text{bias}^2 + \text{variance}$$

In Eq. 18.23, we used the independence of the sampling processes for D and (x, y) to change the order of the expectation values. This allowed to evaluate the conditional expectation value $\mathbb{E}_{y|x}$ because the argument is independent of y .

In Eq. 18.22, we used the following short form

$$\bar{g}(x) = \mathbb{E}_D [\hat{g}(x, D)] \quad (18.25)$$

to write the expectation value of $\hat{g}$ with respect to D giving a mean model $\bar{g}$ over all possible training sets D . Due to the fact that this expectation value integrates over all possible values of D , the resulting $\bar{g}(x)$ depends no longer on it.

By utilizing the conditional expectation value

$$\mathbb{E}_{x,y} = \mathbb{E}_x \mathbb{E}_{y|x}, \quad (18.26)$$

we can further analyze the first term of the above derivation (highlighted in green) using the following relationship:

$$\mathbb{E}_{x,y} y = \mathbb{E}_x \mathbb{E}_{y|x} y = \mathbb{E}_x \bar{y}(x) = \bar{y}. \quad (18.27)$$

Here, it is important to note that $\bar{y}(x)$ is a function of x whereas $\bar{y}$ is not, because the expectation value $\mathbb{E}_x$ integrates over all possible values of x . For clarity reasons, we want to note that y means actually $y(x)$, but to simplify the notation, we suppress this argument so that the derivation is more readable.

Specifically, by utilizing this term, we obtain the following decomposition:

$$\mathbb{E}_{x,y} \left[(y - \bar{g}(x))^2 \right] = \mathbb{E}_{x,y} \left[(y - \mathbb{E}_D [\hat{g}(x, D)])^2 \right] \quad (18.28)$$

$$= \mathbb{E}_{x,y} \left[(y - \bar{y}(x) + \bar{y}(x) - \mathbb{E}_D [\hat{g}(x, D)])^2 \right] \quad (18.29)$$

$$= \mathbb{E}_{x,y} \left[(y - \bar{y}(x))^2 \right] + \mathbb{E}_{x,y} \left[(\bar{y}(x) - \mathbb{E}_D [\hat{g}(x, D)])^2 \right] \\ + 2\mathbb{E}_{x,y} \left[(y - \bar{y}(x))(\bar{y}(x) - \mathbb{E}_D [\hat{g}(x, D)]) \right] \quad (18.30)$$

$$= \mathbb{E}_{x,y} \left[(y - \bar{y}(x))^2 \right] + \mathbb{E}_{x,y} \left[\underbrace{(\bar{y}(x) - \mathbb{E}_D [\hat{g}(x, D)])^2}_{\text{independent of } y} \right] \\ + 2 \mathbb{E}_x \mathbb{E}_{y|x} \left[(y - \bar{y}(x)) \underbrace{(\bar{y}(x) - \mathbb{E}_D [\hat{g}(x, D)])}_{\text{independent of } y} \right] \quad (18.31)$$

$$= \mathbb{E}_{x,y} \left[(y - \bar{y}(x))^2 \right] + \mathbb{E}_{x,y} \left[(\bar{y}(x) - \mathbb{E}_D [\hat{g}(x, D)])^2 \right] + \\ + 2 \mathbb{E}_x \left[(\bar{y}(x) - \bar{y}(x))(\bar{y}(x) - \mathbb{E}_D [\hat{g}(x, D)]) \right] \quad (18.32)$$

$$= \mathbb{E}_{x,y} \left[(y - \bar{y}(x))^2 \right] + \mathbb{E}_x \left[(\bar{y}(x) - \mathbb{E}_D [\hat{g}(x, D)])^2 \right] \quad (18.33) \\ = \text{Noise} + \text{Bias}^2$$

Taking together, we obtain the following combined result:

$$\mathbb{E}_D \mathbb{E}_{x,y} \left[(y - \hat{g}(x, D))^2 \right]$$

$$\begin{aligned}
&= \mathbb{E}_{x,y} \left[(y - \bar{y}(x))^2 \right] + \mathbb{E}_x \mathbb{E}_D \left[(\mathbb{E}_D [\hat{g}(x, D)] - \hat{g}(x, D))^2 \right] \\
&\quad + \mathbb{E}_x \left[(\bar{y}(x) - \mathbb{E}_D [\hat{g}(x, D)])^2 \right] \tag{18.34}
\end{aligned}$$

$$\begin{aligned}
&= \mathbb{E}_{x,y} \left[(y - \bar{y}(x))^2 \right] + \mathbb{E}_x \mathbb{E}_D \left[(\bar{g}(x) - \hat{g}(x, D))^2 \right] \\
&\quad + \mathbb{E}_x \left[(\bar{y}(x) - \bar{g}(x))^2 \right] \tag{18.35}
\end{aligned}$$

$$= \text{Noise} + \text{Variance} + \text{Bias}^2$$

- **Noise:** This term measures the variability within the data, without considering any model. The noise cannot be reduced because it does not depend on the training data D or g or any other parameter under our control. Hence, it is a characteristic of the distribution P from which the data are drawn. For this reason, this component is also called *irreducible error*.
- **Variance:** This term measures the model variability with respect to the changes in the training sets. This variance can be reduced by using less complex models g . However, this can increase the bias (underfitting).
- **Bias:** This term measures the inherent error that you obtain from your model even with an infinitely large training data. The bias can be reduced by using more complex models, g . However, this can increase the variance (overfitting).

In Fig. 18.1, we show a visualization of the model assessment problem and its interpretation based on the bias-variance trade-off. In Fig. 18.1a, the blue curve corresponds to a model family, e.g., a regression model with a fixed number of covariates, and each point along this line corresponds to a particular model obtained from estimating the parameters of the model from a data set. The dark brown point corresponds to the true (but unknown) model and a data set generated by this model. Specifically, this data set has been obtained in the error-free case, i.e., $\varepsilon_i = 0$ for all samples i . If another data set is generated from the true model, this data set will vary to some extent because of the noise term ε_i , which is usually not zero. This variation is indicated by the large (light) brown circle around the true model.

In the case where the model family does not include the true model, there will be a bias corresponding to the distance between the true model and the estimated model indicated by the blue point along the curve of the model family. Specifically, this bias is measured between the error-free data set generated by the true model and the estimated model based on this data set. Also, the estimated model will have some variability indicated by the (light) blue circle around the estimated model. This corresponds to the variance of the estimated model.

It is important to realize that there is no possibility to compare the true model with the estimated model directly with each other, because the true model is usually unknown. The only situation when the true model is known is for simulation studies where all the entities involved are known and we have perfect control over them. We

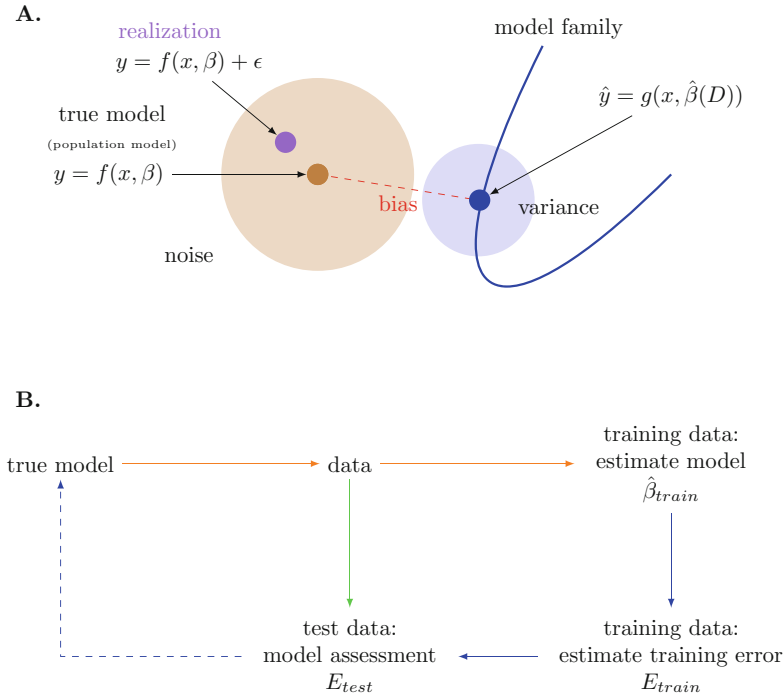


Fig. 18.1 Insights from the bias-variance trade-off for model assessment

will make use of this in the next section, when we show some numerical simulation examples.

In reality, however, the comparison between the true model and the estimated model is carried out indirectly, via data that have been generated by the true model. Hence, these data are serving two purposes. First, they are used to estimate the parameters of the model. For this estimation, the training data are used. If one uses the same training data to evaluate the prediction error of this model, the prediction error is called **training error**

$$E_{train} = E_{train}(D_{train}). \quad (18.36)$$

E_{train} is also called *in-sample error*.

Second, the data are used to assess the estimated model by quantifying its prediction error. For this estimation, the test data are used. In this case, the prediction error is called **test error** or *out-of-sample error*

$$E_{test} = E_{test}(D_{test}). \quad (18.37)$$

In Fig. 18.1b, we summarize the usage of the different data sets, i.e., training data and testing data, for model assessment.

It is important to note that a prediction error is always evaluated with respect to a given data set. For this reason, we emphasized this explicitly in Eqs. 18.36 and 18.37. Unfortunately, this information is frequently omitted in the literature, and this can lead to some confusion in the meaning of the prediction error.

We want to emphasize that the training error is only defined as **sample estimate** but not as population estimate, because the training data set is always finite. That means, the **sample training error**, in Eq. 18.36, is estimated by

$$E_{train}(n_{train}) = \frac{1}{n_{train}} \sum_{i=1}^{n_{train}} L(y_i, \hat{g}(x_i, D_{train})), \quad (18.38)$$

assuming that the sample size of the training data is n_{train} with $D_{train} = \{(x_i, y_i)\}_{i=1}^{n_{train}}$.

In contrast, the test error in Eq. 18.37 (expected generalization error) corresponds to the population estimate given in Eq. 18.15. In practice, this needs to be approximated by a sample estimate, similar to Eq. 18.38, of the form

$$E_{test}(n_{test}, n_{train}) = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} L(y_i, \hat{g}(x_i, D_{train})), \quad (18.39)$$

for a test data set with n_{test} samples. For a finite size of the test data, the **sample test error** depends not only on n_{train} but also n_{test} .

18.5 Error-Complexity Curves

Before we apply the above results to an example, we want to go one step further, discussing **error-complexity curves**. In general, error-complexity curves utilize the above concepts to show the dependency of either the training error or the test error on the complexity of a statistical model.

Definition 18.1 (Error-Complexity Curves for Training and Test Error) Error-complexity curves show the training error and test error as functions of the model complexity. The models underlying these curves are estimated from training data with a fixed sample size.

Since any test error can be decomposed in a bias and variance term, as we have seen in Sect. 18.4, the above definition implies that error-complexity curves can be also used to show the dependency of the bias-variance decomposition of the test error on the model complexity.

Definition 18.2 (Error-Complexity Curves for Bias-Variance Decomposition)

Error-complexity curves show the bias-variance decomposition of the test error on the model complexity. The models underlying these curves are estimated from the training data with a fixed sample size.

Overall, this means that error-complexity curves do not introduce a new concept but apply the above results to different models with varying complexity. In other words, by fixing the model complexity to a particular value, one obtains results for the training and test error and the bias-variance decomposition.

Now, we can summarize all results obtained so far using an example.

18.5.1 Example: Linear Polynomial Regression Model

In this section, we show numerical examples for the above derivations. Due to the fact that the above entities are all population estimates, we may need to use simulations, because they enable us to generate arbitrarily large data sets. In the following, we study linear polynomial regression models.

In Fig. 18.2, we show an example, where the true model, depicted in blue, corresponds to

$$f(x, \boldsymbol{\beta}) = 25 + 0.5x + 4x^2 + 3x^3 + x^4, \quad (18.40)$$

where $\boldsymbol{\beta} = (25, 0.5, 4, 3, 1)^T$ (see Eq. 18.5). The true model is a mixture of polynomials of different degree, where the highest degree is 4, corresponding to a linear polynomial regression model. From this model, we generate training data with a sample size of $n = 30$ (shown by black points), which are used to fit different regression models.

The general model family we use for the regression model is given by

$$g(x, \boldsymbol{\beta}) = \sum_{i=0}^d \beta_i x^i = \beta_0 + \beta_1 x + \cdots + \beta_d x^d. \quad (18.41)$$

That means, we are fitting linear polynomial regression models with a maximal degree of d . The highest degree corresponds to the model complexity of the polynomial family. For our analysis, we are using polynomials with degree d from 1 to 10 and fit these to the training data. The results of these regression analyses are shown as red curves in Fig. 18.2a–j.

In Fig. 18.2a–j, the blue curves show the true model and the red curves the fitted models, and the black points correspond to the training data. The shown results correspond to individual model fits, i.e., no averaging has been performed. Furthermore, for all results, the sample size of the training data was kept fixed (varying sample sizes are studied in Sect. 18.6). Since the degree of the polynomial indicates the complexity of the fitted model, the shown models correspond to

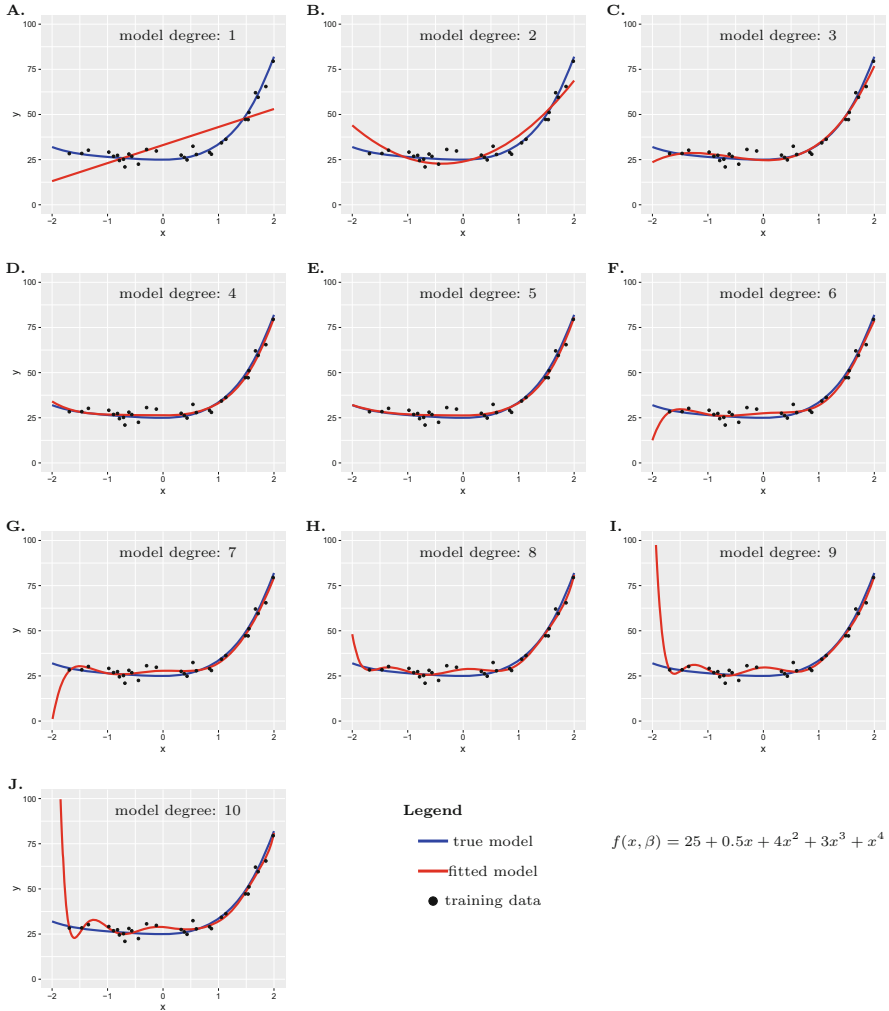


Fig. 18.2 Different examples for fitted linear polynomial regression models with varying degree d , ranging from 1 to 10. The model degree indicates the highest polynomial degree of the fitted model. These models correspond to different model complexities from low complexity ($d = 1$) to high complexity ($d = 10$). The blue curves show the true model, the red curves the fitted models and the black points correspond to the training data. The shown results correspond to individual fits, i.e., no averaging has been performed. For all results, the sample size of the training data was kept fixed

different model complexities from low-complexity ($d = 1$) to high-complexity ($d = 10$) models.

One can see that for both low and high degrees of the polynomials, there are clear differences between the true model and the fitted models. However, these differences

have a different origin. For low-degree models, the differences come from the low complexity of the models, which are not flexible enough to adapt to the variability of the training data. Put simply, the model is too simple. This behavior corresponds to an underfitting of the data (caused by high bias, as explained in detail below). In contrast, for high degrees, the model is too flexible for the few available training samples. In this case the model is too complex for the training data. This behavior corresponds to an overfitting of the data (caused by high variance, as explained in detail below).

18.5.2 Example: Error-Complexity Curves

Based on the above model of linear polynomial regression, we can now show error-complexity curves. Specifically, in Fig. 18.3, we show two different types of results. The first type of results, shown in Fig. 18.3a, c, e, and f, corresponds to numerical simulation results fitting a linear polynomial regression to training data, whereas the second type, shown in Fig. 18.3b and d (highlighted using the dashed brown rectangle), corresponds to idealized results that hold for general statistical models beyond our studied examples. The numerical simulation results in Fig. 18.3a, c, e, and f have been obtained from averaging over an ensemble of repeated model fits. For all these fits, the sample size of the training data was kept fixed.

We start by discussing the results shown in Fig. 18.3a. These results provide the error-complexity curves for the expected training and test errors for the different polynomials. From Fig. 18.3a, one can see that the training error decreases with increasing polynomial degree. In contrast, the test error is U-shaped. Intuitively, it is clear that more complex models fit the training data better; however, there should be an optimal model complexity, and going beyond should worsen the prediction performance. The training error alone does not clearly reflect this, and for this reason, estimates of the test error are needed. Figure 18.3b shows idealized results for characteristic behavior of the training and test error for general statistical models.

In Fig. 18.3c, we show the decomposition of the test error into its noise, bias, and variance component. The noise is constant for all polynomial degrees, whereas the bias is monotonously decreasing and the variance is increasing. Note that this behavior is generic beyond the shown examples. For this reason, we show, in Fig. 18.3d, the idealized decomposition (neglecting the noise since it has a constant contribution).

In Fig. 18.3e, we show the percentage breakdown of the noise, bias, and variance for each polynomial degree. In this representation, the behavior of the noise is not constant since the decomposition is nonlinear for different complexity values of the model. The numerical values of the percentage breakdown depend on the degree of the polynomial and vary as shown. In Fig. 18.3f, we show the same results as in Fig. 18.3e but without the noise part. From these representations, one can see that simple models have a high bias and a low variance, whereas complex models have

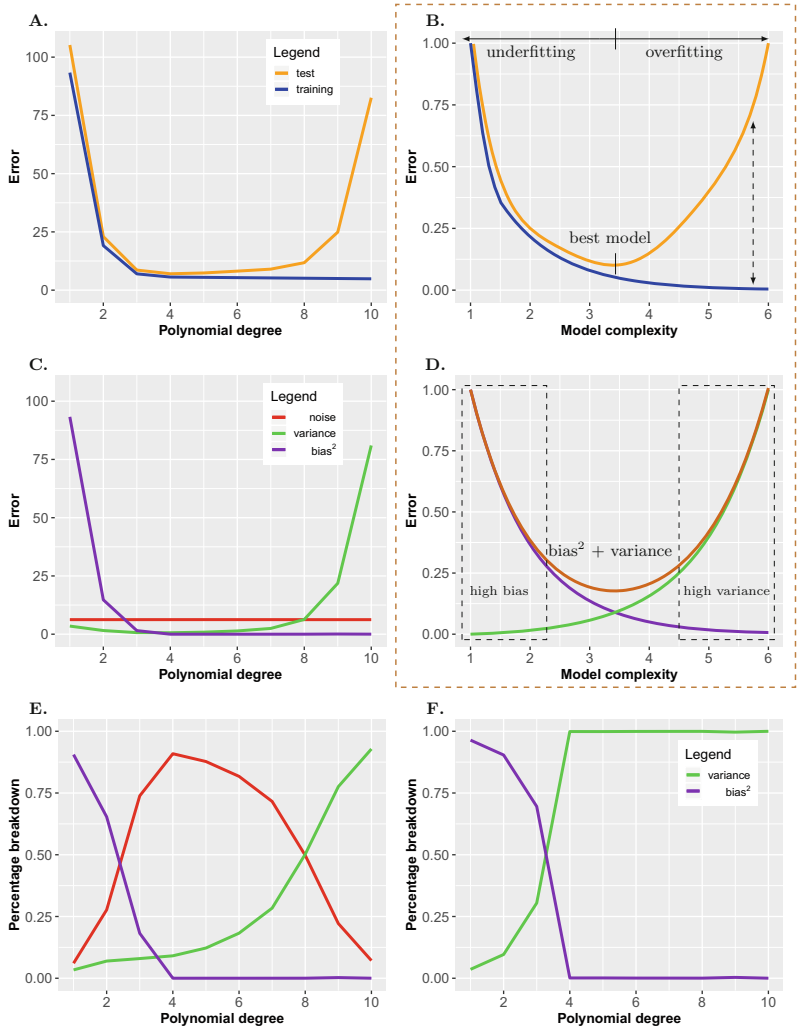


Fig. 18.3 Error-complexity curves showing the prediction error (training and test error) against the model complexity. The panels (a, c, e, and f) show numerical simulation results for a linear polynomial regression model. The model complexity is expressed by the degree of the highest polynomial. For the shown analysis, the training data set was fixed. (b) Idealized error curves for general statistical models. (c) Decomposition of the expected generalization error (test error) into noise, bias, and variance. (d) Idealized decomposition in bias and variance. (e) Percentage breakdown of the noise, bias, and variance, shown in (c), relative to the polynomial degrees. (f) Percentage breakdown of the bias and variance

a low bias and a high variance. This characterization is also generic and not limited to the particular model we studied.

18.5.3 Interpretation of Error-Complexity Curves

From the idealized error-complexity curves in Fig. 18.3b, one can summarize and clarify a couple of important terms.

We say that **a model is overfitting** if its test error is higher than the one of a *less complex* model. That means, to decide whether a model is overfitting, it is necessary to compare it with a simpler model. Hence, overfitting is detected from a comparison, and it is not an absolute measure. In Fig. 18.3b, all models with a model complexity larger than 3.5 are overfitting with respect to the best model having a model complexity of $c_{opt} = 3.5$ and with the lowest test error. One can formalize this by defining an overfitting model as follows.

Definition 18.3 (Model Overfitting) A model with complexity c is said to be *Overfitting*, if for its test error the following holds:

$$E_{test}(c) - E_{test}(c_{opt}) > 0 \quad \forall c > c_{opt}, \quad (18.42)$$

with

$$c_{opt} = \operatorname{argmin}_c \{E_{test}(c)\}, \quad (18.43)$$

$$E_{test}(c_{opt}) = \min_c \{E_{test}(c)\}. \quad (18.44)$$

From the bias-variance decomposition in Fig. 18.3d, one can see that an overfitting model is characterized by:

$$\textbf{Overfitting} : \text{low bias and high variance} \quad (18.45)$$

Furthermore, from Fig. 18.3b, we can also see that for all these models, the difference between the test error and the training error increases for increasing complexity values, i.e.,

$$\left(E_{test}(c) - E_{train}(c)\right) > \left(E_{test}(c') - E_{train}(c')\right), \quad (18.46)$$

$$\forall c > c' \text{ and } c, c' > c_{opt}.$$

Similarly, we say **a model is underfitting** if its test error is higher than the one of a *more complex* model. That means, to decide whether a model is underfitting, it is necessary to compare it with a more complex model. In Fig. 18.3b, all models

with a model complexity smaller than 3.5 are underfitting with respect to the best model. The formal definition of this can be given as follows.

Definition 18.4 (Model Underfitting) A model with complexity c is said to be *underfitting*, if for its test error the following holds:

$$E_{test}(c) - E_{test}(c_{opt}) > 0, \quad \forall c < c_{opt}. \quad (18.47)$$

From the bias-variance decomposition in Fig. 18.3d, one can see that an underfitting model is characterized by:

$$\textbf{Underfitting} : \text{high bias and low variance} \quad (18.48)$$

Finally, the **generalization capabilities of a model** are assessed by its predictive performance of the test error compared with the training error. If the distance between the test error and the training error is small (small gap), i.e.,

$$E_{test}(c) - E_{train}(c) \approx 0, \quad (18.49)$$

then, the model has good generalization capabilities [3]. From Fig. 18.3b, one can see that models with $c > c_{opt}$ have bad generalization capabilities. In contrast, models with $c < c_{opt}$ have good generalization capabilities but not necessarily a small error. This makes sense considering the fact that the sample size is kept fixed.

In Definition 18.5, we formally summarize these characteristics.

Definition 18.5 (Generalization) If for a model with complexity c , the following holds

$$E_{test}(c) - E_{train}(c) < \delta \text{ with } \delta \in \mathbb{R}^+, \quad (18.50)$$

then, we say that the model has good *generalization* capabilities.

In practice, one needs to decide what is a reasonable value of δ because usually $\delta = 0$ is too strict. This makes the definition of generalization problem specific. Put simply, if one can conclude from the training error to the test error (because they are of similar value) then a model generalizes to new data.

Theoretically, for increasing sample size of the training data, we obtain

$$\lim_{n_{train} \rightarrow \infty} E_{test}(c) - E_{train}(c) = 0, \quad (18.51)$$

for all model complexities c since Eqs. 18.38 and 18.39 become identical for an infinitely large test data set, i.e., $n_{test} \rightarrow \infty$.

From the idealized decomposition of the test error shown in Fig. 18.3d, one can see that a simple model with low variance and high bias has, in general, good generalization capabilities. Whereas for a complex model, its variance is high and its generalization capabilities are poor.

18.6 Learning Curves

The last concept we discuss in this chapter is called *learning curves*. A learning curve shows the performance of a model for different sample sizes of the training data [9, 10], where the performance of a model is measured by its prediction error. To extract the most information, one needs to compare the learning curve of the training error and the test error with each other. This leads to complementary information to the error-complexity curves. Hence, learning curves are playing an important role in model diagnosis but are not strictly considered as part of model assessment methods.

Definition 18.6 *Learning curves* show the training error and test error as functions of the sample size of the training data. The models, underlying these curves, have all the same complexity.

In the following, we first present numerical examples for learning curves for linear polynomial regression models. Then, we discuss the behavior of idealized learning curves that can correspond to any type of statistical model.

18.6.1 Example: Learning Curves for Linear Polynomial Regression Models

In Fig. 18.4, we show results for the linear polynomial regression models discussed earlier. It is important to emphasize that each figure shows results for a fixed model complexity but varying sample size of the training data. This is in contrast with the results shown earlier (see Fig. 18.3), which varied the model complexity but kept the sample size of the training data fixed. We show six examples for six different model degrees. The horizontal dashed red line corresponds to the optimal error, $E_{test}(c_{opt})$, attainable by the model family. The first two examples (Fig. 18.4a and b) are qualitatively different from all the others because neither the training nor the test error converges to $E_{test}(c_{opt})$, but they are much higher. This is due to a high bias of the models, because these models are too simple for the data.

Figure 18.4 E exhibits a different extreme behavior. Here, for sample sizes of the training data smaller than ≈ 60 , we obtain very high test errors and a large difference with the training error. This is due to a high variance of the models because these models are too complex for the data. In contrast, Fig. 18.4c shows results for $c_{opt} = 4$, which are the best results obtainable for this model family and the data.

In general, learning curves can be used to answer the following two questions:

1. How much training data is needed?
2. How much bias and variance is present?

For (1): The learning curves can be used to predict the benefit one obtains by increasing the number of samples in the training data.

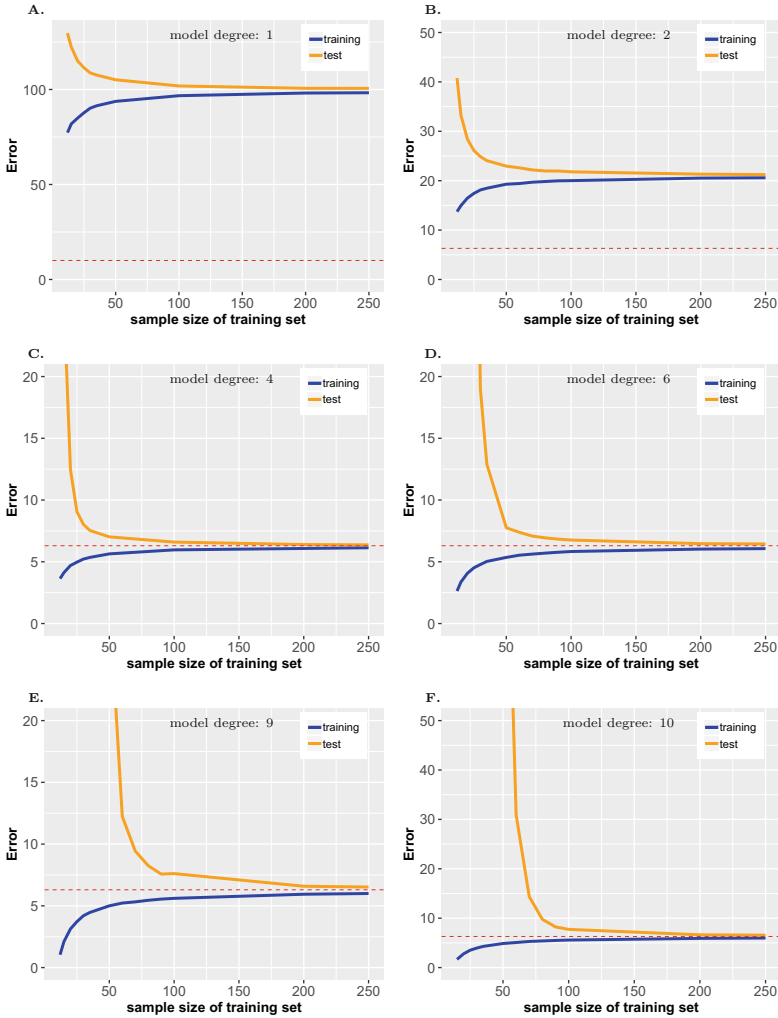


Fig. 18.4 Estimated learning curves for training and test errors for six linear polynomial regression models. The model degree indicates the highest polynomial degree of the fitted model, and the horizontal dashed red line corresponds to the optimal error $E_{test}(c_{opt})$ attainable by the model family for the optimal model complexity $c_{opt} = 4$

- If the curve is slightly changing (increasing for training error and decreasing for test error) → need larger sample size.
- If the curve is completely flattened out → sample size is sufficient.
- If the curve is rapidly changing → need much larger sample size.

This assessment is based on evaluating the tangent of a learning curve toward the highest available sample size.

For (2): In order to study this point, one needs to generate several learning curves for models of different complexity. From this, one obtains information about the smallest attainable test error. In the following, we call this the optimal attainable error $E_{test}(c_{opt})$.

For a specific model, one evaluates its learning curves as follows:

- A model has *high bias* if the training and test error converge to a value much larger than E_{test} . In this case, increasing the sample size of the training data will not improve the results. This indicates an underfitting of the data because the model is too simple. In order to improve the performance, one needs to increase the complexity of the model.
- A model has *high variance* if the training and test error are quite different from each other, i.e., having a large gap between both. Here, the gap is defined as $E_{test}(n) - E_{train}(n)$ for a sample size n of the training data. In this case, the training data are fitted much better than the test data, indicating problems with the generalization capabilities of the model. In order to improve the performance, the sample size of the training data needs to be increased.

These assessments are based on evaluating the gap between the test error and the training error toward the highest available sample size of the training data.

18.6.2 Interpretation of Learning Curves

In Fig. 18.5, we show idealized learning curves for the four cases obtained from combining high/low bias and high/low variance with each other. Specifically, the first/second column shows low/high bias cases, and the first/second row shows low/high variance cases. Figure 18.5a shows the ideal case where the model has a low bias and a low variance. In this case, the training and test error converge both to the optimal attainable error $E_{test}(c_{opt})$ that is shown as a dashed red line.

In Fig. 18.5b, a model with a high bias and a low variance is shown. In this case, the training and test error converge both to values that are distinct from the optimal attainable error, and an increase in the sample size of the training data will not solve this problem. The small gap between the training and test error is indicative of a low variance. A way to improve the performance is to increase the model complexity, e.g., by allowing more free parameters or boosting approaches. This case is the ideal example for an *underfitting model*.

In Fig. 18.5c, a model with a low bias and a high variance is shown. In this case, the training and test error converge both to the optimal attainable error. However, the gap between the training and test error is large, indicating a high variance. In order to reduce this variance, the sample size of the training data needs to be increased to possibly much larger values. The model complexity can also be reduced, e.g., by regularization or bagging approaches. This case is the ideal example for an *overfitting model*.

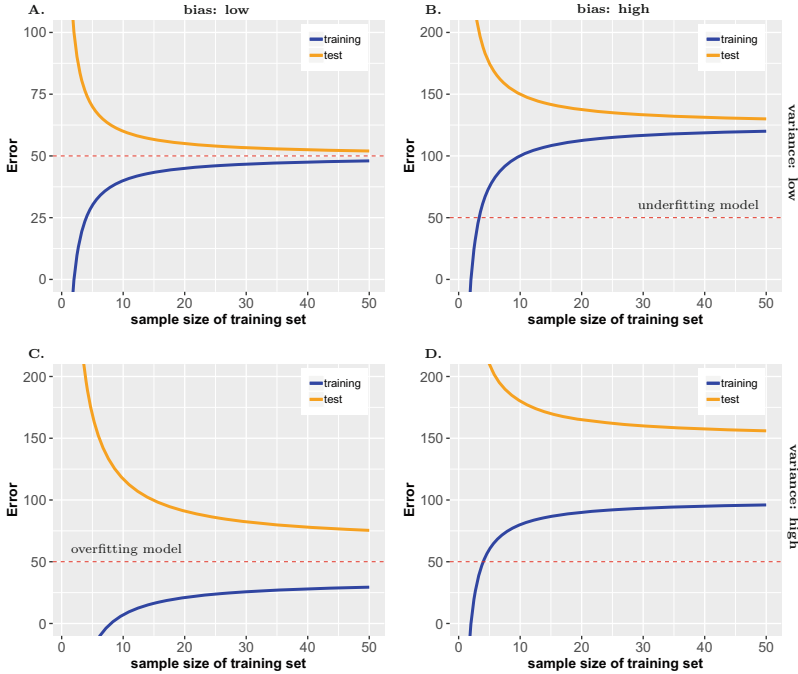


Fig. 18.5 Idealized learning curves. The horizontal dashed red line corresponds to the optimal attainable error $E_{test}(c_{opt})$ by the model family. Shown are the following four cases. (a) Bias, low; variance, low. (b) Bias, high; variance, low. (c) Bias, low; variance, high. (d) Bias, high; variance, high

In Fig. 18.5d, a model with a high bias and a high variance is shown. This is the worst case example. In order to improve the performance, one needs to increase the model complexity and possibly the sample size of the training data. This means that improving such a model is the most demanding case.

The learning curves also allow an evaluation of the generalization capabilities of a model. Only the low variance cases have a small distance between the test error and the training error indicating the model has good generalization capabilities. Hence, a model with a low variance has, in general, good generalization capabilities irrespective of the bias. However, models with a high bias perform bad and their consideration needs to be discussed case-by-case.

18.7 Discussion

In this chapter, we presented the expected generalization error, the bias-variance decomposition, error complexity curves, and learning curves. We discussed the

above concepts theoretical and practically for model assessment, model selection, and model diagnosis [70, 211, 385].

It is important to emphasize that the expected generalization error is a theoretical entity, defined as population estimate. That means, the entire population from which the data are drawn needs to be available. Similarly, not only one training data set is needed but infinitely many. Hence, practically, the expected generalization error is not attainable. Since the bias-variance decomposition is based on the expected generalization error, this is also defined as population estimate. Nevertheless, both concepts are useful to derive insights, e.g., by utilizing simulation studies. In this chapter, we used linear polynomial regression models for such simulations. For practical applications, sample estimates of the above entities need to be obtained, e.g., from resampling methods.

For model assessment, the expected generalization error is the ultimate measure, because it provides the most generic and complete summary information about a statistical model with respect to its prediction capabilities. However, its theoretical nature, as discussed above, requires sample estimates from theoretical data making the concept cumbersome for the beginner, because it is easy to get confused. For this reason, we postponed its discussion until this chapter. Pedagogically, we think that it is better to first gain some practical experience with prediction models and then rethink about their theoretical foundations after a certain degree of understanding has been achieved. As one can see, in this chapter, the expected generalization error is not the end point but the starting one, effecting many other concepts as exemplified by the bias-variance decomposition, error complexity curves, and learning curves. This has also direct connections to model assessment and model selection. Hence, the concept of the expected generalization error triggers an avalanche of other topics, each one nontrivial in its own right.

As mentioned above, error-complexity curves and learning curves can be seen as applications of the expected generalization error and the bias-variance decomposition. While error-complexity curves are based on the bias-variance decomposition to provide a functional dependency on the model complexity, learning curves are based on the expected generalization error to study the dependency on the sample size of the training data. Hence, both concepts provide *dynamical insights* into the estimation capabilities of statistical models.

Interestingly, error-complexity curves can be used to study model selection. In practical terms, model selection is the task of selecting the best statistical model from a family of models for a given data set. In Chap. 12 (see also Chap. 13), we have seen that possible model selection problems include but are not limited to the following:

- Select predictor variables for linear regression models.
- Select among different regularization models, e.g., ridge regression, LASSO, or elastic net.
- Select the best classification method from a list of candidates, e.g., random forest, logistic regression, support vector machine, or neural networks.
- Select the number of neurons and hidden layers in a neural network.

The general problems one tries to counteract with model selection are the overfitting and underfitting of data.

- Underfitting model: Such a model is characterized by high bias, low variance, and a poor test error. In general, such a model is too simple.
- Best model: For such a model, the bias and variance are balanced, and the test error makes good predictions.
- Overfitting model: Such a model is characterized by low bias, high variance, and a poor test error. In general, such a model is too complex.

It is important to realize that the above terms are defined for a given data set with a certain sample size. Specifically, the error-complexity curves are estimated from training data with fixed sample size, and, hence, these curves can change if the sample size changes. In contrast, the learning curves investigate the dependency on the sample size of the training data.

In Chap. 12, we discussed elegant methods for model selection, e.g., the AIC or the BIC; however, the applicability of these methods depends on the availability of analytical results for the models, usually based on their maximum likelihood. Unfortunately, such results can often only be obtained for linear models, as seen in Chap. 12, but may not be available for other types of models. Hence, for practical applications, these methods are far less flexible compared to numerical resampling methods.

The bias-variance trade-off, providing a frequentist view point of model complexity, is for practical problems where the true model is unknown or not accessible. It offers a framework to think about a problem conceptually. Interestingly, the balancing of bias and variance reflects the underlying philosophy of *Ockham's razor* [198], which states that from two similar models the simpler one should be chosen. Importantly, for simulations, the true model is known, and the decomposition into noise, bias, and variance is feasible.

18.8 Summary

In the last chapter of this book, we discussed the expected generalization error as the conceptual roof for many topics discussed in the previous chapters. We have seen that the expected generalization error has a direct effect on model assessment and model selection. Since these topics are also used for a wider purpose, e.g., for selecting the best classification or regression model by means of resampling methods, the expected generalization error penetrates essentially every aspect of data science.

Learning Outcome 18: Expected Generalization Error

The expected generalization error is a theoretical entity, defined as population estimate. That means, the entire population from which the data are drawn needs to be available. Similarly, not only one training data set is needed but infinitely many. Hence, practically, the expected generalization error is not attainable but needs to be estimated via approximations.

For practical applications, the most flexible approach that can be applied to any type of statistical model is a resampling method (for instance, cross-validation). Assuming that the computations can be completed within an acceptable time frame, it is advised to base the decisions for model selection and model assessment on the estimates of the error-complexity curves and the learning curves.

18.9 Exercises

1. Discuss the definition of the expected generalization error.
 - a. What is the meaning of $\mathbb{E}_P$ in this definition?
 - b. What is the meaning of $\mathbb{E}_{D_{\text{train}}}$ in this definition?
2. What is the difference between the in-sample error and the out-of-sample error? What data are used as the test data for these two errors?
3. Derive the bias-variance decomposition of the expected generalization error.
4. What is an error-complexity curve?
5. Estimate error-complexity curves for a polynomial regression model.
 - a. Assume the true polynomial regression model has a degree of 2.
 - b. Assume the true polynomial regression model has a degree of 5.
 - c. Assume the true polynomial regression model has a degree of 7.
 - d. Discuss the differences of these models.
6. Discuss overfitting and underfitting based on an error-complexity curve.
7. What influence have bias and variance on the overfitting and underfitting of a model?
8. What is a learning curve?
9. What data are needed for plotting learning curves?

References

1. O. Aalen, Nonparametric inference for a family of counting processes. *Ann. Stat.* 701–726 (1978)
2. M. Abadi, A. Agarwal, P. Barham et al., Tensorflow: large-scale machine learning on heterogeneous distributed systems (2016)
3. Y.S. Abu-Mostafa, M. Magdon-Ismail, H.-T. Lin, *Learning from Data*, vol. 4. (AMLBook, New York, 2012)
4. K. Aho, D. Derryberry, T. Peterson, Model selection for ecologists: the worldviews of AIC and BIC. *Ecology* **95**(3), 631–636 (2014)
5. H. Akaike, A new look at the statistical model identification, in *Selected Papers of Hirotugu Akaike* (Springer, Berlin, 1974), pp. 215–222
6. B. Alipanahi, A. Delong, M.T. Weirauch, B.J. Frey, Predicting the sequence specificities of DNA and RNA-binding proteins by deep learning. *Nat. Biotechnol.* **33**(8), 831 (2015)
7. G. Altay, F. Emmert-Streib, Structural influence of gene networks on their inference: analysis of C3NET. *Biol. Direct* **6**, 31 (2011)
8. M. Alvi, D. McArt, P. Kelly et al., Comprehensive molecular pathology analysis of small bowel adenocarcinoma reveals novel targets with potential clinical utility. *Oncotarget* **6**(25), 20863–20874 (2015)
9. S.-I. Amari, A universal theorem on learning curves. *Neural Netw.* **6**(2), 161–166 (1993)
10. S.-I. Amari, N. Fujita, S. Shinomoto, Four types of learning curves. *Neural Comput.* **4**(4), 605–618 (1992)
11. V. Amrhein, S. Greenland, B. McShane, Scientists rise up against statistical significance. *Nature* **567**, 3055–307 (2019)
12. A. Ancarani, C. Di Mauro, L. Fratocchi et al., Prior to reshoring: a duration analysis of foreign manufacturing ventures. *Int. J. Prod. Eco.* **169**, 141–155 (2015)
13. S. Arlot, A. Celisse et al., A survey of cross-validation procedures for model selection. *Stat. Surv.* **4**, 40–79 (2010)
14. A.B. Arrieta, N. Díaz-Rodríguez, J. Del Ser et al., Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion* **58**, 82–115 (2020)
15. K. Arulkumaran, M.P. Deisenroth, M. Brundage, A.A. Bharath, Deep reinforcement learning: a brief survey. *IEEE Sig. Proces. Mag.* **34**(6), 26–38 (2017)
16. S.R. Austin, I. Dialsingh, N. Altman, Multiple hypothesis testing: a review. *J. Indian Soc. Agric. Stat.* **68**(2), 303–14 (2014)
17. J. Bacher, *Clusteranalyse* (Oldenbourg Verlag, Munich, 1996)

18. R. Baeza-Yates, B. Ribeiro-Neto (eds.), *Modern Information Retrieval* (Addison-Wesley, Reading, 1999)
19. P. Bühlmann, S. Van De Geer, *Statistics for High-Dimensional Data: Methods, Theory and Applications* (Springer Science & Business Media, Berlin, 2011)
20. P. Baldi, S. Brunak, Y. Chauvin et al., Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics* **16**(5), 412–424 (2000)
21. H.U. Bao-Gang, W. Yong, Evaluation criteria based on mutual information for classifications including rejected class. *Acta Automat. Sin.* **34**(11), 1396–1403 (2008)
22. A.-L. Barabási, Network medicine – From obesity to the “Diseaseome”. *N. Engl. J. Med.* **357**(4), 404–407 (2007)
23. M. Baron, *Probability and Statistics for Computer Scientists*. (Chapman and Hall/CRC, Boca Raton, 2013)
24. H. Barraclough, L. Simms, R. Govindan, Biostatistics primer: what a clinician ought to know: hazard ratios. *J. Thorac. Oncol.* **6**(6), 978–982 (2011)
25. E. Bart, S. Ullman, Cross-generalization: Learning novel classes from a single example by feature replacement, in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1 (IEEE, Piscataway, 2005), pp. 672–679
26. A.M. Bartkowiak, Anomaly, novelty, one-class classification: a comprehensive introduction. *Int. J. Comput. Inf. Syst. Ind. Manag. Appl.* **3**(1), 61–71 (2011)
27. E.M.L. Beale, M.G. Kendall, D.W. Mann, The discarding of variables in multivariate analysis. *Biometrika* **54**(3–4), 357–366 (1967)
28. J. Bekker, J. Davis, Learning from positive and unlabeled data: a survey. *Mach. Learn.* **109**(4), 719–760 (2020)
29. R. Bender, Introduction to the use of regression models in epidemiology, in *Cancer Epidemiology* (Springer, Berlin, 2009), pp. 179–195
30. Y. Bengio et al., Learning deep architectures for AI. *Found. Trends Mach. Learn.* **2**(1), 1–127 (2009)
31. D.J. Benjamin, J.O. Berger, Three recommendations for improving the use of p-values. *Am. Stat.* **73**(sup1), 186–191 (2019)
32. Y. Benjamini, Y. Hochberg, Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J. R. Stat. Soc. B (Methodol.)* **57**, 125–133 (1995)
33. Y. Benjamini, Y. Hochberg, On the adaptive control of the false discovery rate in multiple testing with independent statistics. *J. Educat. Behav. Stat.* **25**(1), 60–83 (2000)
34. Y. Benjamini, D. Yekutieli, The control of the false discovery rate in multiple testing under dependency. *Ann. Stat.* **29**(4), 1165–1188 (2001)
35. Y. Benjamini, A.M. Krieger, D. Yekutieli, Adaptive linear step-up procedures that control the false discovery rate. *Biometrika* **93**(3), 491–507 (2006)
36. C.M. Bennett, G.L. Wolford, M.B. Miller, The principled control of false positives in neuroimaging. *Soc. Cogn. Affect. Neurosci.* **4**(4), 417–422 (2009)
37. C.M. Bennett, A.A. Baird, M.B. Miller, G.L. Wolford, Neural correlates of interspecies perspective taking in the post-mortem atlantic salmon: an argument for proper multiple comparisons correction. *J. Serendipitous Unexpect. Results* **1**, 1–5 (2011)
38. C. Bergmeir, J.M. Benítez, Neural networks in R using the stuttgart neural network simulator: RSNNS. *J. Stat. Softw.* **46**(7), 1–26 (2012)
39. D.J. Biau, B.M. Jolles, R. Porcher, P value and the theory of hypothesis testing: an explanation for new researchers. *Clin. Orthop. Relat. Res.* **468**(3), 885–892 (2010)
40. P.J. Bickel, B. Li, Regularization in statistics. *Test* **15**(2), 271–344 (2006)
41. O. Biran, C. Cotton, Explanation and justification in machine learning: a survey, in *IJCAI-17 Workshop on Explainable AI (XAI)*, vol. 8 (2017), p. 1
42. C.M. Bishop, *Pattern Recognition and Machine Learning* (Springer, Berlin, 2006)
43. G. Blanchard, É. Roquain, Adaptive false discovery rate control under independence and dependence. *J. Mach. Learn. Res.* **10**(Dec), 2837–2871 (2009)

44. G. Blanchard, T. Dickhaus, N. Hack et al., μ toss-multiple hypothesis testing in an open software system, in *Proceedings of the First Workshop on Applications of Pattern Analysis* (2010), pp. 12–19
45. A. Blumer, A. Ehrenfeucht, D. Haussler, M.K. Warmuth, Learnability and the vovk-chervonenkis dimension. *J ACM* **36**(4), 929–965 (1989)
46. H.H. Bock, *Automatische Klassifikation. Theoretische und praktische Methoden zur Gruppierung und Strukturierung von Daten*. Studia Mathematica (Vandenhoeck & Ruprecht, Göttingen, 1974)
47. D. Bonchev, *Information Theoretic Indices for Characterization of Chemical Structures* (Research Studies Press, Chichester, 1983)
48. D. Bonchev, D.H. Rouvray, *Chemical Graph Theory: Introduction and Fundamentals*. Mathematical Chemistry (Abacus Press, London, 1991)
49. Y. Bondarenko, Boltzman-machines (2017)
50. E. Bonferroni, Teoria statistica delle classi e calcolo delle probabilita, in *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze* (1936), pp. 3–62
51. B.E. Boser, I.M. Guyon, V.N. Vapnik, A training algorithm for optimal margin classifiers, in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory* (1992), pp. 144–152
52. L. Bottou, Large-scale machine learning with stochastic gradient descent, in *Proceedings of COMPSTAT'2010* (Springer, Berlin, 2010), pp. 177–186
53. A.P. Bradley, The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recogn.* **30**(7), 1145–1159 (1997)
54. L. Breiman, *Statistics. With a View Toward Applications* (Houghton Mifflin Co., Boston, 1973)
55. L. Breiman, Better subset regression using the nonnegative garrote. *Technometrics* **37**(4), 373–384 (1995)
56. L. Breiman, Bagging predictors. *Mach. Learn.* **24**(2), 123–140 (1996)
57. L. Breiman, J.H. Friedman, R.A. Olshen, Ch.J. Stone, *Classification and Regression Trees*. (Routledge, Milton Park, 1999)
58. L. Breiman et al., Statistical modeling: the two cultures. *Stat. Sci.* **16**(3), 199–231 (2001)
59. N. Breslow, Covariance analysis of censored survival data. *Biometrics*, 89–99 (1974)
60. C. Brunsdon, M. Charlton, An assessment of the effectiveness of multiple hypothesis testing for geographical anomaly detection. *Environ. Plann. B. Plann. Des.* **38**(2), 216–230 (2011)
61. N. Buckley, P. Haddock, R. De Matos Simoes et al., A BRCA1 deficient, NF κ B driven immune signal predicts good outcome in triple negative breast cancer. *Oncotarget* **7**(15), 19884–19896 (2016)
62. K.P. Burnham, D.R. Anderson, Multimodel inference: understanding AIC and BIC in model selection. *Sociol. Methods Res.* **33**(2), 261–304 (2004)
63. C.L. Byrne, The EM algorithm theory: theory, applications and related methods (2017). <https://faculty.uml.edu/cbyrne/AnEMbook.pdf>. Last accessed 28 July 2021
64. A. Candel, V. Parmar, E. LeDell, A. Arora, Deep learning with H2O (2015)
65. E. Candes, T. Tao, The Dantzig selector: statistical estimation when p is much larger than n . *Ann. Stat.* **35**(6), 2313–2351 (2007)
66. C. Cao, F. Liu, H. Tan et al., Deep learning and its applications in biomedicine. *Genomics Proteomics Bioinformatics* **16**(1), 17–32 (2018)
67. F. Capra, *The Web of Life: A New Scientific Understanding of Living Systems* (Anchor, South Harpswell, 1996)
68. M.Á. Carreira-Perpiñán, G. Hinton, On contrastive divergence learning, in *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, PMLR* (2005), pp. 33–40
69. R. Caruana, Multitask learning. *Mach. Learn.* **28**(1), 41–75 (1997)
70. G.C. Cawley, N.L.C. Talbot, On over-fitting in model selection and subsequent selection bias in performance evaluation. *J. Mach. Learn. Res.* **11**(Jul), 2079–2107 (2010)

71. O. Chapelle, B. Schölkopf, A. Zien, *Semi-supervised Learning*. Adaptive Computation and Machine Learning (The MIT Press, Cambridge, 2006)
72. A.S. Charles, B.A. Olshausen, C.J. Rozell, Learning sparse codes for hyperspectral imagery. *IEEE J. Select. Topics Sig. Proces.* **5**(5), 963–978 (2011)
73. T. Chen, M. Li, Y. Li et al., MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems (2015)
74. M.R. Chernick, R.A. LaBudde, *An Introduction to Bootstrap Methods with Applications to R*. (John Wiley & Sons, Hoboken, 2014)
75. Chimera0. pydbm (2019)
76. K. Cho, B. Van Merriënboer, C. Gulcehre et al., Learning phrase representations using RNN encoder-decoder for statistical machine translation (2014). Preprint. arXiv:1406.1078
77. F. Chollet et al., Keras (2015). <https://github.com/fchollet/keras>
78. A. Clare, R.D. King, Knowledge discovery in multi-label phenotype data, in *European Conference on Principles of Data Mining and Knowledge Discovery* (Springer, Berlin, 2001), pp. 42–53
79. B. Clarke, E. Fokoue, H.H. Zhang, *Principles and Theory for Data Mining and Machine Learning* (Springer, Dordrecht, 2009)
80. W.S. Cleveland, Data science: an action plan for expanding the technical areas of the field of statistics. *Int. Stat. Rev.* **69**(1), 21–26 (2001)
81. M. Cleves, W. Gould, W.W. Gould et al., *An Introduction to Survival Analysis Using Stata* (Stata Press, College Station, 2008)
82. J. Cohen, A coefficient of agreement for nominal scales. *Educ. Psychol. Meas.* **20**(1), 37–46 (1960)
83. G. Cohen, S. Afshar, J. Tapson, A. van Schaik, Emnist: an extension of MNIST to handwritten letters (2017). Preprint. arXiv:1702.05373
84. D. Cook, L.B. Holder, *Mining Graph Data* (Wiley-Interscience, Hoboken, 2007)
85. J.M. Cortina, W.P. Dunlap, On the logic and purpose of significance testing. *Psychol. Methods* **2**(2), 161 (1997)
86. D.R. Cox, Regression models and life-tables. *J. R. Stat. Soc. B. Methodol.* **34**(2), 187–202 (1972)
87. D.R. Cox, Partial likelihood. *Biometrika* **62**(2), 269–276 (1975)
88. K. Cranmer, Statistical challenges for searches for new physics at the LHC, in *Statistical Problems in Particle Physics, Astrophysics and Cosmology* (World Scientific, Singapore, 2006), pp. 112–123
89. F. Crick, Central dogma of molecular biology. *Nature* **227**, 561–563 (1970)
90. J. Dai, Y. Wang, X. Qiu et al., BigDL: a distributed deep learning framework for big data (2018)
91. A. Dasgupta, Y.V. Sun, I.R. König et al., Brief review of regression-based and machine learning methods in genetic epidemiology: the genetic analysis workshop 17 experience. *Genet. Epidemiol.* **35**(S1), S5–S11 (2011)
92. M. Dehmer, F. Emmert-Streib, Structural information content of networks: graph entropy based on local vertex functionals. *Comput. Biol. Chem.* **32**, 131–138 (2008)
93. M. Dehmer, F. Emmert-Streib, The structural information content of chemical networks. *Z. Naturforsch., A* **63a**, 155–158 (2008)
94. M. Dehmer, F. Emmert-Streib, *Quantitative Graph Theory. Theory and Applications*. (CRC Press, Boca Raton, 2014)
95. M. Dehmer, F. Emmert-Streib, *Frontiers in Data Science*. Chapman & Hall/CRC, Big Data Series. (Taylor & Francis Group, Milton Park, 2018)
96. M. Dehmer, A. Mowshowitz, A history of graph entropy measures. *Inf. Sci.* **1**, 57–78 (2011)
97. D.M. DeLong, G.H. Guirguis, Y.C. So, Efficient computation of subset selection probabilities with application to Cox regression. *Biometrika* **81**(3), 607–611 (1994)
98. D. DeMers, G. Cottrell, Reducing the dimensionality of data with neural networks, in *Advances in Neural Information Processing Systems*, vol. 5 (1993), pp. 580–587

99. A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm (with discussion). *J. R. Stat. Soc. B* **39**, 1–38 (1977)
100. J. Deng, Z. Zhang, E. Marchi, B. Schuller, Sparse autoencoder-based feature transfer learning for speech emotion recognition, in *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction* (IEEE, Piscataway, 2013), pp. 511–516
101. S. Derksen, H.J. Keselman, Backward, forward and stepwise automated subset selection algorithms: frequency of obtaining authentic and noise variables. *Br. J. Math. Stat. Psychol.* **45**(2), 265–282 (1992)
102. G. Deuschl, C. Schade-Brittinger, P. Krack et al., A randomized trial of deep-brain stimulation for Parkinson's disease. *N. Engl. J. Med.* **355**(9), 896–908 (2006)
103. J. Devillers, A.T. Balaban, *Topological Indices and Related Descriptors in QSAR and QSPR* (Gordon and Breach Science Publishers, Amsterdam, 1999)
104. M.M. Deza, E. Deza, *Encyclopedia of Distances*, 2nd edn. (Springer, Berlin, 2012)
105. R. de Matos Simoes, F. Emmert-Streib, Bagging statistical network inference from large-scale gene expression data. *PLoS ONE* **7**(3), e33624 (2012)
106. R. de Matos Simoes, M. Dehmer, F. Emmert-Streib, Interfacing cellular networks of *S. cerevisiae* and *E. coli*: connecting dynamic and genetic information. *BMC Genom.* **14**, 324 (2013)
107. S. Dieleman, J. Schlüter, C. Raffel et al., Lasagne: first release (2015)
108. J. Ding, V. Tarokh, Y. Yang, Model selection techniques: an overview. *IEEE Sig. Proces. Mag.* **35**(6), 16–34 (2018)
109. M.V. Diudea, I. Gutman, L. Jäntschi, *Molecular Topology* (Nova Publishing, New York, 2001)
110. M. Dixon, D. Klabjan, L. Wei, OSTSC: over sampling for time series classification in R (2017)
111. A.P. Diz, A. Carvajal-Rodríguez, D.O.F. Skibinski, Multiple hypothesis testing in proteomics: a strategy for experimental work. *Mol. Cell. Proteomics* **10**(3), M110.004374 (2011)
112. S. Döhler, Validation of credit default probabilities using multiple-testing procedures. *J. Risk Model Validat.* **4**(4), 59 (2010)
113. S. Döhler, G. Durand, E. Roquain et al., New FDR bounds for discrete and heterogeneous tests. *Electron. J. Stat.* **12**(1), 1867–1900 (2018)
114. A. Dmitrienko, A.C. Tamhane, F. Bretz, *Multiple Testing Problems in Pharmaceutical Statistics*. (CRC Press, Boca Raton, 2009)
115. J. Donahue, Y. Jia, O. Vinyals et al., DeCAF: a deep convolutional activation feature for generic visual recognition, in *Proceedings of the 31st International Conference on Machine Learning, PMLR* (2014), pp. 647–655
116. F. Doshi-Velez, B. Kim, Towards a rigorous science of interpretable machine learning (2017). Preprint. arXiv:1702.08608
117. N.R. Draper, H. Smith, *Applied Regression Analysis*, vol. 326. (John Wiley & Sons, Hoboken, 2014)
118. R.O. Duda, P.E. Hart et al., *Pattern Classification* (John Wiley & Sons, Hoboken, 2000)
119. S. Dudoit, M.J. Van Der Laan, *Multiple Testing Procedures with Applications to Genomics* (Springer Science & Business Media, Berlin, 2007)
120. S. Dudoit, M.J. van der Laan, *Multiple Testing Procedures with Applications to Genomics*. (Springer, New York, 2007)
121. S. Dudoit, J.P. Shaffer, J.C. Boldrick, Multiple hypothesis testing in microarray experiments. *Stat. Sci.* **18**(1), 71–103 (2003)
122. P.K. Dunn, G.K. Smyth, *Generalized Linear Models with Examples in R* (Springer, Berlin, 2018)
123. B. Efron, The efficiency of Cox's likelihood function for censored data. *J. Am. Stat. Assoc.* **72**(359), 557–565 (1977)
124. B. Efron, Nonparametric estimates of standard error: the jackknife, the bootstrap and other methods. *Biometrika* **68**(3), 589–599 (1981)
125. B. Efron, *The Jackknife, the Bootstrap, and Other Resampling Plans*, vol. 38. (SIAM, Philadelphia, 1982)

126. B. Efron, T. Hastie, R. Tibshirani, Discussion: the Dantzig selector: statistical estimation when p is much larger than n . *Ann. Stat.* **35**(6), 2358–2364 (2007)
127. B. Efron, *Large-Scale Inference: Empirical Bayes Methods for Estimation, Testing, and Prediction* (Cambridge University Press, Cambridge, 2010)
128. B. Efron, R.J. Tibshirani, *An Introduction to the Bootstrap* (Chapman and Hall/CRC, New York, 1994)
129. S.A. ElHafeez, C. Torino, G. D'Arrigo et al., An overview on standard statistical methods for assessing exposure-outcome link in survival analysis (part II): the Kaplan-Meier analysis and the Cox regression method. *Aging Clin. Exp. Res.* **24**(3), 203–206 (2012)
130. A. Elisseeff, J. Weston, A kernel method for multi-labelled classification. *Adv. Neural Inform. Proces. Syst.* **14** (2001)
131. N. Elsayed, A.S. Maida, M. Bayoumi, Reduced-gate convolutional LSTM using predictive coding for spatiotemporal prediction (2018). Preprint. arXiv:1810.07251
132. F. Emmert-Streib, A heterosynaptic learning rule for neural networks. *Int. J. Mod. Phys. C* **17**(10), 1501–1520 (2006)
133. F. Emmert-Streib, M. Dehmer, Global information processing in gene networks: fault tolerance, in *Proceedings of the Bio-Inspired Models of Network, Information, and Computing Systems, Bionetics 2007* (2007)
134. F. Emmert-Streib, M. Dehmer, Information processing in the transcriptional regulatory network of yeast: functional robustness. *BMC Syst. Biol.* **3**, 35 (2009)
135. F. Emmert-Streib, M. Dehmer (eds.), *Analysis of Microarray Data: A Network-Based Approach*. (Wiley VCH Publishing, Hoboken, 2010)
136. F. Emmert-Streib, M. Dehmer (eds.), *Medical Biostatistics for Complex Diseases* (Wiley-Blackwell, Weinheim, 2010)
137. F. Emmert-Streib, M. Dehmer, Identifying critical financial networks of the DJIA: towards a network-based index. *Complexity* **16**(1), 24–33 (2010)
138. F. Emmert-Streib, M. Dehmer, A machine learning perspective on personalized medicine: an automatized, comprehensive knowledge base with ontology for pattern recognition. *Mach. Learn. Knowl. Extract.* **1**(1), 149–156 (2018)
139. F. Emmert-Streib, M. Dehmer, High-dimensional lasso-based computational regression models: regularization, shrinkage, and selection. *Mach. Learn. Knowl. Extract.* **1**(1), 359–383 (2019)
140. F. Emmert-Streib, M. Dehmer, Evaluation of regression models: model assessment, model selection and generalization error. *Mach. Learn. Knowl. Extract.* **1**(1), 521–551 (2019)
141. F. Emmert-Streib, M. Dehmer, Understanding statistical hypothesis testing: the logic of statistical inference. *Mach. Learn. Knowl. Extract.* **1**(3), 945–961 (2019)
142. F. Emmert-Streib, M. Dehmer, Defining data science by a data-driven quantification of the community. *Mach. Learn. Knowl. Extract.* **1**(1), 235–251 (2019)
143. F. Emmert-Streib, M. Dehmer, Large-scale simultaneous inference with hypothesis testing: multiple testing procedures in practice. *Mach. Learn. Knowl. Extract.* **1**(2), 653–683 (2019)
144. F. Emmert-Streib, S. Tripathi, R. de Matos Simoes et al., The human disease network: opportunities for classification, diagnosis and prediction of disorders and disease genes. *Syst. Biomed.* **1**(1), 1–8 (2013)
145. F. Emmert-Streib, M. Dehmer, Y. Shi, Fifty years of graph matching, network alignment and network comparison. *Inf. Sci.* 346–347, 180–197 (2016)
146. F. Emmert-Streib, S. Moutari, M. Dehmer, The process of analyzing data is the emergent feature of data science. *Front. Genet.* **7**, 12 (2016)
147. F. Emmert-Streib, S. Tripathi, O. Yli-Harja, M. Dehmer, Understanding the world economy in terms of networks: a survey of data-based network science approaches on economic networks. *Front. Appl. Math. Stat.* **4**, 37 (2018)
148. F. Emmert-Streib, S. Tripathi, M. Dehmer, Constrained covariance matrices with a biologically realistic structure: comparison of methods for generating high-dimensional Gaussian graphical models. *Front. Appl. Math. Stat.* **5**, 17 (2019)

149. F. Emmert-Streib, S. Moutari, M. Dehmer, *Mathematical Foundations of Data Science Using R*. (Walter de Gruyter GmbH & Co KG, Berlin, 2020)
150. F. Emmert-Streib, O. Yli-Harja, M. Dehmer, Explainable artificial intelligence and machine learning: A reality rooted perspective. *WIREs Data Min. Knowl. Discov.* **10**, e1368 (2020)
151. F. Emmert-Streib, Z. Yang, H. Feng et al., An introductory review of deep learning for prediction models with big data. *Frontiers Artificial Intelligence Appl.* **3**, 4 (2020)
152. F. Emmert-Streib, K. Manjang, M. Dehmer et al., Are there limits in explainability of prognostic biomarkers? scrutinizing biological utility of established signatures. *Cancers* **13**(20), 5087 (2021)
153. S. Enarvi, M. Kurimo, Theanolm – an extensible toolkit for neural network language modeling (2016). CoRR, abs/1605.00942
154. B.S. Everitt, S. Landau, M. Leese, D. Stah, *Cluster Analysis*, 5th edn. (Wiley-VCH, Weinheim, 2011)
155. J. Fan, J. Lv, A selective overview of variable selection in high dimensional feature space. *Stat. Sinica* **20**(1), 101 (2010)
156. J. Fan, F. Han, H. Liu, Challenges of big data analysis. *Natl. Sci. Rev.* **1**(2), 293–314 (2014)
157. A. Farcomeni, A review of modern multiple hypothesis testing, with particular attention to the false discovery proportion. *Stat. Methods Med. Res.* **17**(4), 347–88 (2008)
158. T. Fawcett, An introduction to ROC analysis. *Pattern Recogn. Lett.* **27**, 861–874 (2006)
159. L. Fei-Fei, R. Fergus, P. Perona, One-shot learning of object categories. *IEEE Trans. Pattern Anal. Mach. Intell.* **28**(4), 594–611 (2006)
160. L. Fein, The role of the university in computers, data processing, and related fields. *Commun. ACM* **2**(9), 7–14 (1959)
161. J.A. Ferreira, A.H. Zwinderman et al., On the benjamini–hochberg method. *Ann. Stat.* **34**(4), 1827–1849 (2006)
162. A. Fischer, C. Igel, An introduction to restricted Boltzmann machines, in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. CIARP*, ed. by L. Alvarez, M. Mejail, L. Gomez, J. Jacobo. Lecture Notes in Computer Science (Springer, Berlin, 2012)
163. R.A. Fisher, On the mathematical foundations of theoretical statistics. *Philos. Trans. R. Soc. A* **222**, 309–368 (1922)
164. R.A. Fisher, *Statistical Methods for Research Workers* (Genesis Publishing Pvt. Ltd., Delhi, 1925)
165. R.A. Fisher, The statistical method in psychical research, in *Proceedings of the Society for Psychical Research*, vol. 39 (1929), pp. 189–192
166. R.A. Fisher, The arrangement of field experiments (1926), in *Breakthroughs in Statistics* (Springer, Berlin, 1992), pp. 82–91
167. P. Flach, *Machine Learning: The Art of Science and Algorithms That Make Sense of Data*. (Cambridge University Press, New York, 2012)
168. M.R. Forster, Key concepts in model selection: Performance and generalizability. *J. Math. Psychol.* **44**(1), 205–231 (2000)
169. M.R. Forster, Predictive accuracy as an achievable goal of science. *Philos. Sci.* **69**(S3), S124–S134 (2002)
170. A.V. Frane, Are per-family type I error rates relevant in social and behavioral science? *J. Mod. Appl. Stat. Methods* **14**(1), 5 (2015)
171. L.E. Frank, J.H. Friedman, A statistical view of some chemometrics regression tools. *Technometrics* **35**(2), 109–135 (1993)
172. B.R. Frieden, *Science from Fisher Information: A Unification*. (Cambridge University Press, Cambridge, 2004)
173. J. Friedman, T. Hastie, R. Tibshirani, glmnet: Lasso and elastic-net regularized generalized linear models. *R Packag. Ver.* **1**(4) (2009)
174. J. Friedman, T. Hastie, R. Tibshirani, Regularization paths for generalized linear models via coordinate descent. *J. Stat. Softw.* **33**(1), 1 (2010)
175. J. Fürnkranz, E. Hüllermeier, E. Loza Mencía, K. Brinker, Multilabel classification via calibrated label ranking. *Mach. Learn.* **73**(2), 133–153 (2008)

176. A. Gammerman, V. Vovk, V. Vapnik, Learning by transduction, in *UAI'98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence* (1998), pp. 148–155
177. Y.C. Ge, S. Dudoit, T.P. Speed, Resampling-based multiple testing for microarray data analysis. *Test* **12**(1), 1–77 (2003)
178. S. Geisser, The predictive sample reuse method with applications. *J. Am. Stat. Assoc.* **70**(350), 320–328 (1975)
179. S. Geman, E. Bienenstock, R. Doursat, Neural networks and the bias/variance dilemma. *Neural Comput.* **4**(1), 1–58 (1992)
180. C. Genovese, L. Wasserman, Operating characteristics and extensions of the false discovery rate procedure. *J. R. Stat. Soc. Series B Stat. Methodol.* **64**(3), 499–517 (2002)
181. C.R. Genovese, L. Wasserman, Exceedance control of the false discovery proportion. *J. Am. Stat. Assoc.* **101**(476), 1408–1417 (2006)
182. C.R. Genovese, K. Roeder, L. Wasserman, False discovery control with p-value weighting. *Biometrika* **93**(3), 509–524 (2006)
183. A. Genz, F. Bretz, *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics (Springer, Heidelberg, 2009)
184. A. Genz, F. Bretz, T. Miwa et al., mvtnorm: multivariate normal and t distributions (2019). R package version 1.0-9
185. F.A. Gers, J. Schmidhuber, Recurrent nets that time and count, in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 3 (IEEE, Piscataway, 2000), pp. 189–194
186. F.A. Gers, J. Schmidhuber, F. Cummins, Learning to forget: continual prediction with LSTM (1999)
187. F.A. Gers, N.N. Schraudolph, J. Schmidhuber, Learning precise timing with LSTM recurrent networks. *J. Mach. Learn. Res.* **3**(Aug), 115–143 (2002)
188. P. Geurts, Bias vs variance decomposition for regression and classification, in *Data Mining and Knowledge Discovery Handbook* (Springer, Berlin, 2009), pp. 733–746
189. N. Ghamrawi, A. McCallum, Collective multi-label classification, in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management* (2005), pp. 195–200
190. E. Gibaja, S. Ventura, Multi-label learning: a review of the state of the art and ongoing research. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **4**(6), 411–444 (2014)
191. G. Gigerenzer, The superego, the ego, and the id in statistical reasoning, in *A Handbook for Data Analysis in the Behavioral Sciences: Methodological Issues* (1993), pp. 311–339
192. S.G. Gilmour, The interpretation of Mallows's c_p -statistic. *Statistician*, 49–56 (1996)
193. M.K. Goel, P. Khanna, J. Kishore, Understanding survival analysis: Kaplan-Meier estimate. *Int. J. Ayurveda Res.* **1**(4), 274 (2010)
194. J.J. Goeman, A. Solari, The sequential rejection principle of familywise error control. *Ann. Stat.* 3782–3810 (2010)
195. J.J. Goeman, A. Solari, Multiple hypothesis testing in genomics. *Stat. Med.* **33**(11), 1946–1978 (2014)
196. K.-I. Goh, M.E. Cusick, D. Valle et al., The human disease network. *Proc. Natl. Acad. Sci.* **104**(21), 8685–8690 (2007)
197. E.M. Gold, Language identification in the limit. *Inf. Contr.* **10**(5), 447–474 (1967)
198. I.J. Good, Explicativity: a mathematical theory of explanation with statistical applications. *Proc. R. Soc. Lond. A* **354**(1678), 303–330 (1977)
199. P.I. Good, *Resampling Methods* (Springer, Berlin, 2006)
200. I.J. Goodfellow, D. Warde-Farley, P. Lamblin et al., Pylearn2: a machine learning research library (2013)
201. I. Goodfellow, J. Pouget-Abadie, M. Mirza et al., Generative adversarial nets, in *Advances in Neural Information Processing Systems* (2014), pp. 2672–2680
202. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (The MIT Press, Cambridge, 2016)

203. S. Goodman, A dirty dozen: twelve p-value misconceptions, in *Seminars in Hematology*, vol. 45 (Elsevier, Amsterdam, 2008), pp. 135–140
204. R.A. Gordon, *Regression Analysis for the Social Sciences* (Routledge, Milton Park, 2015)
205. A. Gordon, G. Glazko, X. Qiu et al., Control of the mean number of false discoveries, Bonferroni and stability of multiple testing. *Ann. Appl. Stat.* **1**(1), 179–190 (2007)
206. A. Graves, Generating sequences with recurrent neural networks (2013). Preprint. arXiv:1308.0850
207. A. Graves, J. Schmidhuber, Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Netw.* **18**(5–6), 602–610 (2005)
208. A. Graves, A. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, Piscataway, 2013), pp. 6645–6649
209. S. Greenland, S.J. Senn, K.J. Rothman et al., Statistical tests, p values, confidence intervals, and power: a guide to misinterpretations. *Eur. J. Epidemiol.* **31**(4), 337–350 (2016)
210. S.R. Gross, B. O'Brien, C. Hu, E.H. Kennedy, Rate of false conviction of criminal defendants who are sentenced to death. *Proc. Natl. Acad. Sci.* **111**(20), 7230–7235 (2014)
211. I. Guyon, A. Saffari, G. Dror, G. Cawley, Model selection: beyond the bayesian/frequentist divide. *J. Mach. Learn. Res.* **11**(Jan), 61–87 (2010)
212. I. Hacking, *Logic of Statistical Inference* (Cambridge University Press, Cambridge, 2016)
213. M. Halkidi, Y. Batistakis, M. Vazirgiannis, On clustering validation techniques. *J. Intel. Inf. Syst.* **17**, 107–145 (2001)
214. J. Han, M. Kamber, *Data Mining: Concepts and Techniques* (Morgan and Kaufmann Publishers, Burlington, 2001)
215. F. Harary, *Graph Theory* (Addison Wesley Publishing Company, Reading, 1969)
216. J. Hardin, R. Hoerl, N.J. Horton et al., Data science in statistics curricula: Preparing students to ‘think with data’. *Am. Stat.* **69**(4), 343–353 (2015)
217. F.E. Harrell, *Regression Modeling Strategies* (Springer, New York, 2001)
218. F.E. Harrell, K.L. Lee, Verifying assumptions of the Cox proportional hazards model, in *Proceedings of the Eleventh Annual SAS Users Group International Conference* (SAS Institute Inc., Cary, 1986), pp. 823–828
219. C.R. Harvey, Y. Liu, Evaluating trading strategies. *J. Portf. Manag.* **40**(5), 108–118 (2014)
220. T. Hastie, R. Tibshirani, J.H. Friedman, *The Elements of Statistical Learning*. (Springer, Berlin, 2001)
221. T.J. Hastie, R.J. Tibshirani, J.H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics (Springer, New York, 2009)
222. T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction* (Springer, New York, 2009)
223. T. Hastie, R. Tibshirani, M. Wainwright, *Statistical Learning with Sparsity: The Lasso and Generalizations* (CRC Press, Boca Raton, 2015)
224. C. Hayashi, What is data science? Fundamental concepts and a heuristic example, in *Data Science, Classification, and Related Methods* (Springer, Berlin, 1998), pp. 40–51
225. H.O. Hayter, *Probability and Statistics for Engineers and Scientists*, 4th edn. (Duxbury Press, Belmont, 2012)
226. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778
227. D.O. Hebb, *The Organization of Behavior* (Wiley, New York, 1949)
228. D. Helbing, The automation of society is next: How to survive the digital revolution (2015). Available at SSRN 2694312
229. M. Henaff, J. Bruna, Y. LeCun, Deep convolutional networks on graph-structured data (2015). Preprint. arXiv:1506.05163
230. P. Henderson, R. Islam, P. Bachman et al., Deep reinforcement learning that matters, in *Thirty-Second AAAI Conference on Artificial Intelligence* (2018)
231. J. Hertz, A. Krogh, R.G. Palmer, *Introduction to the Theory of Neural Computation*. (Addison-Wesley, Boston, 1991)

232. G.E. Hinton, A practical guide to training restricted Boltzmann machines, in *Neural Networks: Tricks of the Trade* (Springer, Berlin, 2012), pp. 599–619
233. G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks. *Science* **313**, 504–507 (2006)
234. G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks. *Science* **313**(5786), 504–507 (2006)
235. G.E. Hinton, T.J. Sejnowski, Optimal perceptual inference, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Citeseer, 1983), pp. 448–453
236. G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets. *Neural Comput.* **18**(7), 1527–1554 (2006)
237. D.C. Hoaglin, F. Mosteller, J.W. Tukey, *Understanding Robust and Exploratory Data Analysis* (Wiley, New York, 1983)
238. Y. Hochberg, A sharper Bonferroni procedure for multiple tests of significance. *Biometrika* **75**(4), 800–802 (1988)
239. J. Hochberg, A. Tamhane, *Multiple Comparison Procedures* (John Wiley & Sons, New York, 1987)
240. S. Hochreiter, The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertainty Fuzziness Knowledge Based Syst.* **6**(02), 107–116 (1998)
241. S. Hochreiter, J. Schmidhuber, Long short-term memory. *Neural Comput.* **9**(8), 1735–1780 (1997)
242. W. Hoeffding, Probability inequalities for sums of bounded random variables. *J. Am. Stat. Assoc.* **58**(301), 13–30 (1963)
243. A.E. Hoerl, R.W. Kennard, Ridge regression: biased estimation for nonorthogonal problems. *Technometrics* **12**(1), 55–67 (1970)
244. P. Hogeweg, B. Hesper, Interactive instruction on population interactions. *Comput. Biol. Med.* **8**(4), 319–327 (1978)
245. S. Holm, A simple sequentially rejective multiple test procedure. *Scandinavian J. Stat.*, 65–70 (1979)
246. A. Holzinger, C. Biemann, C.S. Pattichis, D.B. Kell, What do we need to build explainable ai systems for the medical domain? (2017). Preprint. arXiv:1712.09923
247. G. Hommel, A stagewise rejective multiple test procedure based on a modified Bonferroni test. *Biometrika* **75**(2), 383–386 (1988)
248. J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA* **79**, 2554–2558 (1982)
249. K. Hornik, Approximation capabilities of multilayer feedforward networks. *Neural Netw.* **4**(2), 251–257 (1991)
250. H. Hotelling, Analysis of a complex of statistical variables into principal components. *J. Educ. Psychol.* **24**, 417–441 (1933)
251. M. Hou, B. Chaib-Draa, C. Li, Q. Zhao, Generative adversarial positive-unlabelled learning (2017). Preprint. arXiv:1711.08054
252. J. Howard et al., fastai (2018). <https://github.com/fastai/fastai>
253. B.G. Hu, Y. Wang, Evaluation criteria based on mutual information for classifications including rejected class. *Acta Automat. Sin.* **34**(11), 1396–1403 (2008)
254. R. Hubbard, R.A. Parsa, M.R. Luthy, The spread of statistical significance testing in psychology: the case of the journal of applied psychology, 1917–1994. *Theory Psychol.* **7**(4), 545–554 (1997)
255. W. Huber, V. Carey, L. Long, S. Falcon, R. Gentleman, Graphs in molecular biology. *BMC Bioinf.* **8**(Suppl 6), S8 (2007)
256. J.D. Huling, P.Z.G. Qian, Fast penalized regression and cross validation for tall data with the oem package. *J. Stat. Softw.* (2018). To appear
257. K. Hwang, W. Sung, Single stream parallelization of generalized LSTM-like RNNs on a GPU, in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, Piscataway, 2015), pp. 1047–1051

258. C. Igel, M. Hüsken, Improving the RPROP learning algorithm, in *Proceedings of the Second International ICSC Symposium on Neural Computation (NC 2000)* (2000), pp. 115–121
259. J.P.A. Ioannidis, Retiring significance: a free pass to bias. *Nature* **567**(7749), 461–461 (2019)
260. A.K. Jain, R.C. Dubes, *Algorithms for Clustering Data* (Prentice-Hall Inc., Upper Saddle River, 1988)
261. N. Japkowicz, Concept-learning in the absence of counter-examples: an autoassociation-based approach to classification. Ph.D. Thesis. State University of New Jersey (1999)
262. K. Jaskie, A. Spanias, Positive and unlabeled learning algorithms and applications: a survey, in *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)* (IEEE, Piscataway, 2019), pp. 1–8
263. E.T. Jaynes, *Probability Theory: The Logic of Science* (Cambridge University Press, Cambridge, 2003)
264. Y. Jia, E. Shelhamer, J. Donahue et al., Caffe: convolutional architecture for fast feature embedding, in *Proceedings of the 22Nd ACM International Conference on Multimedia, MM '14* (ACM, New York, 2014), pp. 675–678
265. I.M. Johnstone, D.M. Titterington, Statistical challenges of high-dimensional data. *Philos. Transact. A Math. Phys. Eng. Sci.* **367**(1906), 4237 (2009)
266. I.T. Jolliffe, *Principal Component Analysis* (Springer Science & Business Media, Berlin, 2002)
267. M.I. Jordan, *Learning in Graphical Models* (MIT Press, Cambridge, 1998)
268. E.-Y. Jung, C. Baek, J.-D. Lee, Product survival analysis for the app store. *Market. Lett.* **23**(4), 929–941 (2012)
269. S. Kadam, V. Vaidya, Review and analysis of zero, one and few shot learning approaches, in *International Conference on Intelligent Systems Design and Applications* (Springer, Berlin, 2018), pp. 100–112
270. J.D. Kalbfleisch, R.L. Prentice, *The Statistical Analysis of Failure Time Data*, vol. 360. (John Wiley & Sons, Hoboken, 2011)
271. E.L. Kaplan, P. Meier, Nonparametric estimation from incomplete observations. *J. Am. Stat. Assoc.* **53**(282), 457–481 (1958)
272. R.E. Kass, A.E. Raftery, Bayes factors. *J. Am. Stat. Assoc.* **90**(430), 773–795 (1995)
273. A. Kassambara, M. Kosinski, P. Biecek et al., survminer: drawing survival curves using ‘ggplot2’ (2017). R package version 0.3
274. R.L. Kaufman, *Heteroskedasticity in Regression: Detection and Correction*, vol. 172 (Sage Publications, Thousand Oaks, 2013)
275. L. Kaufman, P.J. Rousseeuw, *Clustering by Means of Medoids* (North Holland/Elsevier, Amsterdam, 1987), pp. 405–416
276. V. Kaushik, C.A. Walsh, Pragmatism as a research paradigm and its implications for social work research. *Soc. Sci.* **8**(9), 255 (2019)
277. S.S. Khan, M.G. Madden, One-class classification: taxonomy of study and review of techniques. *Knowl. Eng. Rev.* **29**(3), 345–374 (2014)
278. J.-H. Kim, Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Comput. Stat. Data Anal.* **53**(11), 3735–3745 (2009)
279. Y. Kim, Convolutional neural networks for sentence classification (2014). Preprint. arXiv:1408.5882
280. D.G. Kleinbaum, M. Klein, *Survival Analysis: A Self-Learning Text*. Statistics for Biology and Health (Springer, New York, 2005)
281. D.G. Kleinbaum, L.L. Kupper, *Applied Regression Analysis and Other Multivariable Methods*. (Duxbury Press, London, 1978)
282. G. Koch, R. Zemel, R. Salakhutdinov, Siamese neural networks for one-shot image recognition, in *ICML Deep Learning Workshop*, Lille, vol. 2 (2015)
283. R. Kohavi, D.H. Wolpert et al., Bias plus variance decomposition for zero-one loss functions, in *International Conference on Machine Learning*, vol. 96 (1996), pp. 275–83

284. R. Kohavi et al., A study of cross-validation and bootstrap for accuracy estimation and model selection, in *International Joint Conference on Artificial Intelligence*, Montreal, vol. 14 (1995), pp. 1137–1145
285. D. Koller, N. Friedman, *Probabilistic Graphical Models: Principles and Techniques* (The MIT Press, Cambridge, 2009)
286. I. Koo, S. Yao, X. Zhang, S. Kim, Comparative analysis of false discovery rate methods in constructing metabolic association networks. *J. Bioinform. Comput. Biol.* **12**, 1450018 (2014)
287. Q. Kou, Y. Sugomori, RcppDL (2014)
288. G. Kraemer, M. Reichstein, M.D. Mahecha, dimRed and coRanking—unifying dimensionality reduction in R. *R J.* **10**(1), 342–358 (2018). coRanking version 0.2.3
289. A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105
290. D. Krstajic, L.J. Buturovic, D.E. Leahy, S. Thomas, Cross-validation pitfalls when selecting and assessing regression and classification models. *J. Cheminform.* **6**(1), 10 (2014)
291. K.G. Kugler, L.A.J. Müller, A. Graber, M. Dehmer, Integrative network biology: Graph prototyping for co-expression cancer networks. *PLoS ONE* **6**, e22843 (2011)
292. J. Kuha, AIC and BIC: comparisons of assumptions and performance. *Sociol. Methods Res.* **33**(2), 188–229 (2004)
293. T.S. Kuhn, *The Structure of Scientific Revolutions* (University of Chicago Press, Chicago, 1970)
294. S. Lafon, A.B. Lee, Diffusion maps and coarse-graining: a unified framework for dimensionality reduction, graph partitioning, and data set parameterization. *IEEE Trans. Pattern Anal. Mach. Intell.* **28**(9), 1393–1403 (2006)
295. M. Lavine, M.J. Schervish, Bayes factors: what they are and what they are not. *Am. Stat.* **53**(2), 119–122 (1999)
296. S. Lawrence, C. Giles, A. Tsoi, A. Back, Face recognition: a convolutional neural network approach. *IEEE Trans. Neural Netw.* **8**, 98–113 (1997)
297. Y. Lecun, Generalization and network design strategies, in *Connectionism in Perspective*, ed. by R. Pfeifer, Z. Schreter, F. Fogelman, L. Steels (Elsevier, Amsterdam, 1989)
298. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. *Nature* **521**(7553), 436 (2015)
299. D.D. Lee, H.S. Seung, Algorithms for non-negative matrix factorization. *Adv. Neural Inf. Process. Syst.* **13**, 556–562 (2001)
300. E.T. Lee, J. Wang, *Statistical Methods for Survival Data Analysis*, vol. 476 (John Wiley & Sons, Hoboken, 2003)
301. H. Lee, P. Pham, Y. Largman, A.Y. Ng, Unsupervised feature learning for audio classification using convolutional deep belief networks, in *Advances in Neural Information Processing Systems* (2009), pp. 1096–1104
302. E.L. Lehmann, The Fisher, Neyman-Pearson theories of testing hypotheses: one theory or two? *J. Am. Stat. Assoc.* **88**(424), 1242–1249 (1993)
303. M.D. Lesem, T.K. Tran-Johnson, R.A. Riesenberger et al., Rapid acute treatment of agitation in individuals with schizophrenia: multicentre, randomised, placebo-controlled study of inhaled loxapine. *Br. J. Psychiatry* **198**(1), 51–58 (2011)
304. K.-M. Leung, R.M. Elashoff, A.A. Afifi, Censoring issues in survival analysis. *Ann. Rev. Pub. Health* **18**(1), 83–104 (1997)
305. M.K.K. Leung, H.Y. Xiong, L.J. Lee, B.J. Frey, Deep learning of the tissue-regulated splicing code. *Bioinformatics* **30**(12), i121–i129 (2014)
306. D. Li, T.D. Dye, Power and stability properties of resampling-based multiple testing procedures with applications to gene oncology studies. *Comput. Math. Methods Med.* (2013)
307. X. Li, T. Zhao, X. Yuan, H. Liu, The flare package for high dimensional linear regression and precision matrix estimation in R. *J. Mach. Learn. Res.* **16**(1), 553–557 (2015)
308. R. Li, S. Wang, F. Zhu, J. Huang, Adaptive graph convolutional neural networks, in *Thirty-Second AAAI Conference on Artificial Intelligence* (2018)

309. K. Liang, D. Nettleton, Adaptive and dynamic adaptive procedures for false discovery rate control and estimation. *J. R. Stat. Soc. Series B Stat. Methodol.* **74**(1), 163–182 (2012)
310. C. Liedtke, C. Mazouni, K.R. Hess et al., Response to neoadjuvant therapy and long-term survival in patients with triple-negative breast cancer. *J. Clin. Oncol.* **26**(8), 1275–1281 (2008)
311. M. Lin, Q. Chen, S. Yan, Network in network (2013). Preprint. arXiv:1312.4400
312. Z.C. Lipton, J. Berkowitz, C. Elkan, A critical review of recurrent neural networks for sequence learning (2015). Preprint. arXiv:1506.00019
313. W. Liu, J. Wang, S.-F. Chang, Robust and scalable graph-based semisupervised learning. *Proc. IEEE* **100**(9), 2624–2638 (2012)
314. W.-Y. Loh, Fifty years of classification and regression trees. *Int. Stat. Rev.* **82**(3), 329–348 (2014)
315. J.S. Long, The origins of sex differences in science. *Soc. Forces* **68**, 1297–1315 (1990)
316. M. Loukides, *What Is Data Science?* (O'Reilly Media, Sebastopol, 2011)
317. Z. Lu, H. Pu, F. Wang, Z. Hu, L. Wang, The expressive power of neural networks: a view from the width, in *Advances in Neural Information Processing Systems* (2017), pp. 6231–6239
318. S.M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in *Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017), pp. 4768–4777
319. J. Li, S. Ma, *Survival Analysis in Medicine and Genetics* (Chapman and Hall/CRC, Boca Raton, 2013)
320. J.B. MacQueen, Some methods for classification and analysis of multivariate observations, in *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability* (University of California Press, Berkeley, 1967), pp. 281–297
321. L.M. Manevitz, M. Yousef, Document classification on neural networks using only positive examples, in *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2000), pp. 304–306
322. K. Manjang, S. Tripathi, O. Yli-Harja et al., Prognostic gene expression signatures of breast cancer are lacking a sensible biological meaning. *Sci. Rep.* **11**(1), 1–18 (2021)
323. R.N. Mantegna, Hierarchical structure in financial markets. *Euro. Phys. J. B* **11**(1), 193–197 (1999)
324. N. Mantel, Evaluation of survival data and two new rank order statistics arising in its consideration. *Cancer Chemother. Rep.* **50**, 163–170 (1966)
325. A. Marshall, *Principles of Economics* (Macmillan, London, 1890)
326. B.W. Matthews, Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochim. Biophys. Acta Protein Struct. Mol. Enzymol.* **405**(2), 442–451 (1975)
327. A.G. McKendrick, Applications of mathematics to medical problems. *Proc. Edinb. Math. Soc.* **44**, 98–130 (1926)
328. G.J. McLachlan, T. Krishnan, *The EM Algorithm and Extensions*, 2nd edn. (Wiley, New York, 2008)
329. R.J. Meijer, T.J.P. Krebs, J.J. Goeman, Hommel's procedure in linear time. *Biom. J.* **61**(1), 73–82 (2019)
330. N. Meinshausen, B. Yu et al., Lasso-type recovery of sparse representations for high-dimensional data. *Ann. Stat.* **37**(1), 246–270 (2009)
331. N. Meinshausen, M.H. Maathuis, P. Bühlmann et al., Asymptotic optimality of the Westfall–Young permutation procedure for multiple testing under dependence. *Ann. Stat.* **39**(6), 3369–3391 (2011)
332. T. Mikolov, I. Sutskever, K. Chen et al., Distributed representations of words and phrases and their compositionality, in *Advances in Neural Information Processing Systems* (2013), pp. 3111–3119
333. C.J. Miller, C. Genovese, R.C. Nichol et al., Controlling the false-discovery rate in astrophysical data analysis. *Astron. J.* **122**(6), 3492 (2001)
334. Y. Ming, Sh. Cao, R. Zhang et al., Understanding hidden memories of recurrent neural networks, in *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)* (IEEE, Piscataway, 2017), pp. 13–24

335. T.M. Mitchell, The need for biases in learning generalizations, in *Readings in Machine Learning* ed. by J.W. Shavlik, T.G. Dietterich (Morgan Kaufman, Burlington, 1980), pp. 184–191
336. T. Mitchell, *Machine Learning* (McGraw-Hill, New York, 1997)
337. V. Mnih, K. Kavukcuoglu, D. Silver et al., Human-level control through deep reinforcement learning. *Nature* **518**(7540), 529 (2015)
338. A. Mohamed, G.E. Dahl, G. Hinton, Acoustic modeling using deep belief networks. *IEEE Trans. Audio Speech Lang. Proces.* **20**(1), 14–22 (2011)
339. M. Mohri, A. Rostamizadeh, A. Talwalkar, *Foundations of Machine Learning*. (MIT Press, Cambridge, 2018)
340. I. Molina, J.G.I. Prat, F. Salvador, B. Treviño, E. Sulleiro, N. Serre, D. Pou, S. Roure, J. Cabezas, L. Valerio et al., Randomized trial of posaconazole and benznidazole for chronic chagas' disease. *N. Engl. J. Med.* **370**(20), 1899–1908 (2014)
341. A.M. Molinaro, R. Simon, R.M. Pfeiffer, Prediction error estimation: a comparison of resampling methods. *Bioinformatics* **21**(15), 3301–3307 (2005)
342. F. Mordelet, J.-P. Vert, A bagging SVM to learn from positive and unlabeled examples. *Pattern Recogn. Lett.* **37**, 201–209 (2014)
343. R.D. Morey, J.-W. Romeijn, J.N. Rouder, The philosophy of Bayes factors and the quantification of statistical evidence. *J. Math. Psychol.* **72**, 6–18 (2016)
344. V. Moskvina, K.M. Schmidt, On multiple-testing correction in genome-wide association studies. *Genet. Epidemiol. Off. Publ. Int. Genet. Epidemiol. Soc.* **32**(6), 567–573 (2008)
345. A. Mowshowitz, Entropy and the complexity of the graphs I: an index of the relative complexity of a graph. *Bull. Math. Biophys.* **30**, 175–204 (1968)
346. M.M. Moya, D.R. Hush, Network constraints and multi-objective optimization for one-class classification. *Neural Netw.* **9**(3), 463–474 (1996)
347. L. Mueller, K. Kugler, A. Graber et al., Structural measures for network biology using QuACN. *BMC Bioinf.* **12**(1), 492 (2011)
348. L.A.J. Müller, M. Schutte, K.G. Kugler, M. Dehmer, QuACN: Quantitative Analyze of Complex Networks (2012). R Package Version 1.6.
349. L.A.J. Müller, M. Dehmer, F. Emmert-Streib, Network-based methods for computational diagnostics by means of R, in *Computational Medicine* (Springer, Berlin, 2012), pp. 185–197
350. D.J. Murdoch, Y.-L. Tsai, J. Adcock, P-values are random variables. *Am. Stat.* **62**(3), 242–245 (2008)
351. D.W. Murray, A.J. Carr, C. Bulstrode, Survival analysis of joint replacements. *J. Bone Joint Surg. Br.* **75**(5), 697–704 (1993)
352. V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (2010), pp. 807–814
353. P. Naur, Concise survey of computer methods (1974)
354. A.A. Neath, J.E. Cavanaugh, The bayesian information criterion: background, derivation, and applications. *Wiley Interdiscip. Rev. Comput. Stat.* **4**(2), 199–203 (2012)
355. W. Nelson, Theory and applications of hazard plotting for censored failure data. *Technometrics* **14**(4), 945–966 (1972)
356. S. Newcomb, A generalized theory of the combination of observations so as to obtain the best result. *Am. J. Math.* **8**, 343–366 (1886)
357. M.E.J. Newman, Modularity and community structure in networks. *Proc. Natl. Acad. Sci. USA* **103**, 8577–8582 (2006)
358. J. Neyman, Sur un teorema concernente le cosidette statistiche sufficienti. *Giorn. Ist. Ital. Att.* **6**, 320–334 (1935)
359. J. Neyman, E.S. Pearson, On the use and interpretation of certain test criteria for purposes of statistical inference: part I. *Biometrika*, 175–240 (1928)
360. J. Neyman, E.S. Pearson, On the problem of the most efficient tests of statistical hypotheses. *Philos. Trans. R. Soc. Lond. A* **231**, 289–337 (1933)

361. A. Nichols, Causal inference with observational data. *Stata J.* **7**(4), 507–541 (2007)
362. T. Nichols, S. Hayasaka, Controlling the familywise error rate in functional neuroimaging: a comparative review. *Stat. Methods Med. Res.* **12**(5), 419–446 (2003)
363. A.M. Nicholson, Generalization error estimates and training data valuation. Ph.D. Thesis, California Institute of Technology (2002)
364. R.S. Nickerson, Null hypothesis significance testing: a review of an old and continuing controversy. *Psychol. Methods* **5**(2), 241 (2000)
365. M.A. Nielsen, *Neural Networks and Deep Learning* (Determination Press, 2015)
366. G. Niu, M.C. du Plessis, T. Sakai et al., Theoretical comparisons of positive-unlabeled learning against positive-negative learning, in *Advances in Neural Information Processing Systems* (2016), pp. 1199–1207
367. T.W. Nix, J.J. Barnette, The data analysis dilemma: ban or abandon. A review of null hypothesis significance testing. *Res. Sch.* **5**(2), 3–14 (1998)
368. W.S. Noble, How does multiple testing correction work? *Nat. Biotechnol.* **27**(12), 1135 (2009)
369. B.A. Olshausen, D.J. Field, Sparse coding with an overcomplete basis set: a strategy employed by v1? *Vis. Res.* **37**(23), 3311–3325 (1997)
370. Online Mendelian Inheritance in Man, OMIM (TM) (2007)
371. J. Oyelade, I. Isewon, F. Oladipupo et al., Clustering algorithms: their application to gene expression data. *Bioinf. Biol. Insights* **10**, 237–253 (2016)
372. S.J. Pan, Q. Yang, A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* **22**(10), 1345–1359 (2009)
373. O.A. Panagiotou, J.P.A. Ioannidis, Genome-Wide Significance Project. What should the genome-wide significance threshold be? Empirical replication of borderline genetic associations. *Int. J. Epidemiol.* **41**(1), 273–286 (2011)
374. A. Paszke, S. Gross, S. Chintala et al., Automatic differentiation in pytorch (2017)
375. A.B. Patel, T. Nguyen, R.G. Baraniuk, A probabilistic framework for deep learning, in *NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems* (2016), pp. 2558–2566
376. T.H.D.J. Patil, T.H. Davenport, Data scientist: the sexiest job of the 21st century. *Harv. Bus. Rev.* (2012)
377. M.Q. Patton, *Qualitative Research & Evaluation Methods* (SAGE Publications, Thousand Oaks, 2002)
378. J. Pearl, M. Glymour, N.P. Jewell, *Causal inference in statistics: A primer*. (John Wiley & Sons, Hoboken, 2016)
379. K. Pearson, Contributions to the mathematical theory of evolution, II: Skew variation in homogeneous material. *Trans. R. Philos. Soc. A* **186**, 343–414 (1895)
380. K. Pearson, On lines and planes of closest fit to systems of points in space. *Philos. Mag.* **2**, 559–572 (1901)
381. F. Pedregosa, G. Varoquaux, A.G. Gramfort et al., Scikit-learn: machine learning in Python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011)
382. H. Peng, F. Long, C. Ding, Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Trans. Pattern Anal. Mach. Intell.* **27**(8), 1226–1238 (2005)
383. J.D. Perezgonzalez, Fisher, Neyman-Pearson or NHST? A tutorial for teaching data testing. *Front. Psychol.* **6**, 223 (2015)
384. D. Phillips, D. Ghosh et al., Testing the disjunction hypothesis using Voronoi diagrams with applications to genetics. *Ann. Appl. Stat.* **8**(2), 801–823 (2014)
385. J. Piironen, A. Vehtari, Comparison of bayesian predictive methods for model selection. *Stat. Comput.* **27**(3), 711–735 (2017)
386. N. Pike, Using false discovery rates for multiple comparisons in ecology and evolution. *Methods Ecol. Evol.* **2**(3), 278–282 (2011)
387. K.S. Pollard, S. Dudoit, M.J. van der Laan, Multiple testing procedures: R multtest package and applications to genomics. uc berkeley division of biostatistics working paper series (2004). Technical report, Working Paper 164. <http://www.bepress.com/ucbbiostat/paper164>

388. J.C. Principe, D.X. Xu, Q. Zhao, J.W. Fisher, Learning from examples with information-theoretic criteria. *Signal Proces. Syst.* **26**(1–2), 61–77 (2000)
389. F. Provost, T. Fawcett, Data science and its relationship to big data and data-driven decision making. *Big Data* **1**(1), 51–59 (2013)
390. Y. Pu, Z. Gan, R. Henao et al., Variational autoencoder for deep learning of images, labels and captions, in *Advances in Neural Information Processing Systems* (2016), pp. 2352–2360
391. J. Quackenbush, *The Human Genome: The Book of essential knowledge*. Curiosity Guides (Imagine Publishing, New York, 2011)
392. B. Quast, RNN: a recurrent neural network in R. Working Papers (2016)
393. R Development Core Team, R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna (2008). ISBN 3-900051-07-0
394. A.E. Raftery, Bayesian model selection in social research. *Sociol. Methodol.* 111–163 (1995)
395. Y. Rahmatallah, F. Emmert-Streib, G. Glazko, Gene Sets Net Correlations Analysis (GSNCA): a multivariate differential coexpression test for gene sets. *Bioinformatics* **30**(3), 360–368 (2014)
396. Y. Rahmatallah, B. Zybailov, F. Emmert-Streib, G. Glazko, GSAR: bioconductor package for gene set analysis in R. *BMC Bioinf.* **18**(1), 61 (2017)
397. W. Rawat, Z. Wang, Deep convolutional neural networks for image classification: a comprehensive review. *Neural Comput.* **29**(9), 2352–2449 (2017)
398. J. Read, B. Pfahringer, G. Holmes, E. Frank, Classifier chains for multi-label classification. *Mach. Learn.* **85**(3), 333–359 (2011)
399. G.A. Rempala, Y. Yang, On permutation procedures for strong control in multiple testing with gene expression data. *Stat. Interf.* **6**(1) (2013)
400. M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning: the rprop algorithm, in *IEEE International Conference on Neural Networks* (1993)
401. O.Y. Rodionova, P. Oliveri, A.L. Pomerantsev, Rigorous and compliant approaches to one-class classification. *Chemom. Intell. Lab. Syst.* **159**, 89–96 (2016)
402. J.P. Romano, M. Wolf et al., Balanced control of generalized error rates. *Ann. Stat.* **38**(1), 598–633 (2010)
403. X. Rong, Deep Learning Toolkit in R (2014)
404. F. Rosenblatt, *The Perceptron, a Perceiving and Recognizing Automaton Project Para.* (Cornell Aeronautical Laboratory, Buffalo, 1957)
405. B. Rost, C. Sander, Prediction of protein secondary structure at better than 70 accuracy. *J. Mol. Biol.* **232**(2), 584–599 (1993)
406. P.J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Comput. Appl. Math.* **20**, 53–65 (1987)
407. S. Ruder, An overview of multi-task learning in deep neural networks (2017). Preprint. arXiv:1706.05098
408. L. Ruff, R. Vandermeulen, N. Goernitz et al., Deep one-class classification, in *International Conference on Machine Learning* (2018), pp. 4393–4402
409. J.S. Saczynski, S.E. Andrade, L.R. Harrold et al., A systematic review of validated methods for identifying heart failure using administrative data. *Pharmacoepidemiol. Drug Saf.* **21**(S1), 129–140 (2012)
410. R. Salakhutdinov, G. Hinton, Deep Boltzmann machines, in *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, PMLR* (2009), pp. 448–455
411. S. Santini, R. Jain, Similarity measures. *IEEE Trans. Pattern Anal. Mach. Intell.* **21**(9), 871–883 (1999)
412. F. Santosa, W.W. Symes, Linear inversion of band-limited reflection seismograms. *SIAM J. Sci. Stat. Comput.* **7**(4), 1307–1330 (1986)
413. R. Sarikaya, G. Hinton, A. Deoras, Application of deep belief networks for natural language understanding. *IEEE/ACM Trans. Audio Speech Lang. Proces.* **22**, 778–784 (2014)
414. S.K. Sarkar, On methods controlling the false discovery rate. *Sankhyā Indian J. Stat. A*, 135–168 (2008)

415. A.G. Sawyer, J.P. Peter, The significance of statistical significance tests in marketing research. *J. Market. Res.* **20**(2), 122–133 (1983)
416. B. Schölkopf, A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. (The MIT Press, Massachusetts, 2002)
417. B. Schölkopf, R.C. Williamson, A.J. Smola et al., Support vector method for novelty detection, in *Advances in Neural Information Processing Systems*, vol. 12 (Citeseer, 1999), pp. 582–588
418. C. Schaffer, A conservation law for generalization performance, in *Machine Learning Proceedings 1994* (Elsevier, Amsterdam, 1994), pp. 259–265
419. D. Scherer, A. Müller, S. Behnke, Evaluation of pooling operations in convolutional architectures for object recognition, in *Artificial Neural Networks – ICANN 2010*, ed. by K. Diamantaras, W. Duch, L.S. Iliadis. Lecture Notes in Computer Science (Springer, Berlin, 2010)
420. J. Schmidhuber, Deep learning in neural networks: an overview. *Neural Netw.* **61**, 85–117 (2015)
421. D. Schoenfeld, Partial residuals for the proportional hazards regression model. *Biometrika* **69**(1), 239–241 (1982)
422. M. Schumacher, N. Holländer, W. Sauerbrei, Resampling and cross-validation techniques: a tool to reduce bias caused by model building? *Stat. Med.* **16**(24), 2813–2827 (1997)
423. G. Schwarz et al., Estimating the dimension of a model. *Ann. Stat.* **6**(2), 461–464 (1978)
424. T. Schweder, E. Spjøtvoll, Plots of p-values to evaluate many tests simultaneously. *Biometrika* **69**(3), 493–502 (1982)
425. C. Seidel, Introduction to dna microarrays, in *Analysis of Microarray Data: A Network-Based Approach*, ed. by F. Emmert-Streib, M. Dehmer (Wiley-VCH, Weinheim, 2008), pp. 1–26
426. J.P. Shaffer, Multiple hypothesis testing. *Ann. Rev. Psychol.* **46**(1), 561–584 (1995)
427. S. Shalev-Shwartz, S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms* (Cambridge University Press, Cambridge, 2014)
428. S. Sheather, *A Modern Approach to Regression with R* (Springer Science & Business Media, Berlin, 2009)
429. D. Shen, G. Wu, H.-I. Suk, Deep learning in medical image analysis. *Ann. Rev. Biomed. Eng.* **19**, 221–248 (2017)
430. D.J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, 3rd edn. (RC Press, Boca Raton, 2004)
431. D.J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures* (CRC Press, Boca Raton, 2020)
432. G. Shmueli et al., To explain or to predict? *Stat. Sci.* **25**(3), 289–310 (2010)
433. D.V. Shridhar, E.B. Bartlett, R.C. Seagrave, Information theoretic subset selection for neural network models. *Comput. Chem. Eng.* **22**(4–5), 613–626 (1998)
434. Z. Šidák, Rectangular confidence regions for the means of multivariate normal distributions. *J. Am. Stat. Assoc.* **62**(318), 626–633 (1967)
435. R.J. Simes, An improved Bonferroni procedure for multiple tests of significance. *Biometrika* **73**(3), 751–754 (1986)
436. N. Simon, J. Friedman, T. Hastie, R. Tibshirani, A sparse-group lasso. *J. Comput. Graph. Stat.* **22**(2), 231–245 (2013)
437. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in *International Conference on Learning Representations* (2015)
438. D. Siroker, P. Koomen, *A/B Testing: The Most Powerful Way to Turn Clicks into Customers* (John Wiley & Sons, Hoboken, 2013)
439. J. Smolander, Deep learning classification methods for complex disorders. Master's thesis, The school of the thesis, Tampere University of Technology (2016). <https://dspace.cc.tut.fi/dpub/handle/123456789/23845>
440. J. Smolander, A. Stupnikov, G. Glazko et al., Comparing biological information contained in mRNA and non-coding RNAs for classification of lung cancer patients. *BMC Cancer* **19**(1), 1176 (2019)

441. J. Smolander, M. Dehmer, F. Emmert-Streib, Comparing deep belief networks with support vector machines for classifying gene expression data from complex disorders. *FEBS Open Bio.* **9**(7), 1232–1248 (2019)
442. Q. Song, An overview of reciprocal l1-regularization for high dimensional regression data. *Wiley Interdiscip. Rev. Comput. Stat.* **10**(1), e1416 (2018)
443. T. Sørbye, C.M. Perou, R. Tibshirani et al., Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications. *Proc. Natl. Acad. Sci.* **98**(19), 10869–10874 (2001)
444. S. Sosnin, M. Vashurina, M. Withnall et al., A survey of multi-task learning methods in chemoinformatics. *Mol. Inf.* **38**(4), 1800108 (2019)
445. P. Spirtes, Introduction to causal inference. *J. Mach. Learn. Res.* **11**(5) (2010)
446. A. Stang, H. Pohlabein, K.M. Müller et al., Diagnostic agreement in the histopathological evaluation of lung cancer tissue in a population-based case-control study. *Lung Cancer* **52**(1), 29–36 (2006)
447. J.R. Stevens, A. Al Masud, A. Suyundikov, A comparison of multiple testing adjustment methods with block-correlation positively-dependent tests. *PLoS One* **12**(4), e0176124 (2017)
448. M. Stone, Cross-validated choice and assessment of statistical predictions. *J. R. Stat. Soc. Ser. B Methodol.* 111–147 (1974)
449. A. Strehl, J. Ghosh, Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.* **3**(Dec), 583–617 (2002)
450. A. Stupnikov, S. Tripathi, R. de Matos Simoes et al., samExploreR: exploring reproducibility and robustness of RNA-seq results based on SAM files. *Bioinformatics*, 475 (2016)
451. F. Sung, Y. Yang, L. Zhang et al., Learning to compare: relation network for few-shot learning, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 1199–1208
452. R.S. Sutton, A.G. Barto, *Reinforcement Learning* (MIT Press, Cambridge, 1998)
453. M.R.E. Symonds, A. Moussalli, A brief guide to model selection, multimodel inference and model averaging in behavioural ecology using Akaike's information criterion. *Behav. Ecol. Sociobiol.* **65**(1), 13–21 (2011)
454. C. Szegedy et al., Going deeper with convolutions, in *2015 IEEE Conference on Computer Vision and Pattern Recognition CVPR* (2015), pp. 1–9
455. D. Szucs, J. Ioannidis, When null hypothesis significance testing is unsuitable for research: a reassessment. *Front. Hum. Neurosci.* **11**, 390 (2017)
456. L. Tarassenko, P. Hayton, N. Cerneaz, M. Brady, Novelty detection for the identification of masses in mammograms (1995)
457. D.M.J. Tax, One-class classification: concept learning in the absence of counter-examples. Ph.D. Thesis. Technische Universiteit Delft (2001)
458. J.B. Tenenbaum, V. de Silva, J.C. Langford, A global geometric framework for nonlinear dimensionality reductions. *Science* **290**(5500), 2319–2323 (2000)
459. Theano Development Team, Theano: a Python framework for fast computation of mathematical expressions (2016). arXiv e-prints, abs/1605.02688
460. T.M. Therneau, A Package for Survival Analysis in S (2015). version 2.38
461. T.M. Therneau, P.M. Grambsch, *Modeling Survival Data: Extending the Cox Model* (Springer Science & Business Media, Berlin, 2013)
462. R. Tibshirani, Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Ser. B* **58**, 267–288 (1996)
463. A.N. Tikhonov, On the stability of inverse problems, in *Doklady Akademii Nauk SSSR*, vol. 39 (1943), pp. 195–198
464. I. Tosic, P. Frossard, Dictionary learning. *IEEE Sig. Proces. Mag.* **28**(2), 27–38 (2011)
465. N. Trinajstić, *Chemical Graph Theory* (CRC Press, Boca Raton, 1992)
466. S. Tripathi, F. Emmert-Streib, mvgraphnorm: multivariate Gaussian graphical models (2019). R package version 1.0.0
467. G. Tsoumakas, I. Katakis, Multi-label classification: an overview. *Int. J. Data Warehouse. Min.* **3**(3), 1–13 (2007)

468. G. Tsoumakas, I. Katakis, I. Vlahavas, Mining multi-label data, in *Data Mining and Knowledge Discovery Handbook* (Springer, Berlin, 2009), pp. 667–685
469. G. Tsoumakas, I. Katakis, I. Vlahavas, Random k-labelsets for multilabel classification. *IEEE Trans. Knowl. Data Eng.* **23**(7), 1079–1089 (2010)
470. J.W. Tukey, *Exploratory Data Analysis* (Addison-Wesley, New York, 1977)
471. G. Tutz, J. Ulbricht, Penalized regression with correlation-based penalty. *Stat. Comput.* **19**(3), 239–253 (2009)
472. U.N. Umesh, R.A. Peterson, M.H. Sauber, Interjudge agreement and the maximum value of kappa. *Educ. Psychol. Meas.* **49**, 835–850 (1989)
473. I. Unal, Defining an optimal cut-point value in roc analysis: an alternative approach. *Comput. Math. Methods Med.* **2017** (2017)
474. L.G. Valiant, A theory of the learnable. *Commun. ACM* **27**(11), 1134–1142 (1984)
475. M.J. Van De Vijver, Y.D. He, L.J. Van't Veer et al., A gene-expression signature as a predictor of survival in breast cancer. *N. Engl. J. Med.* **347**(25), 1999–2009 (2002)
476. S. van de Geer, *L1-regularization in High-Dimensional Statistical Models* (World Scientific, Singapore, 2011), pp. 2351–2369
477. J.E. Van Engelen, H.H. Hoos, A survey on semi-supervised learning. *Mach. Learn.* **109**(2), 373–440 (2020)
478. V.N. Vapnik, *The Nature of Statistical Learning Theory* (Springer, Berlin, 1995)
479. S. Venkataraman, Z. Yang, D. Liu et al., SparkR: scaling R programs with spark, in *Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16* (ACM, New York, 2016), pp. 1099–1104
480. P. Vincent, H. Larochelle, I. Lajoie et al., Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.* **11**(Dec), 3371–3408 (2010)
481. O. Vinyals, A. Toshev, S. Bengio, D. Erhan, Show and tell: a neural image caption generator, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 3156–3164
482. O. Vinyals, C. Blundell, T. Lillicrap et al., Matching networks for one shot learning (2016). Preprint. arXiv:1606.04080
483. U. Von Luxburg, B. Schölkopf, Statistical learning theory: models, concepts, and results, in *Handbook of the History of Logic*, vol. 10 (Elsevier, Amsterdam, 2011), pp. 651–706
484. S.I. Vrieze, Model selection and psychological theory: a discussion of the differences between the Akaike information criterion (AIC) and the bayesian information criterion (BIC). *Psychol. Methods* **17**(2), 228 (2012)
485. Q.H. Vuong, Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica*, 307–333 (1989)
486. H. Wallach, Evaluation metrics for hard classifiers. Technical report. Cambridge University (2006)
487. L. Wan, M. Zeiler, S. Zhang et al., Regularization of neural networks using DropConnect, in *Proceedings of the 30th International Conference on Machine Learning, PMLR* (2013), pp. 1058–1066
488. J. Wand, X. Shen, Estimation of generalization error: random and fixed inputs. *Stat. Sin.* **16**(2), 569 (2006)
489. Z. Wang, M. Gerstein, M. Snyder, RNA-Seq: a revolutionary tool for transcriptomics. *Nat. Rev. Genet.* **10**, 57–63 (2009)
490. Y. Wang, M. Huang, L. Zhao et al., Attention-based LSTM for aspect-level sentiment classification, in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (2016), pp. 606–615
491. Y. Wang, Q. Yao, J.T. Kwok, L.M. Ni, Generalizing from a few examples: a survey on few-shot learning. *ACM Comput. Surv.* **53**(3), 1–34 (2020)
492. J.H. Ward, Hierarchical grouping to optimize an objective function. *J. Am. Stat. Assoc.* **58**, 236–244 (1963)

493. R.L. Wasserstein, N.A. Lazar et al., The ASA's statement on p-values: context, process, and purpose. *Am. Stat.* **70**(2), 129–133 (2016)
494. R.L. Wasserstein, A.L. Schirm, N.A. Lazar, Moving to a World Beyond $p < 0.05$. *Am. Stat.* **73**(sup1), 1–19 (2019)
495. A.R. Webb, K.D. Copesey, *Statistical Pattern Recognition*, 3rd edn. (Wiley, Hoboken, 2011)
496. R. Wehrens, H. Putter, L.M.C. Buydens, The bootstrap: a tutorial. *Chemom. Intell. Lab. Syst.* **54**(1), 35–52 (2000)
497. K. Weinberger, Lecture Notes in Machine Learning (CS4780/CS5780) (2017). <http://www.cs.cornell.edu/courses/cs4780/2017sp/lectures/lecturenote11.html>
498. K. Weiss, T.M. Khoshgoftaar, D. Wang, A survey of transfer learning. *J. Big Data* **3**(1), 9 (2016)
499. P.H. Westfall, On using the bootstrap for multiple comparisons. *J. Biopharmaceut. Stat.* **21**(6), 1187–1205 (2011)
500. P.H. Westfall, J.F. Troendle, Multiple testing with minimal assumptions. *Biomet. J. J. Math. Methods Biosci.* **50**(5), 745–755 (2008)
501. P.H. Westfall, S.S. Young et al., *Resampling-Based Multiple Testing: Examples and Methods for p-Value Adjustment*, vol. 279. (John Wiley & Sons, Hoboken, 1993)
502. D.R. Wilson, T.R. Martinez, Bias and the probability of generalization, in *Proceedings Intelligent Information Systems. IIS'97* (IEEE, Piscataway, 1997), pp. 108–114
503. D.H. Wolpert, The supervised learning no-free-lunch theorems. *Soft Comput. Ind.*, 25–42 (2002)
504. S. Wright, Correlation and causation. *J. Agricult. Res.* **20**, 557–585 (1921)
505. Z. Wu, S. Pan, F. Chen et al., A comprehensive survey on graph neural networks (2019). Preprint. arXiv:1901.00596
506. S. Xingjian, Z. Chen, H. Wang et al., Convolutional lstm network: A machine learning approach for precipitation nowcasting, in *Advances in Neural Information Processing Systems* (2015), pp. 802–810
507. S. Xiong, B. Dai, J. Huling, P.Z.G. Qian, Orthogonalizing EM: a design-based least squares algorithm. *Technometrics* **58**, 285–293 (2016)
508. Y. Yang, Can the strengths of AIC and BIC be shared? A conflict between model identification and regression estimation. *Biometrika* **92**(4), 937–950 (2005)
509. Y. Yang, H. Zou, gglasso: group lasso penalized learning using a unified BMD algorithm. R package version, 1 (2013)
510. Z. Yang, M. Dehmer, O. Yli-Harja, F. Emmert-Streib, Combining deep learning with token selection for patient phenotyping from electronic health records. *Sci. Rep.* (2020)
511. L. Yao, C. Mao, Y. Luo, Graph convolutional networks for text classification, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33 (2019), pp. 7370–7377
512. W.J. Youden, Index for rating diagnostic tests. *Cancer* **3**(1), 32–35 (1950)
513. T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing. *IEEE Comput. Intel. Mag.* **13**(3), 55–75 (2018)
514. D. Yu, J. Li, Recent progresses in deep learning based acoustic models. *IEEE/CAA J. Automat. Sin.* **4**(3), 396–409 (2017)
515. M. Yuan, Y. Lin, Model selection and estimation in regression with grouped variables. *J. R. Stat. Soc. Series B Stat. Methodol.* **68**(1), 49–67 (2006)
516. M. Yuan, Y. Lin, On the non-negative garrotte estimator. *J. R. Stat. Soc. Ser. B Stat Methodol.* **69**(2), 143–161 (2007)
517. Y. Zhang, Q. Yang, An overview of multi-task learning. *Natl. Sci. Rev.* **5**(1), 30–43 (2018)
518. Z. Zhang, H. Zha, Principal manifolds and nonlinear dimensionality reduction via local tangent space alignment. *SIAM J. Sci. Comput.* **26**(1), 313–338 (2004)
519. M.-L. Zhang, Z.-H. Zhou, ML-KNN: a lazy learning approach to multi-label learning. *Pattern Recogn.* **40**(7), 2038–2048 (2007)
520. M.-L. Zhang, Z.-H. Zhou, A review on multi-label learning algorithms. *IEEE Trans. Knowl. Data Eng.* **26**(8), 1819–1837 (2013)

- 521. B. Zhang, W. Zuo, Learning from positive and unlabeled examples: a survey, in *2008 International Symposiums on Information Processing* (IEEE, Piscataway, 2008), pp. 650–654
- 522. W. Zhang, T. Ota, V. Shridhar et al., Network-based survival analysis reveals subnetwork signatures for predicting outcomes of ovarian cancer treatment. *PLoS Comput. Biol.* **9**(3), e1002975 (2013)
- 523. S. Zhang, J. Zhou, H. Hu et al., A deep learning framework for modeling structural features of RNA-binding protein targets. *Nucleic Acids Res.* **44**(4), e32–e32 (2015)
- 524. Y. Zhou, Sentiment classification with deep neural networks. Master's thesis (2019)
- 525. N. Zhou, J. Zhu, Group variable selection via a hierarchical lasso and its oracle property (2010). Preprint. arXiv:1006.2871
- 526. X. Zhu, A.B. Goldberg, Introduction to semi-supervised learning. *Synth. Lect. Artif. Intel. Mach. Learn.* **3**(1), 1–130 (2009)
- 527. F. Zhuang, Z. Qi, K. Duan et al., A comprehensive survey on transfer learning. *Proc. IEEE* **109**(1), 43–76 (2020)
- 528. H. Zou, The adaptive lasso and its oracle properties. *J. Am. Stat. Assoc.* **101**(476), 1418–1429 (2006)
- 529. H. Zou, T. Hastie, Regularization and variable selection via the elastic net. *J. R. Stat. Soc. Ser. B Stat Methodol.* **67**(2), 301–320 (2005)
- 530. C. Zuccaro, Mallows' cp statistic and model selection in multiple linear regression. *Market Res. Soc. J.* **34**(2), 1–10 (1992)

Index

A

Adaptive LASSO, 347
Akaike's information criterion (AIC), 302, 314–315
Alternative hypothesis, 26, 242–243, 248, 250, 260, 264, 267, 270, 280, 304, 320, 421
Architecture, 83, 360–365, 372, 379, 380, 391, 407, 416, 512
Area under the receiver operator characteristic curve, 41–44
Artificial intelligence, 1–4, 11, 24, 91, 359
Artificial neuron, 360–362
Autoencoders, 164, 360, 365, 391, 392, 416

B

Bayesian credible intervals, 118–121
Bayesian inference, 9, 92, 110–123, 134
Bayesian information criterion (BIC), 314–315
Bias-variance trade-off, 519, 523–528, 540
Biased estimator, 106–107, 135
Boltzmann machine, 359, 384, 385, 416
Bonferroni correction, 422, 432–433, 435, 443, 448
Bootstrap, 53, 54, 59–61, 70, 127–128, 244, 431
Business data, 9, 72, 85–87

C

Causal model, 22–23, 27, 410
Censoring, 83, 84, 455–456, 460, 474, 479, 481, 485

Classification, 9, 20–22, 28–30, 38, 40, 42, 46, 54, 61, 154, 156, 174, 191–236, 273, 306, 309, 359, 366, 368, 381, 382, 403, 415, 487–490, 495, 498, 504, 510, 516, 517, 539
Cluster validation, 137, 155–160
Clustering, 9, 21, 74, 137–161, 487
Coefficient of determination (COD), 312
Coefficients, 153, 277–281, 284–285, 291, 293, 300, 303, 333, 336–339, 345–347, 349, 354, 357, 468, 474, 478, 481, 484
Cohen's kappa, 39–40
Confidence interval, 127–129, 240, 253–256, 279, 323, 479, 484
Conjugate prior, 112, 113, 117
Contingency table, 28, 30–32, 37, 45, 48, 50, 157, 158, 191, 195, 196, 262, 421
Convolutional neural networks (CNN), 360, 365, 372–382, 392, 416, 512
Correlation test, 259–261, 473, 474
Cox model, 453, 477–479, 485
Cross validation (CV), 9, 50

D

Dantzig selector, 333, 345–346, 355, 356
Data, 1–14, 18–19, 24–28, 60, 71–87, 92–104, 138–143, 215–221
Data analysis process, 5
Data science, 2–4, 7, 9, 21, 27, 50, 54, 70, 72, 91, 135, 154, 161, 177, 272, 410, 501, 505, 519
Data types, 5, 9, 19, 72–87, 138, 266, 415

Decision making, 29–32, 44, 47–49, 191, 194, 195, 225, 239
 Decision tree, 9, 20, 23, 191, 223–235, 309, 517
 Deep belief networks (DBN), 360, 382–390, 416
 Deep feedforward neural networks (DFNN), 360, 365–372, 384, 398
 Deep learning, 359–416
 Deep neural networks, 20, 21, 333, 360
 Dendrogram, 149–151
 Dimension reduction, 9, 163–188, 487
 Distance, 42, 49, 63, 137, 140, 144–152, 154, 159, 200–202, 212, 217, 218, 356, 502, 512, 534, 538

E

Elastic net, 329, 333, 336, 348–352, 355, 520, 539
 Error measures, 9, 24, 29–50, 194–196, 202, 205, 420, 490, 519
 Error-complexity curves, 519, 528–535, 539, 540
 Estimation, 9, 19, 50, 54–61, 92, 104–107, 123–129, 212, 244, 268, 277–279, 284, 299, 302, 324, 336, 386, 416, 422, 474–476, 508, 520, 527, 539
 Euclidean norm, 334
 Event, 83–85, 98, 104, 174, 227, 453–456, 458–461, 467, 473, 484
 Expectation-maximization (EM) algorithm, 9, 92, 129–134
 Expected generalization error, 17, 309, 310, 330, 521–523, 528, 532, 539
 Expected out-of-sample error, 522
 Explainable AI (XAI), 23, 410–415
 Exploratory data analysis (EDA), 8, 9, 74, 92–104, 134, 161

F

F-score, 24, 34–35, 46, 157, 415
 False-discovery rate (FDR), 36, 419, 421
 Family-wise error, 419, 421, 431, 432
 Feature extraction, 164–184, 188
 Feature selection, 164, 184–188, 390, 516
 Few/one-shot learning, 19, 488, 510–512
 Fundamental errors, 29, 31, 33, 34, 36–38, 42, 45–50, 194–196, 202

G

Generalized linear models, 3, 21, 273, 297–306, 318–324
 Genomic data, 9, 72–74
 Group LASSO, 333, 336, 352–355

H

Hazard function, 453, 459, 461–467, 469, 477
 Hazard ratios, 470–471, 483–484
 Hierarchical clustering, 149–152
 Hochberg correction, 435–436
 Holdout set (HOS), 54–55, 58, 69, 327
 Holm correction, 434–435
 Hommel correction, 436–439, 448
 Hypergeometric test, 262–265, 272
 Hypothesis testing, 9, 10, 19, 50, 78, 239–272, 279, 419, 451, 453, 508

I

Interactions, 292

K

K-means clustering, 145–147, 159
 Kaplan-Meier estimator, 458–459

L

Least absolute shrinkage and selection operator (LASSO), 329, 333, 336, 339–344
 Layer, 138, 359, 362, 366, 368, 371, 374, 377, 382, 389, 407
 Learning curves, 53, 415, 519, 523, 535–540
 Leave-one-out cross validation (LOO-CV), 53, 55–56, 69
 Linear regression, 273–306, 312–318
 Link function, 297–300, 302, 305, 318
 Logistic regression, 9, 21, 42, 44, 45, 208, 210, 235, 305
 Long short-term memory networks (LSTM), 398–410, 415, 416
 Loss function, 368, 389, 392, 404, 407, 504, 522

M

Machine learning, 1–3, 11, 17, 37, 79, 91, 135, 163, 186, 416, 487, 505–517
 Mallows's C_p statistic, 313–314
 Mathematics, 3, 4, 91, 490
 Matthews correlation coefficient, 37–39

Maximum likelihood estimation, 9, 92,
123–129, 134, 198, 299, 302, 314, 386,
540
Maximum norm, 334
Measure of location, 94
Measure of scale, 98
Measures of shape, 93, 99–101
Minimum redundancy and maximum relevance
(MRMR), 164, 188
Model assessment, 10, 54, 309–311, 326, 520,
523, 527, 539–541
Model identification, 310
Model selection, 122, 309–330, 357, 519, 520,
539
Multi-label learning, 516–517
Multi-task learning, 513, 515–516
Multiple testing corrections (MTC), 10, 272,
419–451
Multiple testing procedures (MTP), 419,
424–425, 428–431, 449

N

Naive Bayesian classifier, 197–203, 235
Nearest neighbor classifier, 191, 212, 235
Nelson-Aalen estimator, 459
Network data, 9, 72, 74–79
Non-hierarchical clustering, 145–147
Non-negative garrote regression, 333, 339, 356
Non-negative matrix factorization (NNMF),
164, 179–184
Nonlinearities, 46, 273, 292, 306
Normalized mutual information, 40–41, 50,
157
Null hypothesis, 26, 27, 158, 241–244, 250,
253, 257, 259, 260, 263–265, 270, 280,
286, 288, 304, 319, 320, 323, 422, 429,
432, 473, 481

O

One-class classification, 487, 507–508
Ordinary least squares, 273, 277–279, 284,
291, 333, 343
Out-of-sample error, 63, 186, 199, 523, 527
Overfitting, 177, 188, 227, 230, 313, 315, 372,
379–381, 394, 531, 533, 540

P

P-value, 26, 241, 248–249, 252, 264, 268, 270,
279, 286, 288, 303, 304, 319, 320, 420,
422, 424, 428, 429, 432–440, 442–449,
451, 479, 481

Per comparison error rate, 421
Per family error rate, 419, 421
Performance, 37, 42, 46, 175, 177, 186, 202,
227, 280, 291, 310, 344, 379, 381, 392,
404, 407, 484, 495, 515, 520, 531, 535,
537
Permutation test, 267–269
Point estimation, 104–105, 118
Poisson regression, 300, 318, 320, 322, 323
Positive-unlabeled learning, 487, 507–510
Power, 188, 240, 250–253, 410, 443, 446, 449,
460, 516
Prediction model, 9, 17–28, 53, 66, 70, 87,
192, 237, 309, 359, 410, 491, 492, 519,
539
Predictive model, 22–23, 27, 507
Principal component analysis (PCA), 164
Probably approximately correct (PAC), 487,
490–498, 505
Programming, 3, 4, 9, 11, 12, 63, 269, 345,
420, 423
Proportional hazard model, 469–474

Q

Quantification, 29, 50, 118, 153, 240, 243, 273,
517, 519

R

Recurrent neural networks (RNN), 363–364
Regularization, 10, 282, 291, 329, 333–356,
372, 386, 503, 504, 516, 520, 537, 539
Resampling methods, 3, 9, 25, 53–70, 326,
327, 539
Residual, 277, 286–289, 312, 336, 382, 521
Residual standard error, 279
Residual sum of squares (RSS), 278, 336, 522
Ridge regression, 291, 333, 334, 336–339,
342–345, 349, 350, 355, 539
Right censoring, 455, 456

S

Sampling distribution, 105, 126, 127, 158, 241,
243–246, 248, 250, 253, 255, 266–269,
272, 420, 508
Sampling from a distribution, 9, 54, 63–65,
398
Semi-supervised learning, 18, 19, 487,
506–508
Significance level, 26, 241, 253, 268, 420, 422,
425–427, 431–433, 443, 479
Similarity, 20, 74, 83, 137, 140, 142, 161, 187,
212, 314, 415, 510, 511, 514

Singular value decomposition (SVD),
 168–169, 344
 Standard error, 9, 50, 54, 56, 58, 66–68, 98,
 233, 252, 279, 291, 483
 Statistical inference, 91–135,
 279
 Statistical learning, 488–501, 504
 Statistics, 2, 3, 11, 17, 22, 25, 63, 93–94, 104,
 134, 191, 239, 240, 242, 257, 258, 415,
 431, 439, 440, 488
 Stepwise selection, 316–318, 428, 451
 Student's t-test, 9, 14, 256–259
 Subsampling, 9, 53, 61–62
 Subset selection, 316, 339
 Supervised learning, 18, 191, 192, 235, 273,
 306, 387, 505, 519
 Support vector machine (SVM), 20, 191,
 216–223, 235, 502, 508, 539
 Survival analysis, 21, 85, 453–485
 Survival curves, 453, 459–461, 470–473,
 479
 Survival function, 457–459, 461–463,
 474

T

Test statistic, 158, 240, 242–244, 248, 253,
 258, 264, 266, 272, 286, 420, 431, 435,
 439, 440, 461
 Text data, 9, 72, 79–83, 87, 407, 415
 Time, 72, 84, 103, 131, 183, 377
 Time-to-event data, 9, 72, 83–85, 453, 455,
 484
 Transfer learning, 19, 288, 512–515
 Type 1 error, 250, 252, 419, 421, 428, 451
 Type 2 error, 250–253, 419, 449

U

Unbiased estimator, 105–106, 135, 313
 Underfitting, 315, 531, 533, 534, 537, 540
 Unsupervised learning, 9, 18, 19, 137, 164,
 179, 384, 392, 411, 415, 487, 505

V

Vapnik-Chervonenkis (VC) dimension, 487,
 499