Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

CAS Practical Machine Learning

Unsupervised Learning

Prof. Dr. habil. Matthias Dehmer; matthias.dehmer@umit.at

Email: matthias.dehmer@bfh.ch, matthias.dehmer@ffhs.ch

Professor for Data Science at FFHS, Brig, Switzerland and Research Professor at UMIT (Austria)
Experience:

- Data Science, Analytics, Network Analysis, Statistics
- Process Mining, Complex Networks

Guest Professor at Nankai University, China (Thousend Talents Program; 2016-2019)
Professor for Business Analytics and Data Science at University of Applied Sciences Upper Austria 2017-2019
University Professor at UMIT, Austria for Data Science (2008- to date)
,

Habilitation in Applied Mathematics at Vienna University of Technology (2008)
PhD (Dr.rer.nat.) at Darmstadt University of Technology (2005)
Diplom Mathematiker at University of Siegen (1998)
Physikalisch - Technischer Assistent at HBF Mainz (1991)

Outline

1. Unsupervised Learning vs. Supervised Learning
2. Clustering Techniques
   - K-Means Clustering: number and quality of clusters
   - k-Medoids/PAM (Partition Around Medoids)
   - k-Modes (Clustering with categorical attributes)
   - DBSCAN
   - Hierarchical Clustering
   - Density-based Methods (Gaussian Mixture Models)

Unsupervised Learning

Introduction

Types of Machine Learning

Supervised Learning
Überwachtes Lernen

Input: Feature Vectors and Data Labels (desired output)
Output: Model for prediction

Approaches
- Classification: binary, multiclass
- Regression

Applications
- Spam Filtering
- Image Classification
- Gesture Recognition
- Fraud Detection

Unsupervised Learning
Unüberwachtes Lernen

Input: Unlabeled Feature Vectors
Output: Model discovered under-lying structure of the data

Approaches
- Clustering
- Dimension Reduction
- Discovering Association Rules

Applications
- Exploratory Data Analysis
- Outlier Detection
- Data Compression
- Denoising

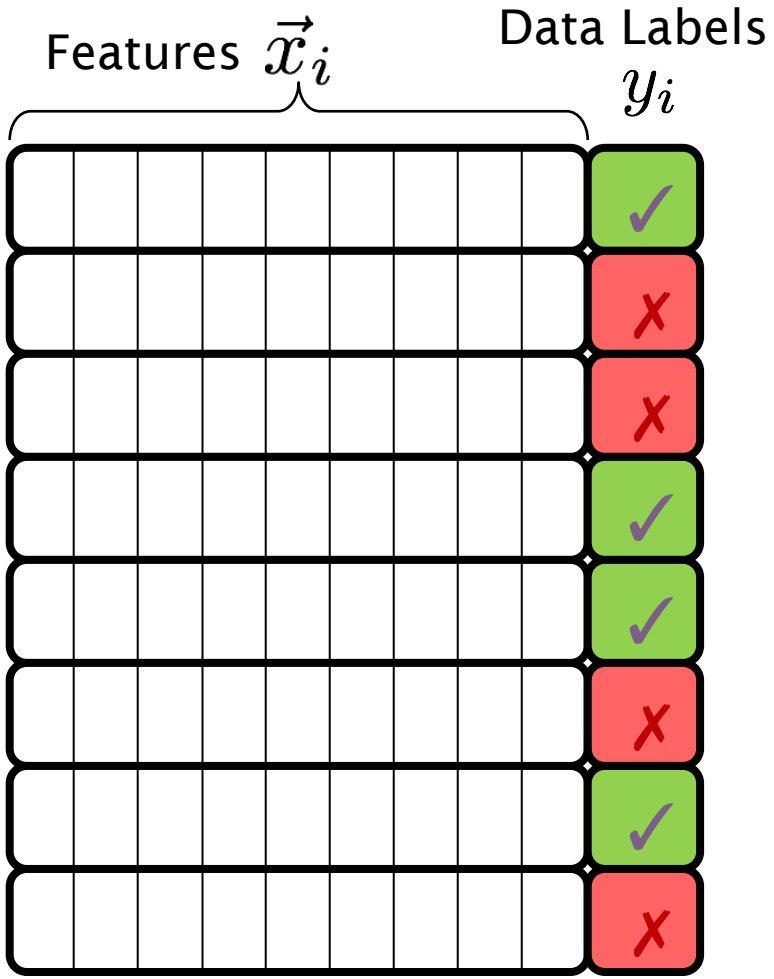
Reinforcement Learning
Bestärkendes Lernen

Input: Current observed state from environment and reward score
Output: Agent chooses next action

Approaches
- Control
- Q-Learning

Application
- Games (Go, Chess)
- AlphaGo

Recap: Supervised Learning



- Model

$$f(\vec{\theta}, \vec{x}_i) = y_i$$

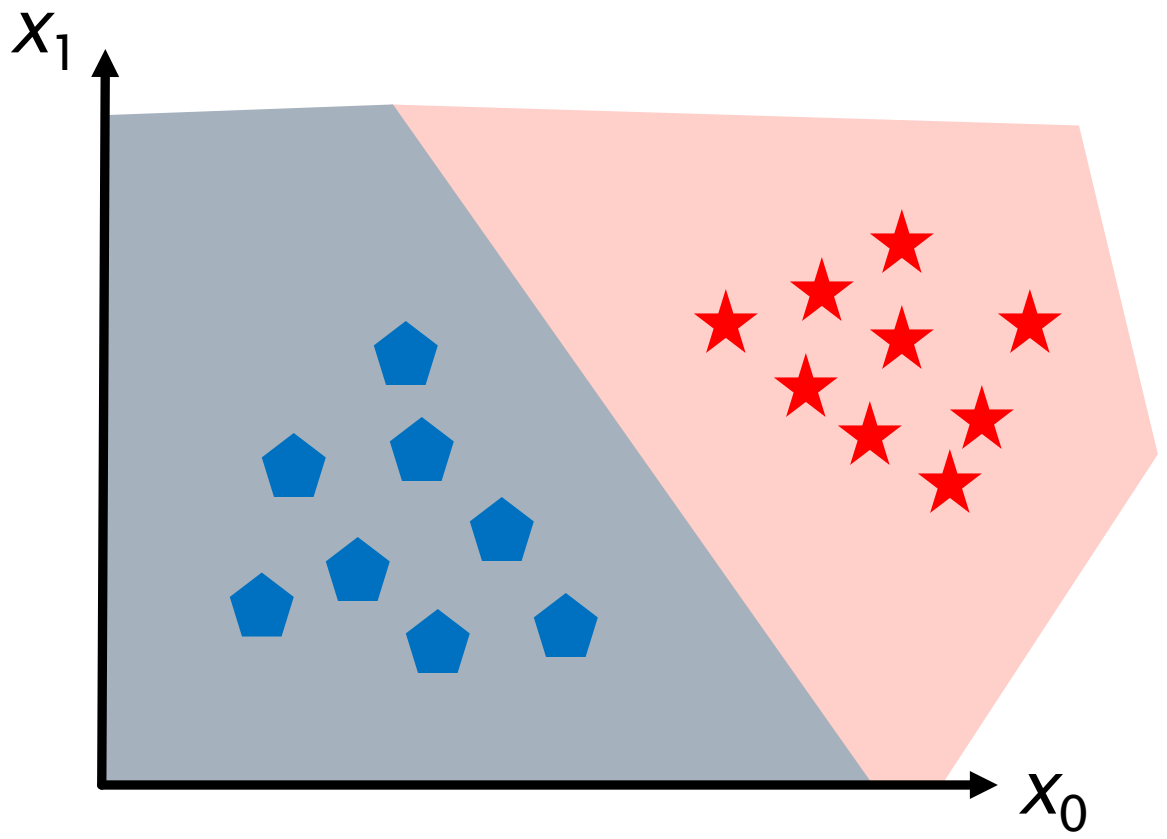
- **Training**: Find parameter vector $\vec{\theta}$ such that model produces y_i for input $\vec{x}_i$ for all data points i.

- Optimization: minimize squared error, for example:

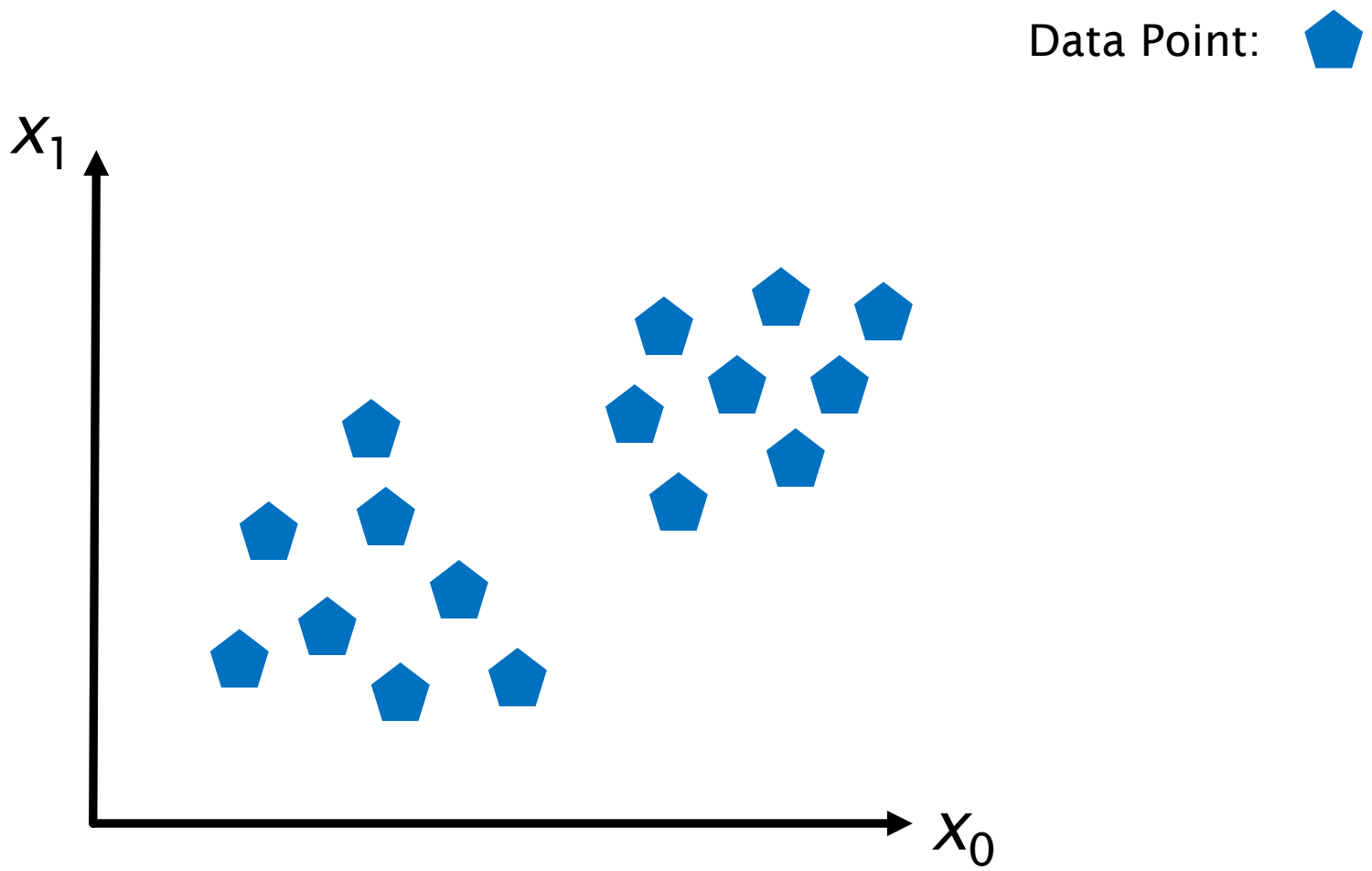
$$\sum_i \left[f(\vec{\theta}, \vec{x}_i) - y_i \right]^2 \rightarrow \min$$

Supervised Learning

Labeled Data Points:



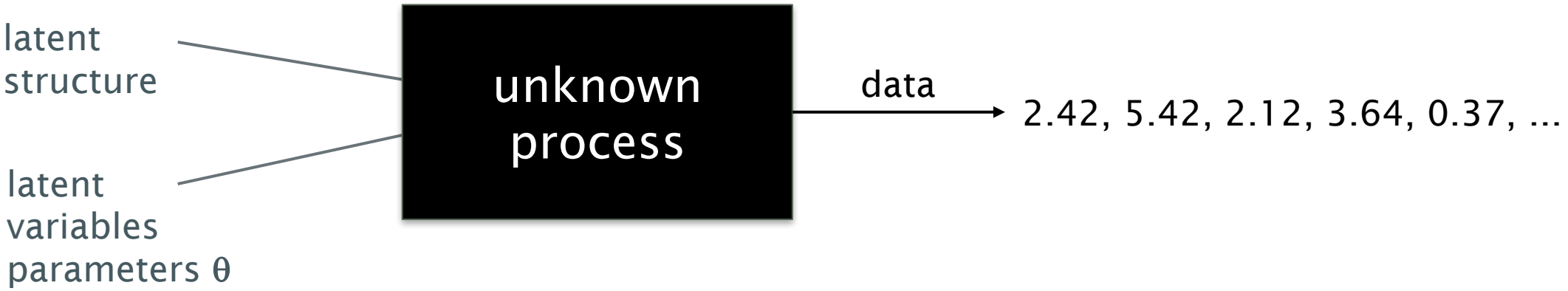
Unsupervised Learning



Unsupervised Learning: No Labels

- What is there to learn?
  - There is no outcome variable (a label) to learn!

- One way to think of unsupervised learning is the following:
  - Data is generated by some unknown process.
  - The process is characterized by some latent (versteckte) variables or features.
  - Unsupervised learning is the attempt to discover information about structure of the process or the latent variables given the unlabeled data.



Unsupervised Learning

Unsupervised learning finds patterns in the data.

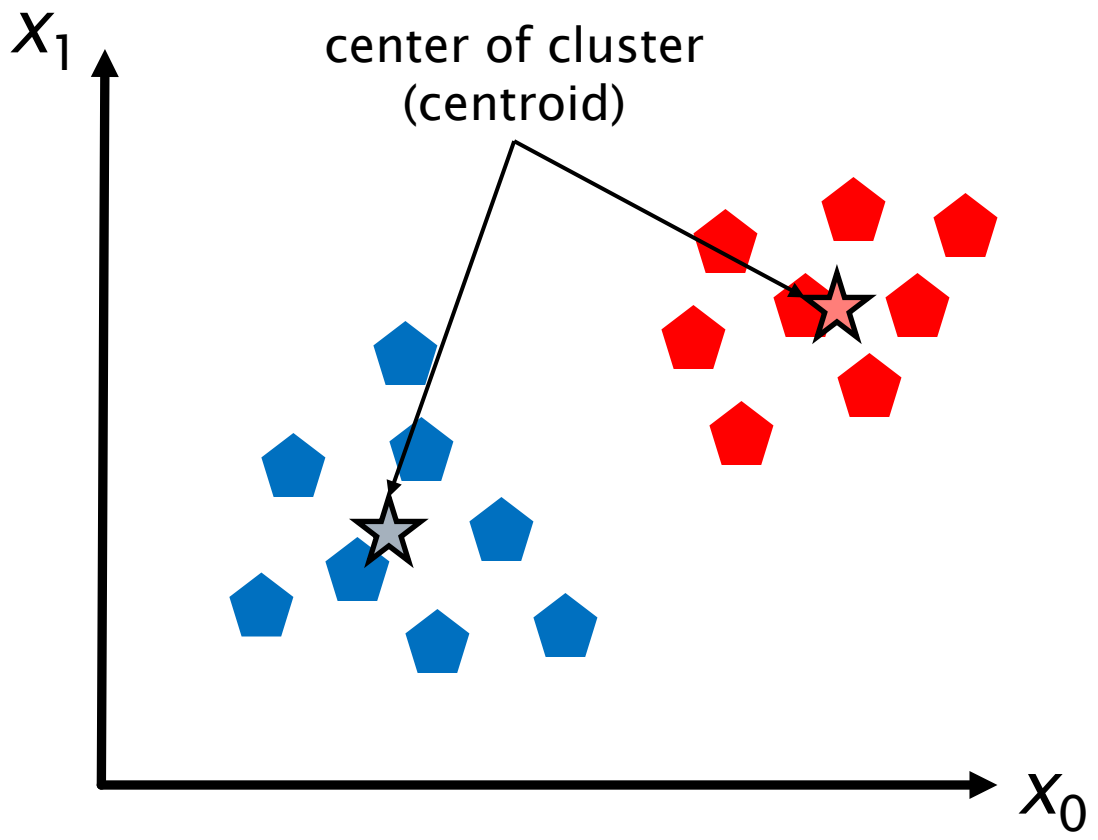
Common Methods in the domain of unsupervised learning:

- Clustering (k-Means, Hierarchical, DBSCAN, ...)
- Density Estimation: Gaussian Mixture Models
- Dimensionality Reduction (discussed later in Feature Engineering Module)
  - Principal Component Analysis, Singular Value Decomposition
  - Latent Discriminant Analysis
- Neural Networks
  - Autoencoder
  - Generative Adversarial Models
  - Restricted Boltzman Machines
- ...

Clustering

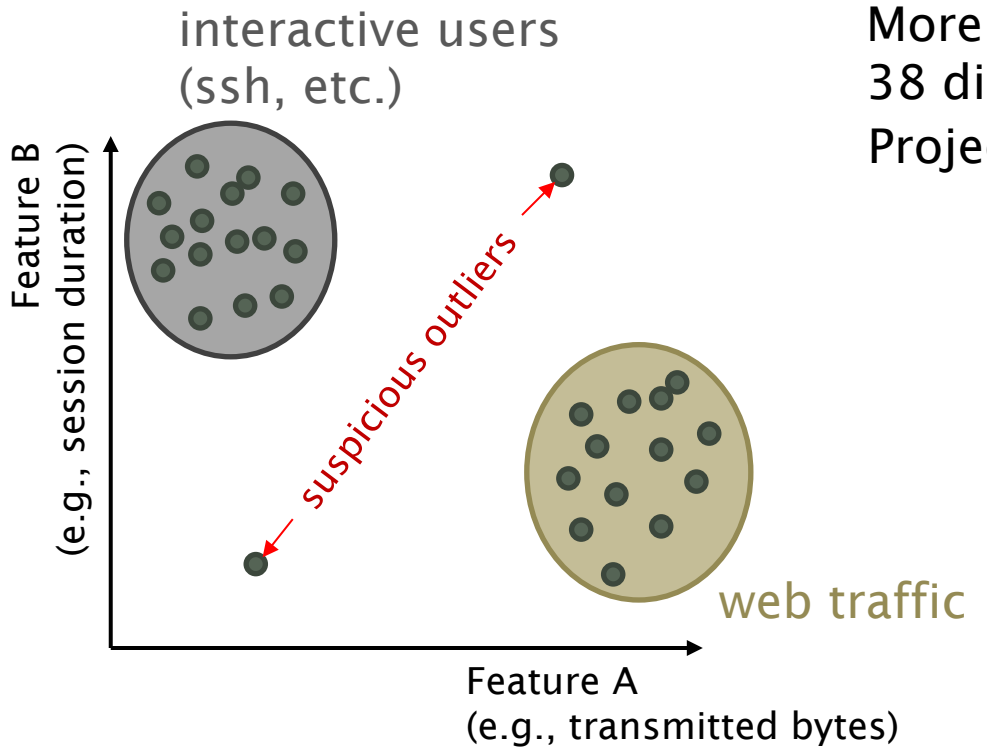
- Clustering is the process of organizes data into subgroups
- Elements in a subgroup are similar to each other.

- Use some kind of distance metric (measure) to identify clusters
  - Distance measure determines the tightness of the cluster (intra-cluster measure).
  - Connectivity measure between clusters (inter-cluster measure)

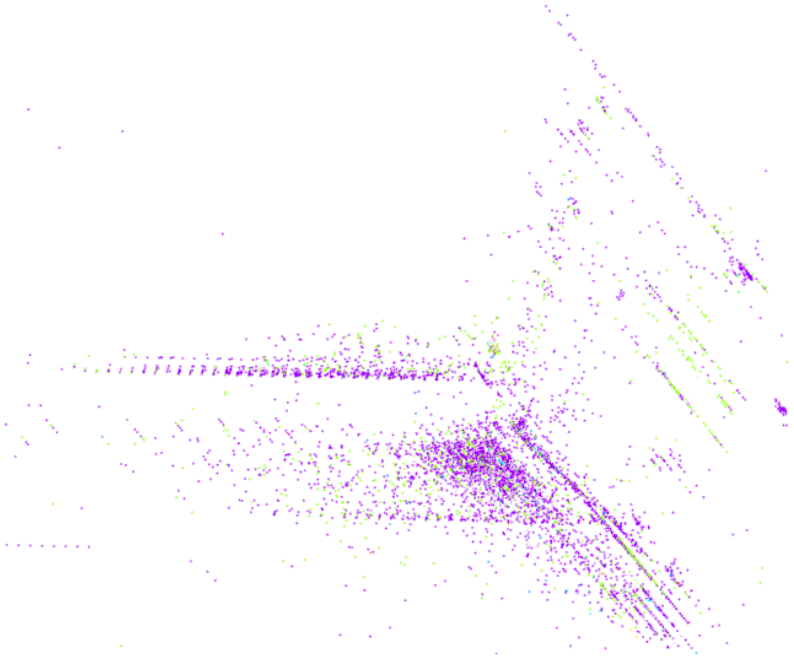


Application of Clustering

Application: Intrusion Detection in Networks.
Data points far away from cluster centers are “suspicious guys”.



More complex in practice: KDD Cup dataset has 38 dimensions
Projection on three arbitrary random axes:



KDD Cup 1999: Computer network intrusion detection

Density Estimation: A Probabilistic View

- Observed data points are regarded to be distributed by some unknown distribution.
- Let **x** be a data point that is sampled from the unknown distribution.
- The distribution can be characterized by a probability density function $p(\mathbf{x})$.

- **Example**: Estimate parameters $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$, $\boldsymbol{\Sigma}_1$, and $\boldsymbol{\Sigma}_2$ of the bimodal multivariate Gaussian distribution:

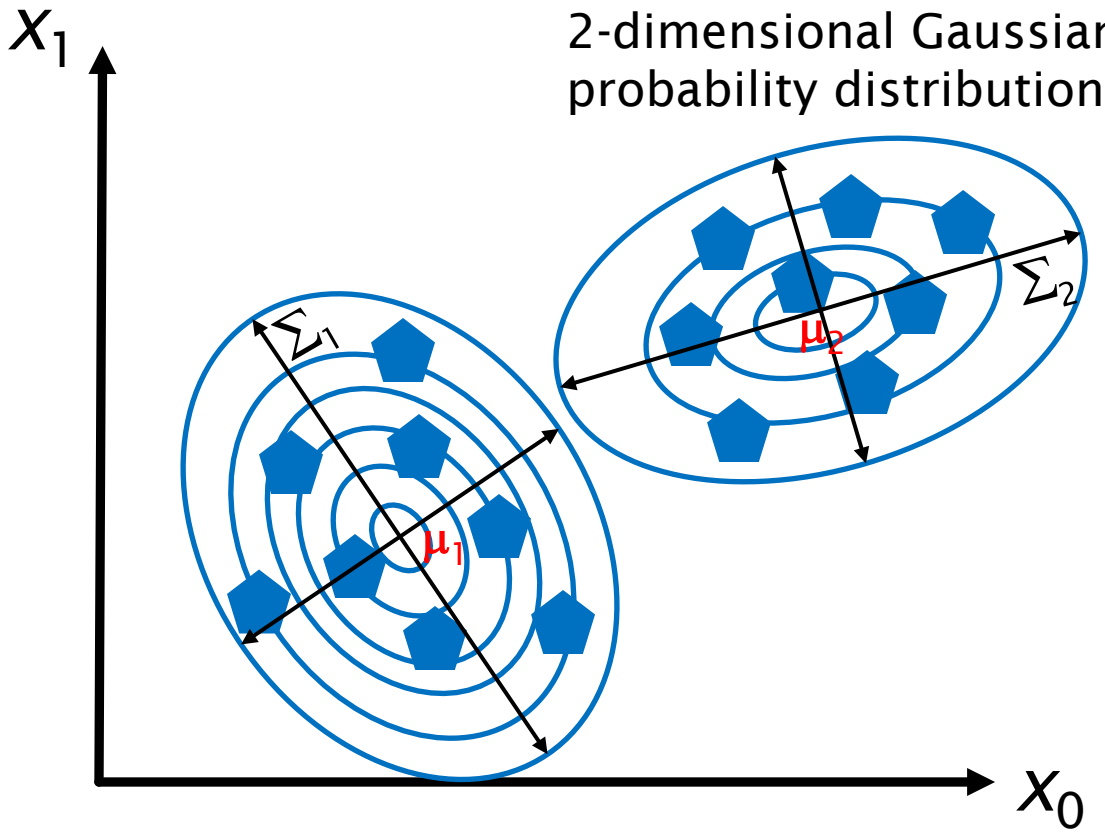
$$p(\mathbf{x}) = \frac{1/2}{\sqrt{|2\pi\boldsymbol{\Sigma}_1|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^{\mathrm{T}}\boldsymbol{\Sigma}_1^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)} + \frac{1/2}{\sqrt{|2\pi\boldsymbol{\Sigma}_2|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^{\mathrm{T}}\boldsymbol{\Sigma}_2^{-1}(\mathbf{x}-\boldsymbol{\mu}_2)}$$

Determinant

Covariance Matrices

Mean Vectors

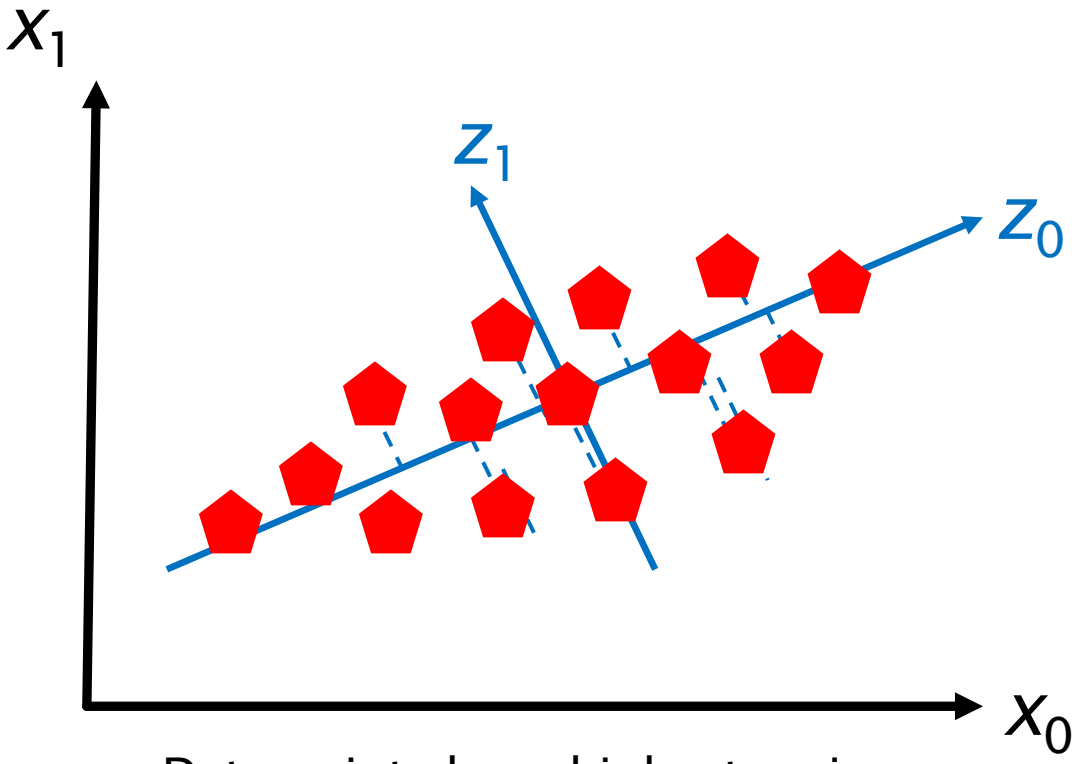
contour plot of two 2-dimensional Gaussian probability distributions.



Dimensionality Reduction

- Project high-dimensional data set into appropriately chosen space of fewer dimensions.
- Given a *d*-dimensional data set, choosing a coordinate system *z* such that a projection to the (d-1) most meaningful axes best represents the original data.

- We will discuss techniques such as PCA and SVD in the module on Feature Engineering.



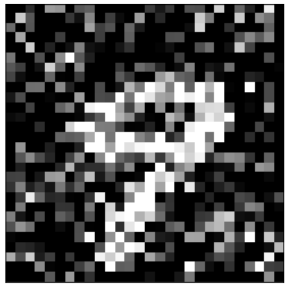
Data points have highest variance along the z_0 axis.
→ Project data points onto z_0 axis.
→ 2D data becomes 1D.

Autoencoder Artificial Neural Networks

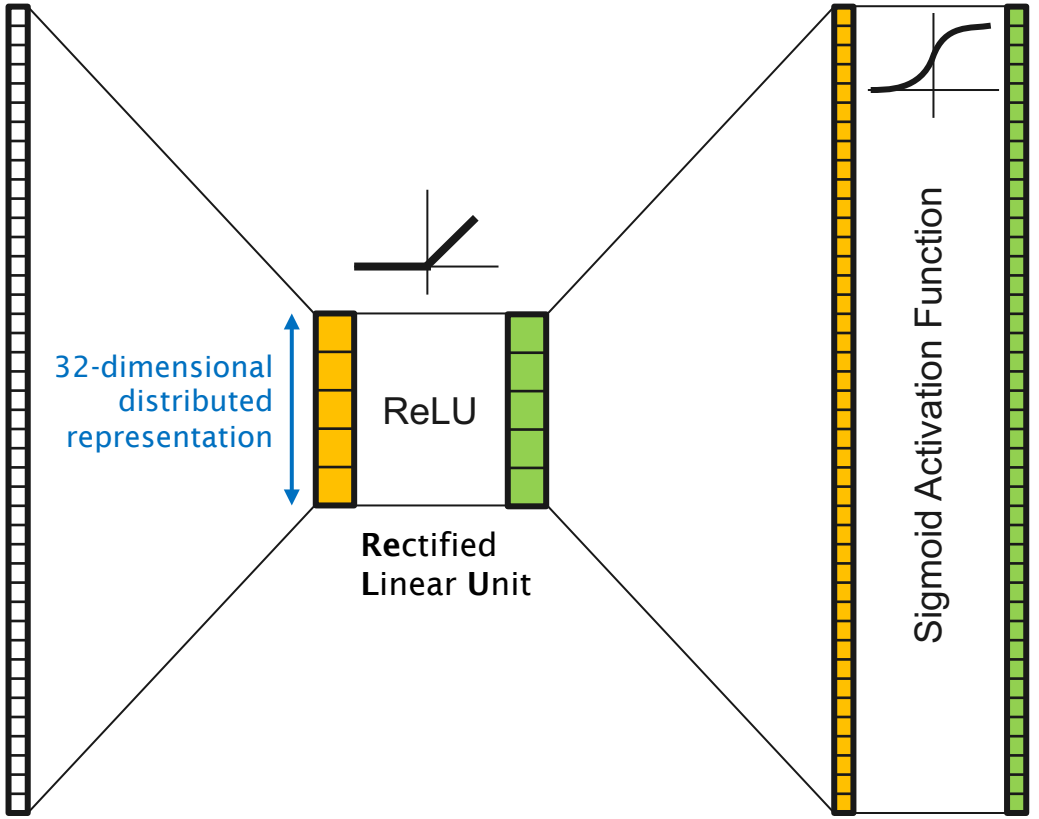
Learns compressed representation that faithfully reproduces the input (e.g., an image) at the output. Here, the autoencoder is trained on images of handwritten digits (28x28, MNIST dataset).

clean input image

Noise Removal



noisy input image



→ training = learning 50,240 weights

reproduced image at output

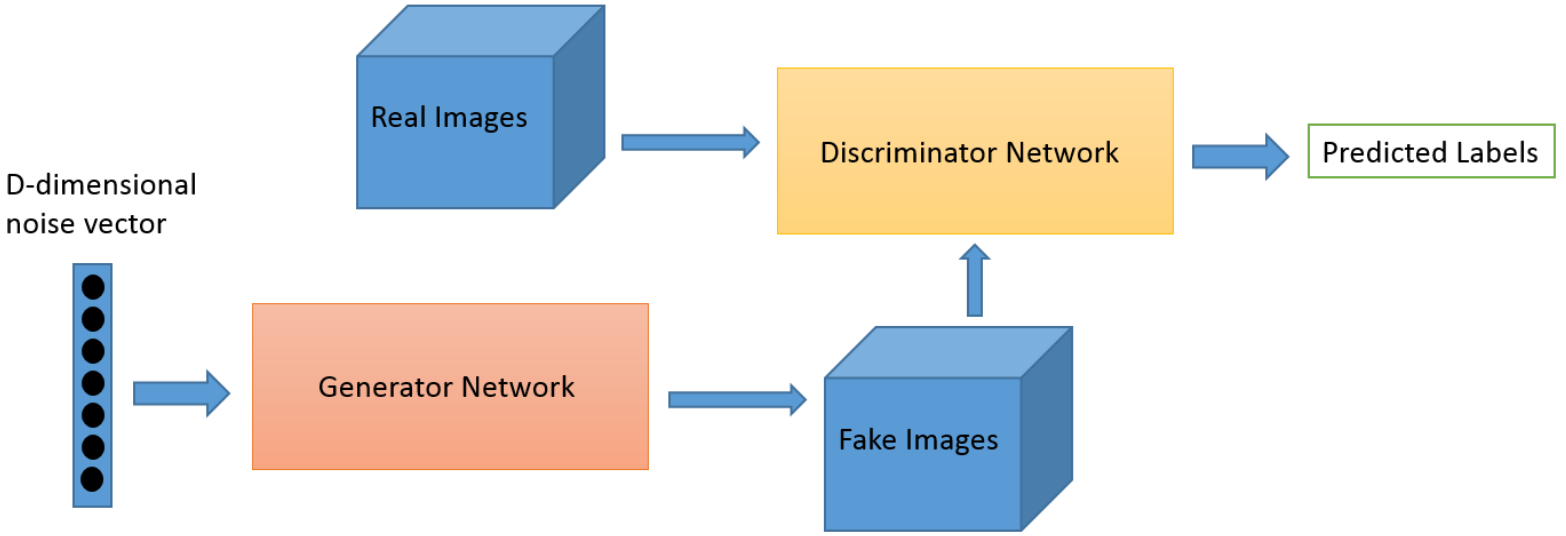
reproduced image from noisy input image

Autoencoder learns interesting features from digit images.

This allows it to remove noise in input images.

Generative Adversarial Networks

- **Goal**: Learn the probability distribution of the training set.
- Once probability distribution is learned, sample from this distribution.
  → Generates "fake" data that is similar like training data!
- **Approach**: Two Neural Networks working against each other (adversarial)
  - Generator Network: Learns to generate "fake" data
  - Discriminator Network: Learns to distinguish "fake" data from real data (from training set).



Goodfellow et al.: *Generative Adversarial Networks*, arXiv:1406.2661, 2014.

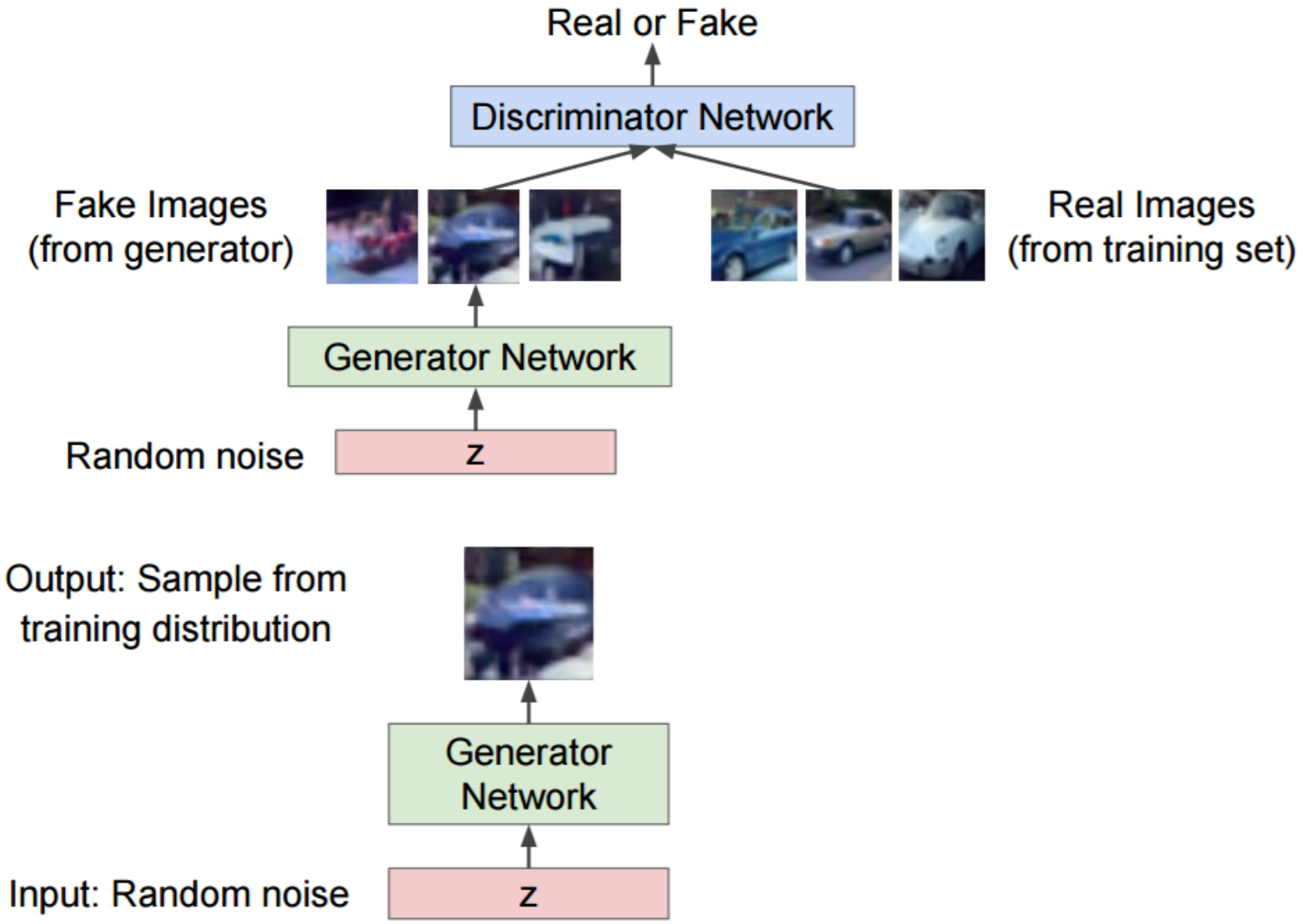
Generative Adversarial Networks

Training

- Feed discriminator fake and real data (labeled fake or real)
- Generator receives random vector (noise) at input
- Generator tries to produce better “fake” data
- Discriminator tries to discover “fake” images.
- If all goes well, generator and discriminator reach a Nash Equilibrium.

Generation of Data

- Supply random noise to input of generator network → Generator network will produce “fake” data according to learned distribution.



Generative Models

- Applications
  - Generate missing data (occluded parts in an image)
  - Generate artificial images

Generated images using the images of the volcano class in the ImageNet data set.

volcano

Nguyen, Clune, Bengio, Dosovitskiy, Yosinski: Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space, arXiv:1612.00005, 2016.

Clustering Data

Overview

Applications

- Segmentation of the customer base into different groups. Use different marketing strategies for different groups.
  - Automatically cluster customers by similarity.
  - Challenge: Can you assign a semantic meaning to inferred groups, e.g., “this groups represents males between the ages of 30 and 40 interested in soccer”?
- Anomaly Detection:
  - Cluster known “good” or ”mostly good” data points.
  - If new data point arrives, measure distance to cluster centers. If new data point is not near any cluster, the point is classified as an outlier, possibly indicating suspicious activity.

Clustering

- Used to analyze data and find clusters within the data, also called cluster analysis.

- Clustering is the process of organizing data points from a set $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ into subsets, called clusters, $C_1, C_2, \ldots, C_m$ such that $C_1 \cup C_2, \cup \cdots \cup C_m = X$.
- Elements in a cluster are similar to each other.
- Use some kind of distance metric (measure) to identify clusters
  - Distance measure determines the tightness of the cluster (intra-cluster measure)
    → Minimize
  - Connectivity measure between clusters (inter-cluster measure, linkage)
    → Maximize

Types of Clustering

- **Hard Clustering**
  - An element belongs to exactly one cluster
  - Formally, $C_i \cap C_j = \varnothing \quad \forall\, i \neq j$
  - Hence, Hard Clustering = Set Partitioning using a distance metric.
  - Methods: k-Means, k-Mediods

- **Soft Clustering**
  - Fuzzy membership: An element only corresponds to a cluster to a certain degree (confidence).
  - An element can belong more than one cluster with non-zero degree.
  - Methods: Gaussian Mixture Models (EM Algorithm)

Measuring Similarity between Data Points

Distance metric D(**u**, **v**) between points **u** and **v**

- Euclidean Distance (L_2 norm):

$$\|\mathbf{v}-\mathbf{u}\|_2^2 = \sum_{k=1}^{d}(\mathbf{v}_k-\mathbf{u}_k)^2$$

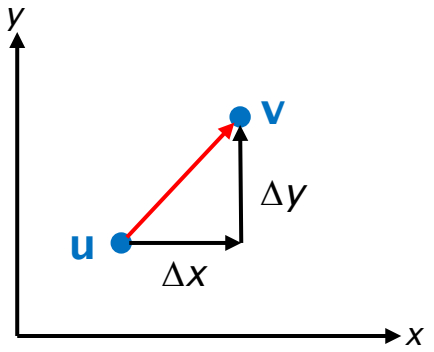
- Manhattan Distance (L_1 norm):

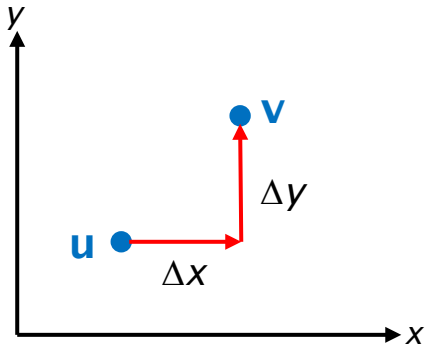
$$\|\mathbf{v}-\mathbf{u}\|_1 = \sum_{k=1}^{d}|\mathbf{v}_k-\mathbf{u}_k|$$

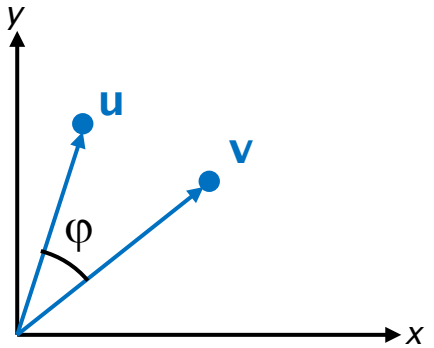
- Cosine Similarity (normalized dot product):

$$\mathbf{v}^{\mathrm{T}}\cdot\mathbf{u} = \|\mathbf{v}\|_2\|\mathbf{u}\|_2\cos\varphi$$

$$\cos\varphi = \frac{\mathbf{v}^{\mathrm{T}}\cdot\mathbf{u}}{\|\mathbf{v}\|_2\|\mathbf{u}\|_2}$$







Generalization: Minkowski Distance

- Manhattan (L_1), Euclidean (L_2) metrics can be generalized to the Minkowski Distance.

$$D(\mathbf{u}, \mathbf{v}) = \left(\sum_{i=1}^{d} |u_i - v_i|^p \right)^{1/p}$$

- p = 1: Manhattan Distance (L_1 norm)

$$D(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{d} |u_i - v_i|$$

- p = 2: Euclidean Distance (L_2 norm)

$$D(\mathbf{u}, \mathbf{v}) = \left(\sum_{i=1}^{d} |u_i - v_i|^2 \right)^{1/2} = \left(\sum_{i=1}^{d} (u_i - v_i)^2 \right)^{1/2}$$

- p = ∞: Chebyshev Distance (L_∞ norm)

$$D(\mathbf{u}, \mathbf{v}) = \lim_{p \to \infty} \left(\sum_{i=1}^{d} |u_i - v_i|^2 \right)^{1/p} = \max_{i=1 \ldots d} |u_i - v_i|$$

Clustering with K-Means

k-Means Algorithm

- Most popular clustering algorithm.
- Uses Euclidian Distance Metric.
- Assume that the number of clusters *k* is known beforehand (disadvantage).

- **Training Phase**: Partition data into *k* subsets by applying a distance metric on data attributes.

- **Inference Phase** (optional): after cluster centers are determined, classify your data to nearest centroid.

K-Means Algorithm

- **Goal**: Cluster data into *k* groups of equal variance while minimizing the within-cluster sum of squares.

- Let $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$ be the *n* *d*-dimensional data points to be clustered.
- Clustering: partition the *n* data points into k disjoint partition sets P_1, ..., P_k such that the within-cluster sum of squares between the data points $\mathbf{x}_j$ and the cluster center (centroid) is $\boldsymbol{\mu}_i$ minimized.
- Choose cluster centroids $\boldsymbol{\mu}_i$ such that $\boldsymbol{\mu}_i$:

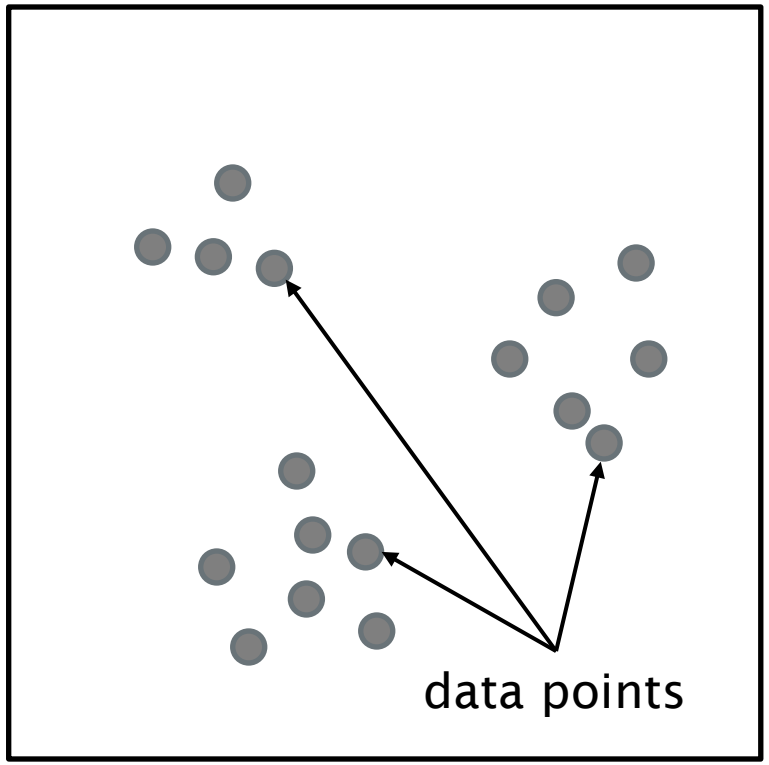
$$\underbrace{\sum_{i=1}^{k} \sum_{\mathbf{x} \in P_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2}_{\text{objective function: within-cluster sum of squares (also called inertia).}} \rightarrow \min \quad \text{and} \quad P_1 \cup P_2 \cup \cdots \cup P_k = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \quad \text{and} \quad P_i \cap P_j = \emptyset \quad \forall i \neq j$$

- Objective function minimizes variance:

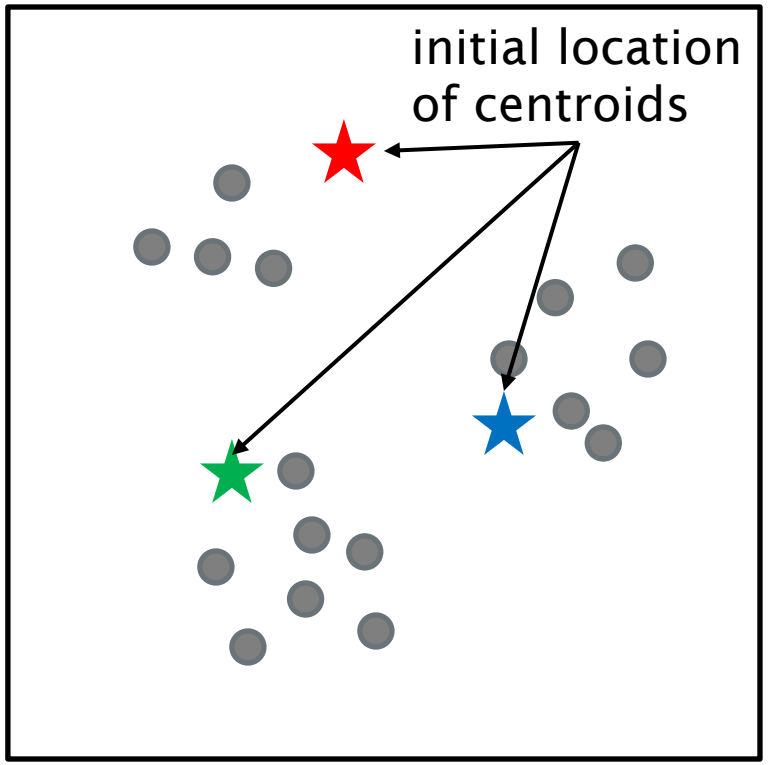
$$\sum_{i=1}^{k} \underbrace{\sum_{\mathbf{x} \in P_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2}_{\frac{|P_i|}{|P_i|} \sum_{\mathbf{x}} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2} = \sum_{i=1}^{k} |P_i| \mathrm{Var}(P_i)$$

k-Means Algorithm Illustration (1)

Illustration in 2D

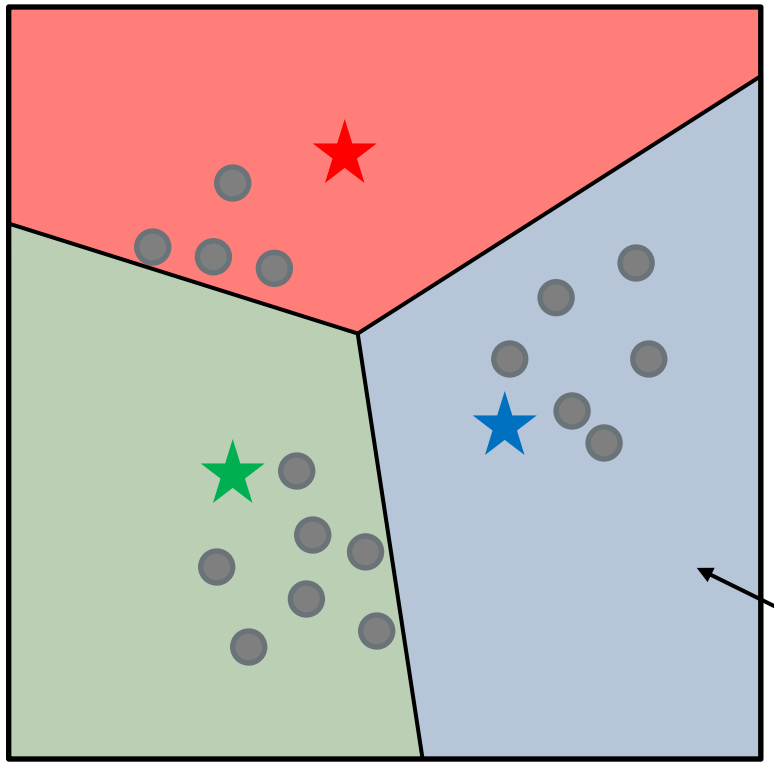


Step 0



Step 1: Initial location of k=3 centroids, chosen by some by some method.

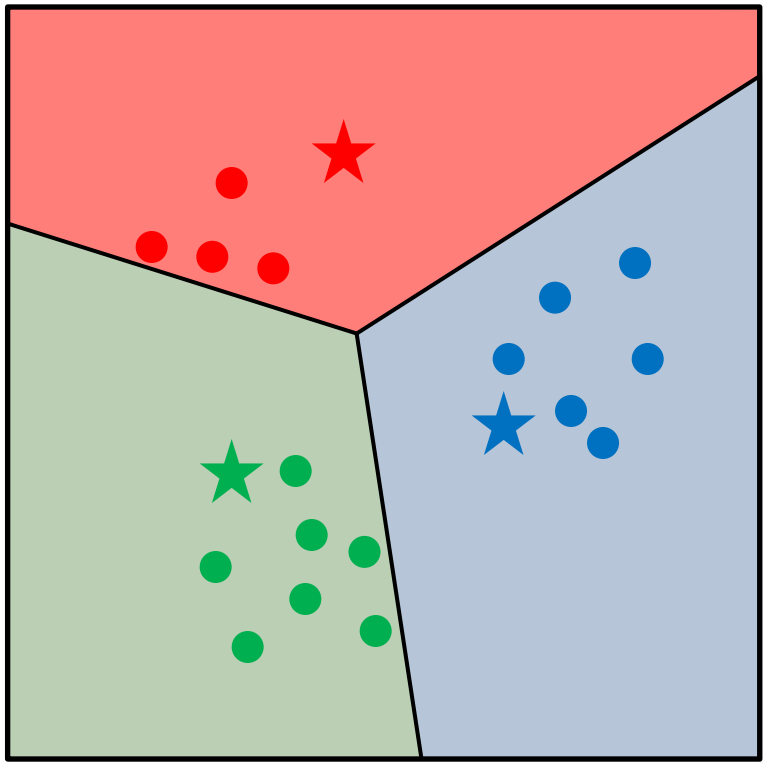
k-Means Algorithm Illustration (2)



perpendicular bisectors (Voronoi Tessellation)

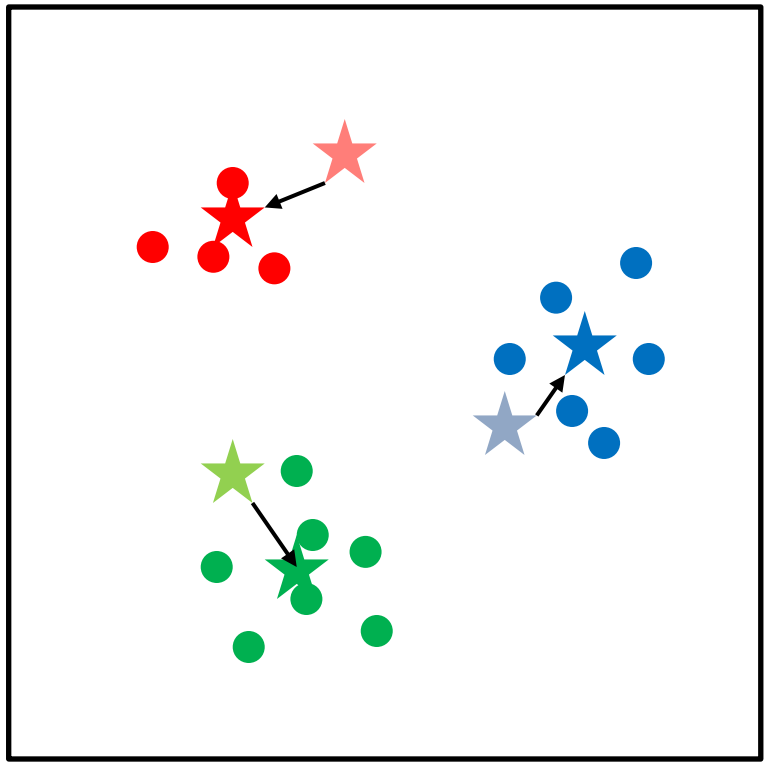
all points in blue region are closest to the blue centroid ★.

Step 2a: Tessellation of data domain into regions using the centroid location.

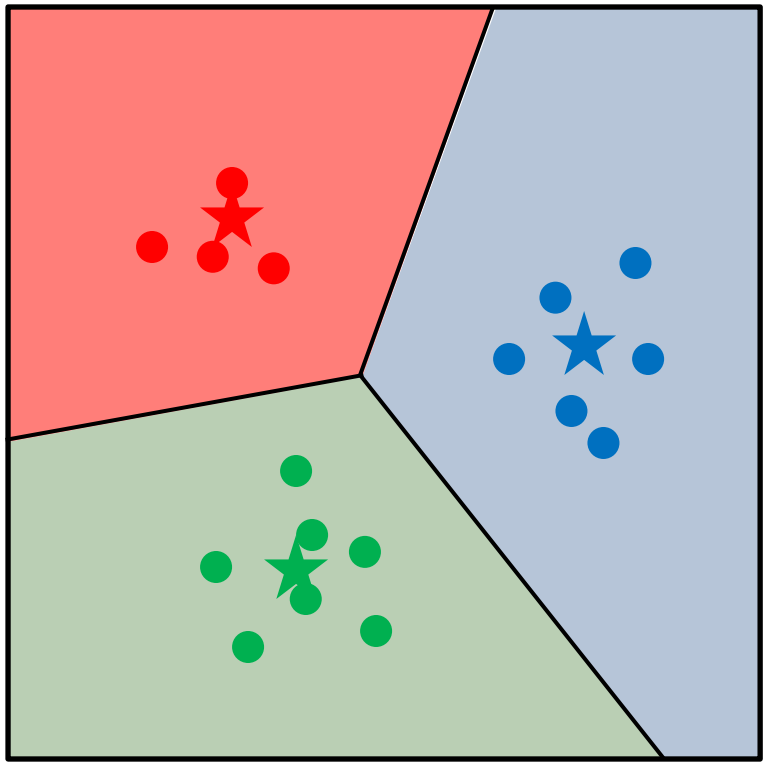


Step 2b: Assign each data point to its closest centroid.

k-Means Algorithm Illustration (3)



Step 3: Update centroid locations by computing the mean of all data points in each cluster.



Repeat steps 2 and 3 until convergence is reached, i.e., cluster assignments do not change anymore.

Classification of new Data Points

After training, the learned cluster locations can be used to classify new data points.

The k-Means Algorithm (Lloyd's Algorithm)

Select *k* arbitrary data points as centroids and the iteratively perform the following steps:

- **Centers to Clusters**: Assign each data point to the cluster correspond to its nearest centroid (ties are broken arbitrarily)

- **Clusters to Centers**: After the assignment of data points to *k* clusters, compute centers as cluster's center of gravity, i.e. mean of all data points currently assigned to the cluster.

Convergence:

- A data point is assigned to a new cluster if it is closer than to the previous center.
- The intra-cluster variance is reduced when updating the centroid locations.

Challenges in the k-Means Algorithm

The problem of optimally partitioning the data is NP-hard.

K-Means is a heuristic algorithm. The Algorithm may converge to a local optimum rather than a global optimum.

K-means looks deceptively simple but a good implementation needs to solve these challenges.

- Initial choice of centroid location:
  - Choose initial locations randomly in data domain.
  - Set initial locations from position of *k* randomly selected data points.
  - k-means++ algorithm: Choose k locations from randomly selected data points but choose new so that they are as far away from already selected centers (initial clusters spread out maximally)
- Empty Clusters
  - There is one or more cluster that does not contain any data point.
    The centroid is too far away to attract any point. → centroid location will not change during update.
  - Workaround: Repeat k-Means multiple times each time with different starting point

Arthur, Vassilvitskii: *k-means++: the advantages of careful seeding*. Proceedings of the 18th annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics Philadelphia, PA, USA. pp. 1027–1035.

K-Means in scikit-learn

Import the KMeans class from the `sklearn.cluster` package:

```
from sklearn.cluster import KMeans
```

Use `fit()` to train k-means on dataset.

seeds pseudorandom number generator

```
k = 3
kmeans = KMeans(n_clusters=k, random_state=0).fit(data)
```

After training, the cluster centers are:

```
kmeans.cluster_centers_
```

NumPy array

```
array([[-1.42734639, 2.081 ], [ 0.36081579, 1.83154971], [-0.22904601,
0.95091411]])
```

and the *n* data labels for the data points. Label from 0 to *k*–1.

```
kmeans.labels_
```

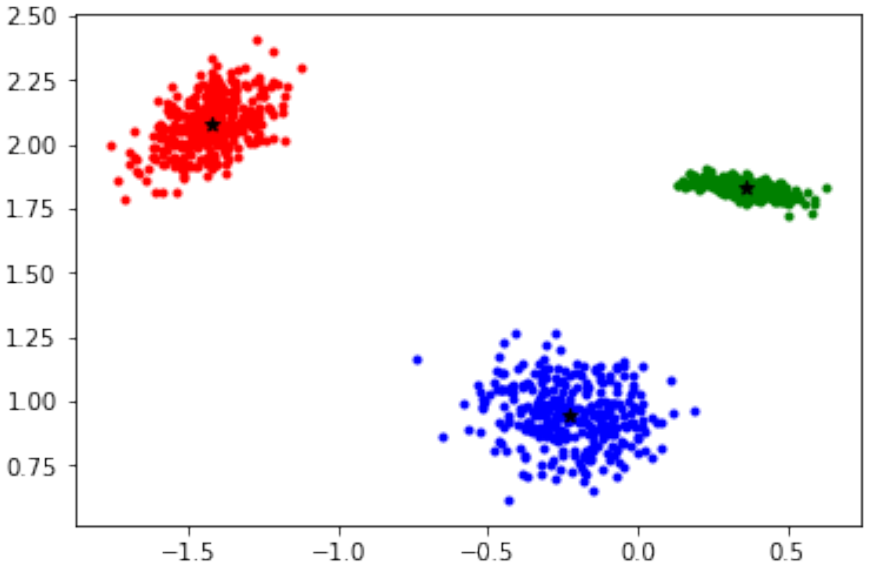
```
array([2, 2, 0, 2, 2, 2, 0, 0, 0, 0, 1, 0, 0, 2, 1, 2, 2, 0, 2, 0, 0, 1, ...
```

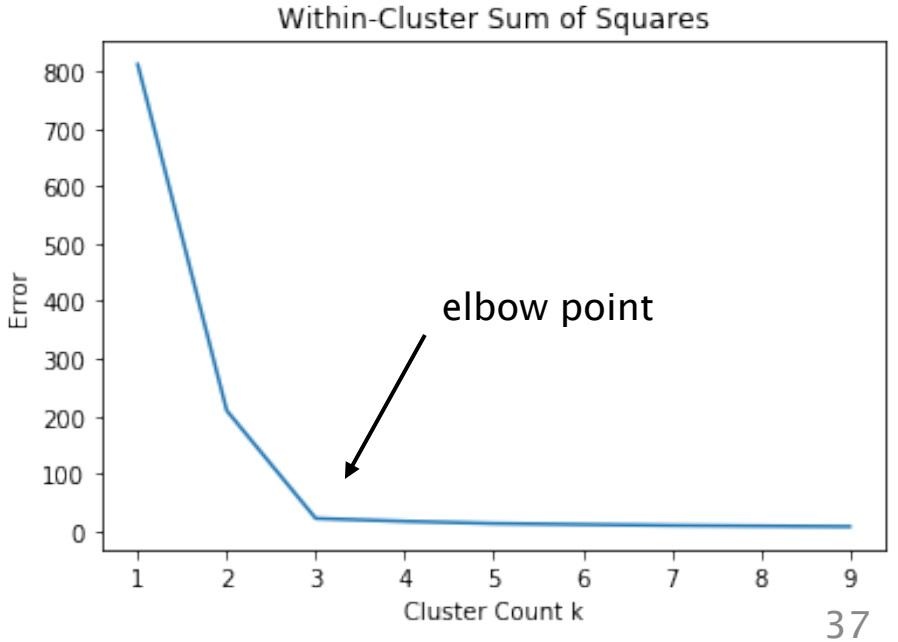
Optimal Number of Clusters k

- Increasing k reduces the within-cluster sum of squares.
  - It goes to zero if $k = n$.
- After some point, there is a diminishing return.

- *What is the optimal choice for k?*
- Note: In this simple 2D example, the best cluster count is obvious, but this is generally not the case for high-dimensional data.

- *How can we measure the quality of a clustering anyway?*



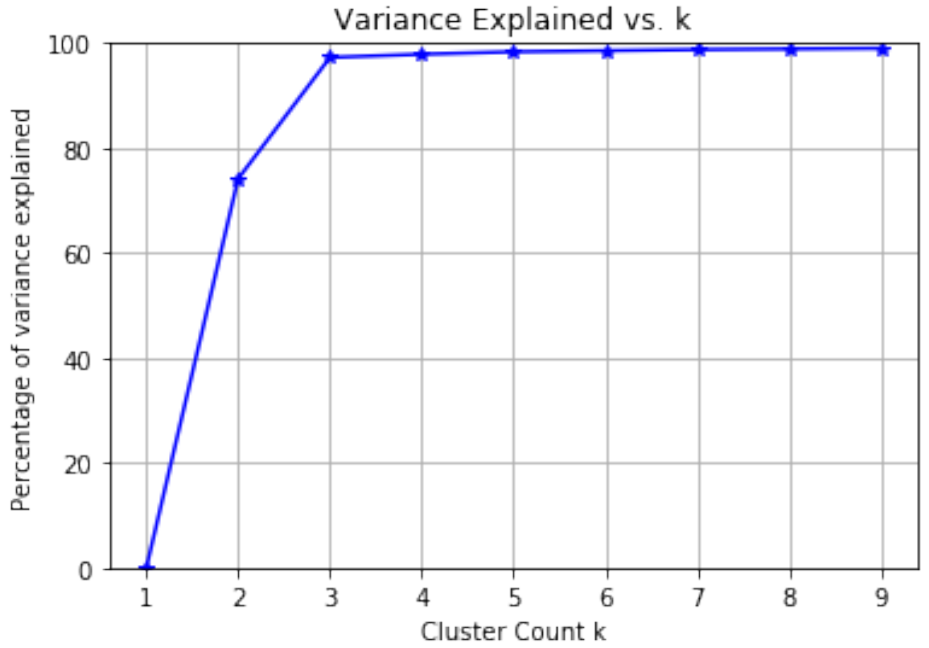


Measuring Quality of the Clustering

- The within-cluster sum of squares is not a good metric since it is not normalized, i.e., it depends on the absolute scale of data points $\mathbf{x}_1$, ..., $\mathbf{x}_n$.
- There are several metrics to evaluate the quality of a cluster assignment.

Percentage of Variance Explained: sometimes also called Elbow Method

- Adding another cluster decreases the variance, increases the explained variance.
- The percentage of variance explained is %-ratio of between-cluster variance and the total variance.
- Ideally,100% of the variance is between clusters.



Silhouette Score (1)

Silhouette Score is another metric to evaluate a cluster assignment

- The Silhouette coefficient is calculated using the mean intra-cluster distance a_i and the mean nearest-cluster distance b_i for each sample.
  a_i is the mean distance between a data point $\mathbf{x}_i$ and all other points in the same cluster.
  b_i is the mean distance between a data point $\mathbf{x}_i$ and all other points in the *next nearest cluster*.
- The Silhouette coefficient s_i for a data point $\mathbf{x}_i$ $\quad s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$
- The Silhouette score is the mean of the s_i of all data points $\mathbf{x}_i$.
- **Interpretation:**
  A score of 1.0 is the best value, -1.0 the worst score. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar.

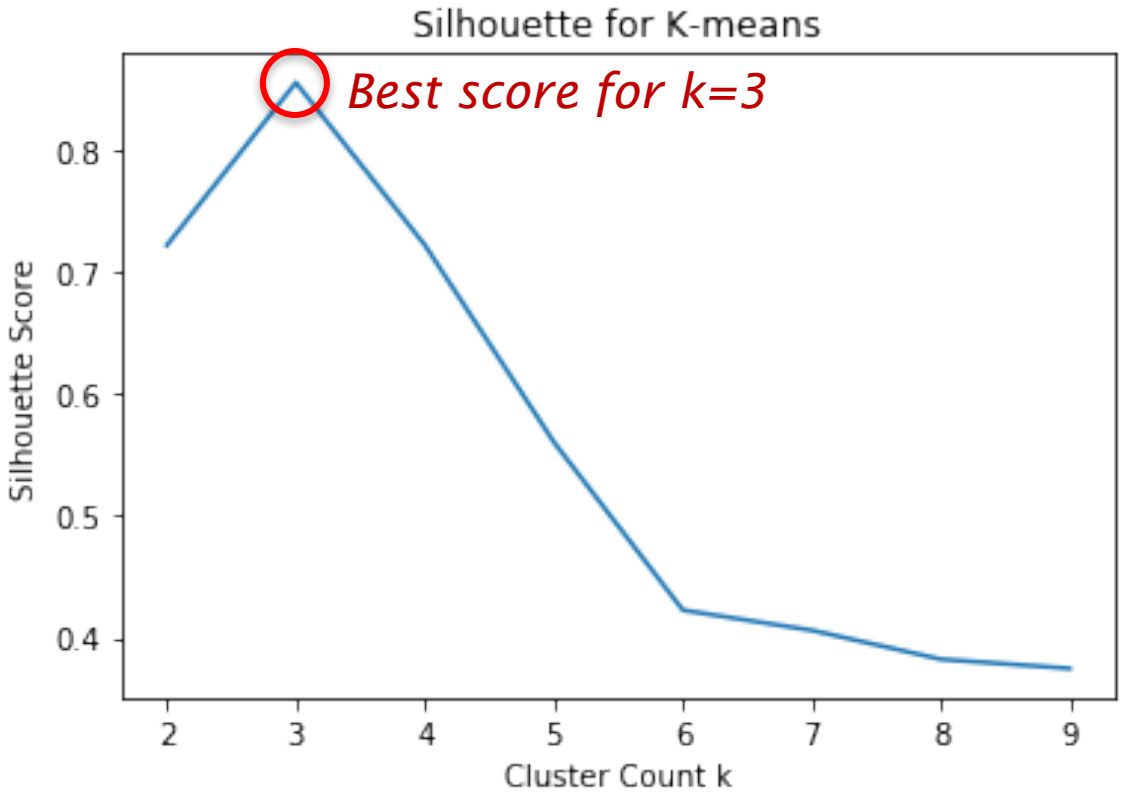
P. J. Rousseeuw: *Silhouettes: A graphical aid to the interpretation and validation of cluster analysis*. Journal of Computational and Applied Mathematics. Vol. 20, Nov. 1987. pp. 53—65.

Silhouette Score (2)

```
from sklearn.metrics import silhouette_score

s = []
ks = range(2,10)
for k in ks:
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(data)
    s.append(silhouette_score( \
       data, kmeans.labels_, \
       metric='euclidean'))

plt.plot(ks, s)
plt.ylabel('Silhouette Score')
plt.xlabel('Cluster Count k')
plt.title('Silouette for K-means')
plt.show()
```



Hartigan Rule:

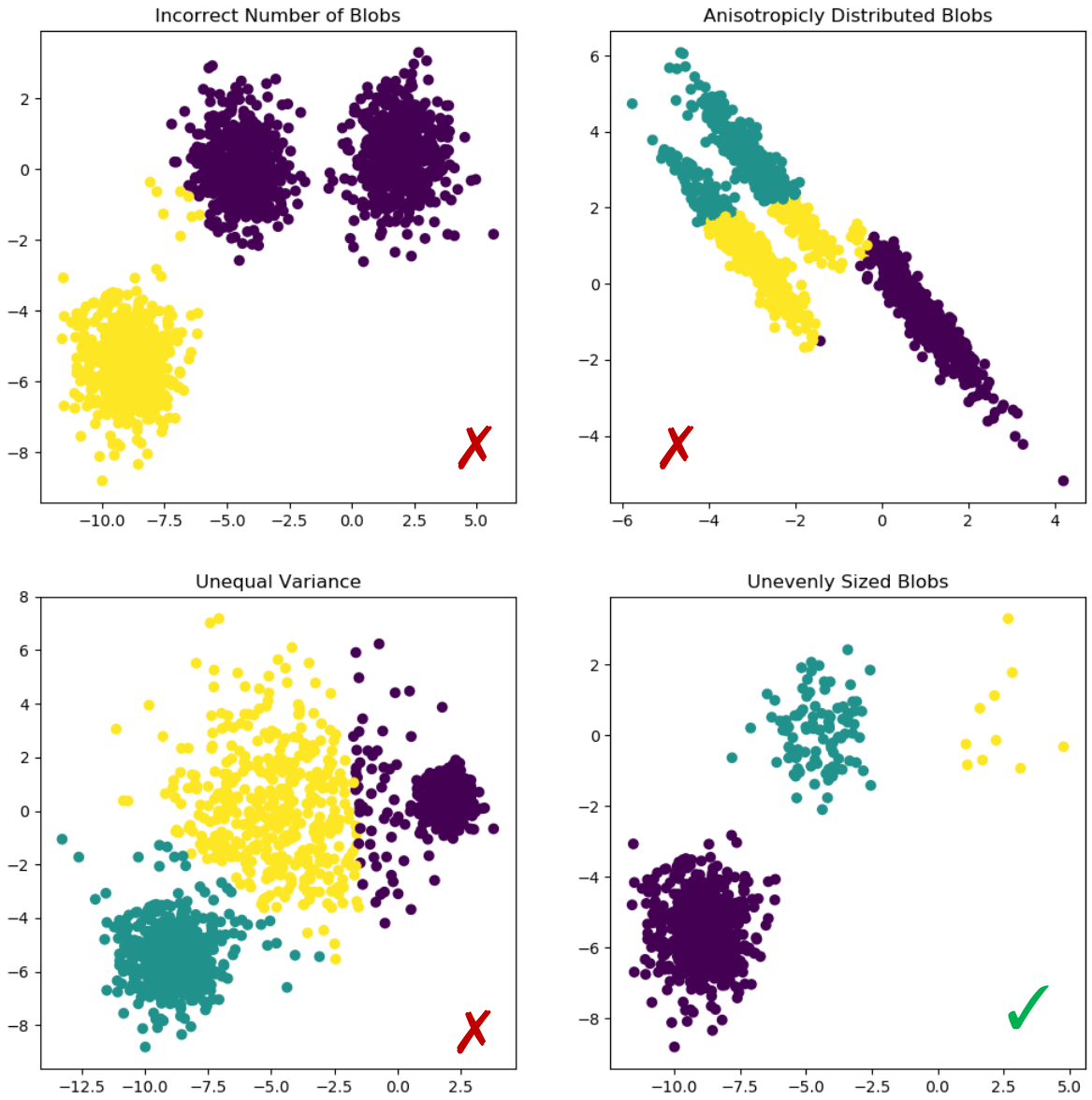
- It essentially compares the ratio of the within-cluster sum of squares (WCSS) for
  - an assignment with k clusters and → WCSS_k
  - an assignment with $k + 1$ clusters → WCSS_{k+1}
- Calculate $$H = \left(\frac{\text{WCSS}_k}{\text{WCSS}_{k+1}} - 1 \right) (n - k - 1)$$
- Rule: If H that number is greater than 10 (sometimes also 12), then it is worth using k + 1 clusters.

M. Chiang, B. Mirkin: *Intelligent Choice of the Number of Clusters in K-Means Clustering: An Experimental Study with Different Cluster Spreads*. Journal Classif (2010) 27: 3. doi:10.1007/s00357-010-9049-5
http://www.dcs.bbk.ac.uk/~mirkin/papers/00357_07-216RR_mirkin.pdf

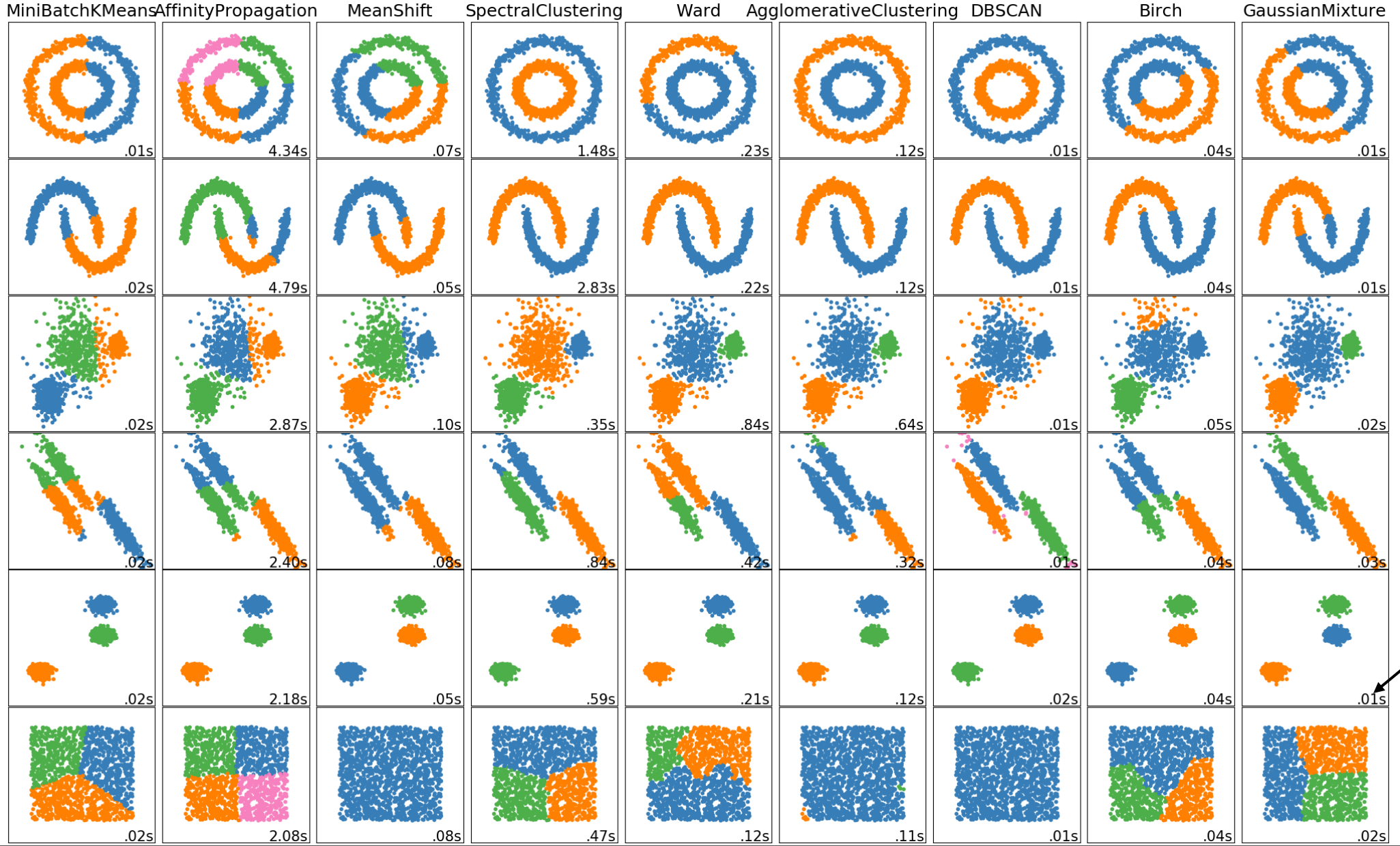
Limitations of K-Means

K-Means relies on the following assumptions:

- Clusters are convex and isotropic
- The inertia is not normalized.
- Due to the Euclidian distance as a metric, k-Means is very sensitive to outliers.



Clustering Methods available in scikit-learn



Literature k-Means

- Peter Flach: *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Chapters 3, 8, and 9. Cambridge Press. 2012.
- D. Arthur, S. Vassilvitskii: *k-means++: The advantages of careful seeding*. In Proceedings of the 18th annual ACM-SIAM symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2007.
- B. Bahmani, B. Moseley, A. Vattani, R. Kumar, S. Vassilvitskii: *Scalable K-Means++.* In Proceedings of VLDB Endowment, Vol 5, No. 7. 2012.
- P. J. Rousseeuw: *Silhouettes: A graphical aid to the interpretation and validation of cluster analysis*. Journal of Computational and Applied Mathematics. Vol. 20, Nov. 1987. pp. 53—65.
- M. Chiang, B. Mirkin: *Intelligent Choice of the Number of Clusters in K-Means Clustering: An Experimental Study with Different Cluster Spreads*. Journal Classif (2010) 27: 3. doi:10.1007/s00357-010-9049-5 http://www.dcs.bbk.ac.uk/~mirkin/papers/00357_07-216RR_mirkin.pdf

Exercise 1: k-Means

In groups of two, 60 min

Solve the Exercise 1 in the notebook Exercise01_kMeans.ipynb.
You will be clustering the **fleet data from delivery drivers**.

1. Load the dataset from tab-separated file using the method of your choice (on-board Python, NumPy, Pandas, etc.)
2. Visualize the data, find a reasonable number of clusters from the visualization and systematically.
3. Cluster the data

Case Study 1: Customer Segmentation with k-Means

Input: Purchase data from a wine shop and number of product offers.

Cluster customers based on their purchase history into groups.

Data set from:

John W. Foreman. *Data Smart: Using Data Science to Transform Information into Insight*. John Wiley & Sons. 2013.

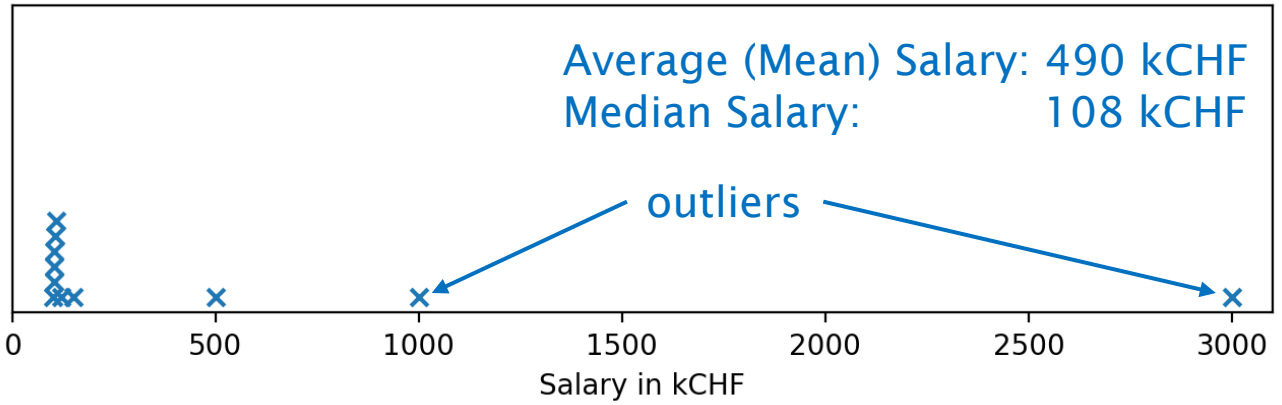
Clustering with k-Medoids/PAM

PAM = Partition Around Medoids

K-Medoids/PAM vs. K-Means

- K-Means does not work well with categorical data
- K-Means is susceptible to outliers (due to the L2 metric, Euclidean distance)

- **Analogy**: Average vs. Median

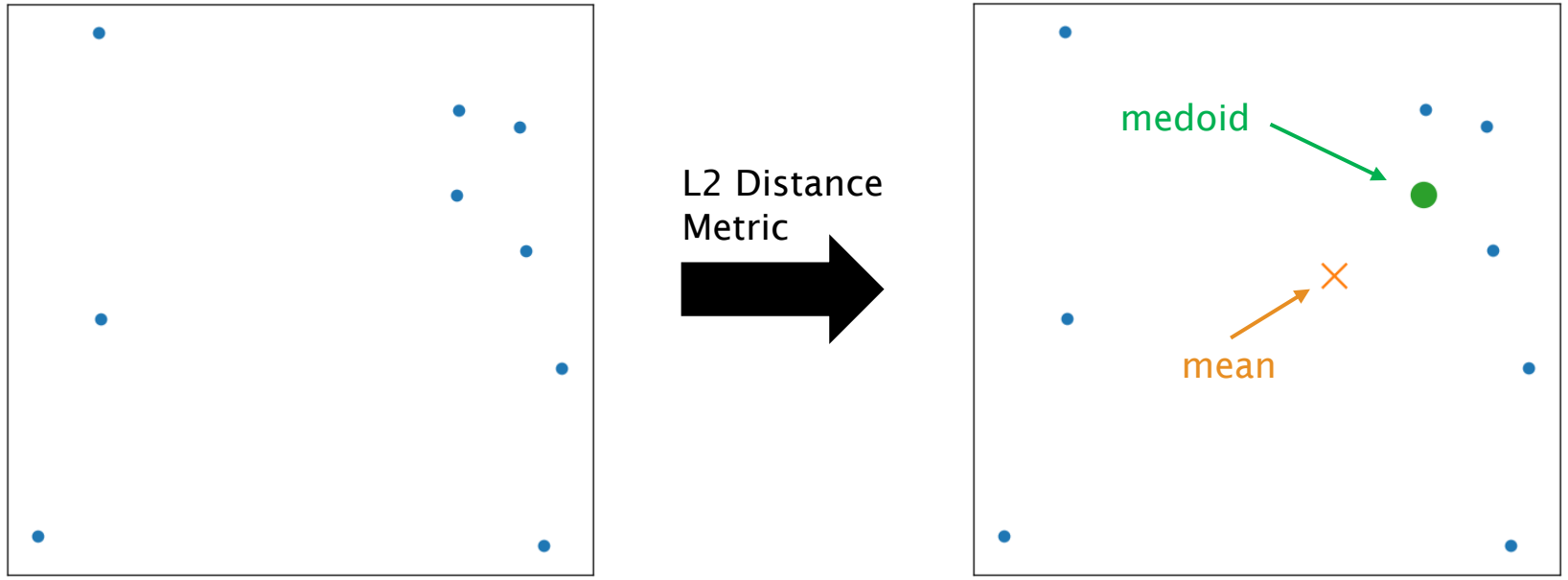


- Informally: “K-Medoids is to K-Means what Median is to Average”
- Partition Around Medoids PAM is an specific algorithm for the k-Mediods

Medoids

- K-Medoids: Instead of taking the mean (average) location of the data points in a cluster as the centroid, use the medoid instead.
- Medoid = Most centrally located point in a cluster.
  - The value of the medoid actually appears in the data set. Same holds for the median unlike for the mean[1].
- **Definition:** Medoid m (x_m is its location)

$$\mathbf{x}_m = \underset{\mathbf{y} \in \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}}{\arg\min} \sum_{i=1}^{n} D(\mathbf{x}_i, \mathbf{y})$$



1) Here, we consider the median to be either the *lower* or the *upper median* if the data set has an even number of elements.

PAM: Typical k-Medoids Algorithm

- K-Medoids Clustering Algorithm:
  1. Select k points as the initial representative data point, i.e., as the initial k medoids
  2. **Assign data points to medoids**: Assign each point to the cluster with the closest medoid
  3. Randomly select a non-presentative data point *i*, i.e., a non-medoid. Let *m* be the medoid to which the data point is currently assigned.
  4. Compute the total cost *J* of swapping the medoid *m* with *i*.
  5. If $J < 0$ the then swap m with *i*, i.e., *i* becomes the new medoid.
  6. Go back to step 2 until convergence.

Update medoids

K-Modes/k-Prototypes

K-Means with Categorical Values

Categorical Values for k-Means/k-Medoids

- Categorical values appear often in data sets, for example:
  - Gender column: FEMALE, MALE
  - Color column: RED, GREEN, BLUE, YELLOW
  - Binary columns: TRUE, FALSE

- Problem with categorical values is that there exists (typically) no order between the values and no arithmetic is possible.
- This is even the case if we encode a categorical value as an integer. For example:
  - RED → 0, GREEN → 1, BLUE → 2, YELLOW → 3
  - Say, K-means determines centroid at 1.32. What color does this correspond to? GREEN or some color between GREEN and BLUE?
  - Say, K-medoids measures distance between GREEN and YELLOW? What does distance 2 correspond to? Would it be larger than between RED and GREEN?

Workarounds to Support Categorical Values

- Convert categorical value into one-hot encoding (later in Feature Engineering).
  - **Example**:
    One 4-Color category (RED, GREEN, BLUE, YELLOW) turns into four new attributes.

	attr_red	attr_green	attr_blue	attr_yellow
RED	1	0	0	0
GREEN	0	1	0	0
BLUE	0	0	1	0
YELLOW	0	0	0	1

only one attribute is 1 (hot) per row.

  - This increases the dimensionality of feature space!
    We still have to deal with the "curse of dimensionality".
    - Consider a hypercube in d dimensional space. Its distance from the center to the corner a side-length $2r$ is $r\sqrt{d}$. As d increases, the distance increase with the square-root of d. → For large d all points points are "far away".
  - **Suggestion**: perform a dimensionality reduction (e.g., PCA, discussed later) first and cluster on subspace.

Handling Categorical Values with k-Modes/k-Prototypes

- k-Modes: Introduces dissimilarity measure, means → modes
- k-Prototype: Combines k-Mode with k-Means for categorical and number attributes.

- **Dissimilarity Measure** for two objects *A* and *B* that consist of *m* categorical attributes:

$$D(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{m} \delta(a_i, b_i)$$

← simply counts the number of mismatches (we can test categorical values for equality).

$$\delta(a_i, b_i) = \begin{cases} 1 & a_i \neq b_i \\ 0 & a_i = b_i \end{cases}$$

- **Mode**: A mode of categorical objects $X = \{ \mathbf{x}_1, \ldots, \mathbf{x}_n \}$, each described by categorical attributes $a_1, \ldots, a_m$, is a vector $\mathbf{q} = (q_1, \ldots, q_m)^T$ that minimizes

$$D(X, \mathbf{q}) = \sum_{i=1}^{n} D(\mathbf{x}_i, \mathbf{q}) \quad .$$

  Note: **q** does not necessarily have to be an element of *X*, neither does it have to be unique.
- A mode **q** can be found be found by measuring the frequency of the categories in the attributes $a_1, \ldots a_m$. Details and proof omitted, see paper [1].

[1] Zhexue Huang: *Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values*. Data Mining and Knowledge Discovery 2, pp. 283--304, 1998.

k-Modes/k-Prototype in Python – the kmodes Package

- Unfortunately, scikit-learn does not contain an implementation of k-Modes or k-Prototype.
  → Install the kmodes package: (see https://github.com/nicodv/kmodes)

```
pip install -U kmodes
```

- **Example:**

```
from kmodes.kprototypes import KPrototypes

kproto = KPrototypes(n_clusters=4, init='Cao', verbose=2)
clusters = kproto.fit_predict(X, categorical=[1, 2])
```

Initialization of centroids based on Cao et al.

a NumPy.ndarray containing the cluster labels (cluster IDs for each data point)

Data set

Specification of the categorical columns in X in a list.

```
kproto.cluster_centroids_
```

Determined cluster centroids

Fuyuan Cao, Jiye Liang, Liang Bai: *A new initialization method for categorical data clustering*. Expert Systems with Applications. Vol 36. 2009.

Literature k-Modes/k-Prototypes

- Zhexue Huang: *Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values*. Data Mining and Knowledge Discovery 2, pp. 283--304, 1998.
- Fuyuan Cao, Jiye Liang, Liang Bai: *A new initialization method for categorical data clustering*. Expert Systems with Applications. Vol 36. 2009.

Exercise 2: k-Prototypes

In groups of two, 60 min

Solve the Exercise 2 in the notebook Exercise02_kPrototypes.
You will be clustering a larger set of **stock data**.

1. Load the dataset from tab-separated file using the method of your choice (on-board Python, NumPy, Pandas, etc.)
2. Cluster the data

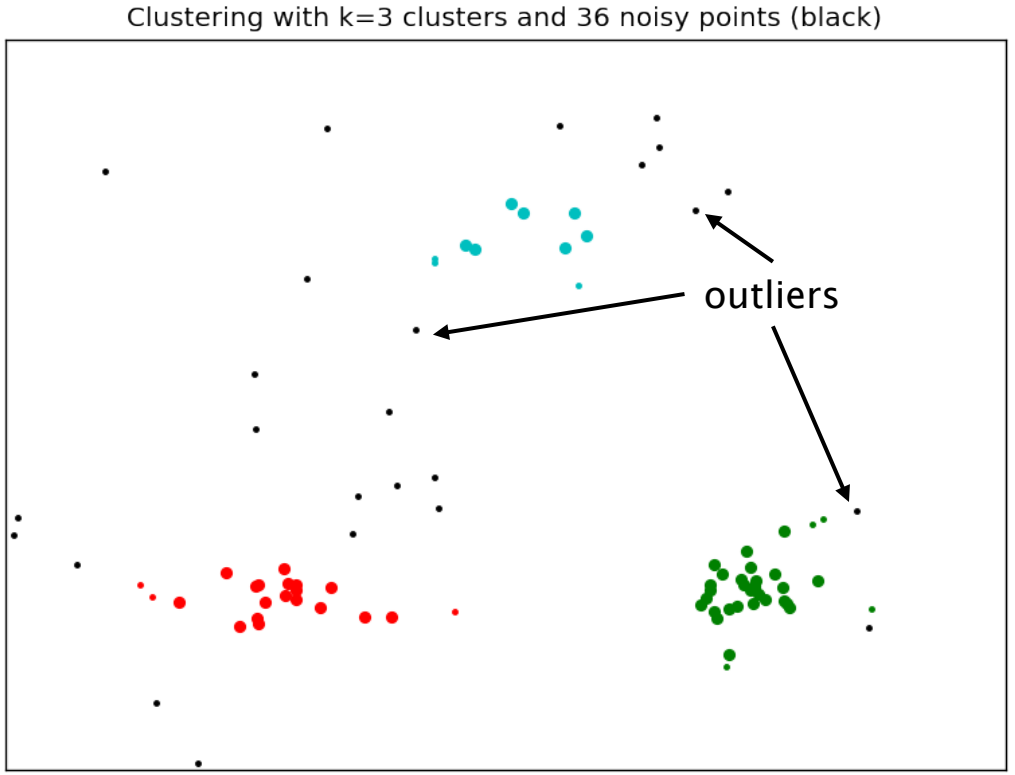
Density-based Clustering

DBSCAN

Idea of Density-Clustering

- So far, we have assigned every data point to a cluster, even points that look like outliers.
- We have not considered the how many points form a cluster and whether points in a cluster is close enough together that they represent a “reasonable” region.
- In density clustering, these two issues are addressed.

- **Density-based clustering**: Groups together points that are closely packed together.
  - Cluster is a dense region, points in it have many close neighbors.
  - Points that do not have many neighbors (an unpopulated range of the feature space) are considered outliers.
  - Outliers are not part of any cluster.



DBSCAN

- DBSCAN is the most popular density-based clustering method.
- The algorithm first classifies data points to either
  - member of a cluster as, such called, core points
  - member of a cluster as, such called, reachable points
  - outlier points.
- The cluster members are assigned to a cluster based on a distance metric.

- DBSCAN has two important parameters:
  - The maximum distance ε (in some metric) between to data points that are considered to be connected, i.e., they are neighbors.
  - The minimum number of points *m* a point *p* needs to be connected to, in order to be a core point.

Ester, Kriegel, Sander, Xu, Sismoudis. *A density-based algorithm for discovering clusters in large spatial databases with noise*. In Proceedings of the 2nd International Conference on Knowledge and Data Mining (KDD). 1996.

DBSCAN – Definitions

- A point p is a core point iff at least m points are within distance ε of it (including p).
- Points that are within radius ε from p are called neighbors of p.

- Directly reachable: A point q is directly reachable from p if point q is a neighbor of p and p is a core point.

- Reachable: A point q is reachable from p_0 if there is a path p_0, p_1, ..., p_j, q, where each p_{i+1} is directly reachable from p_i. The p_i points i=0..j are thus core points. q does not have to be a core point but must be a neighbor of last core point p_j in the path.

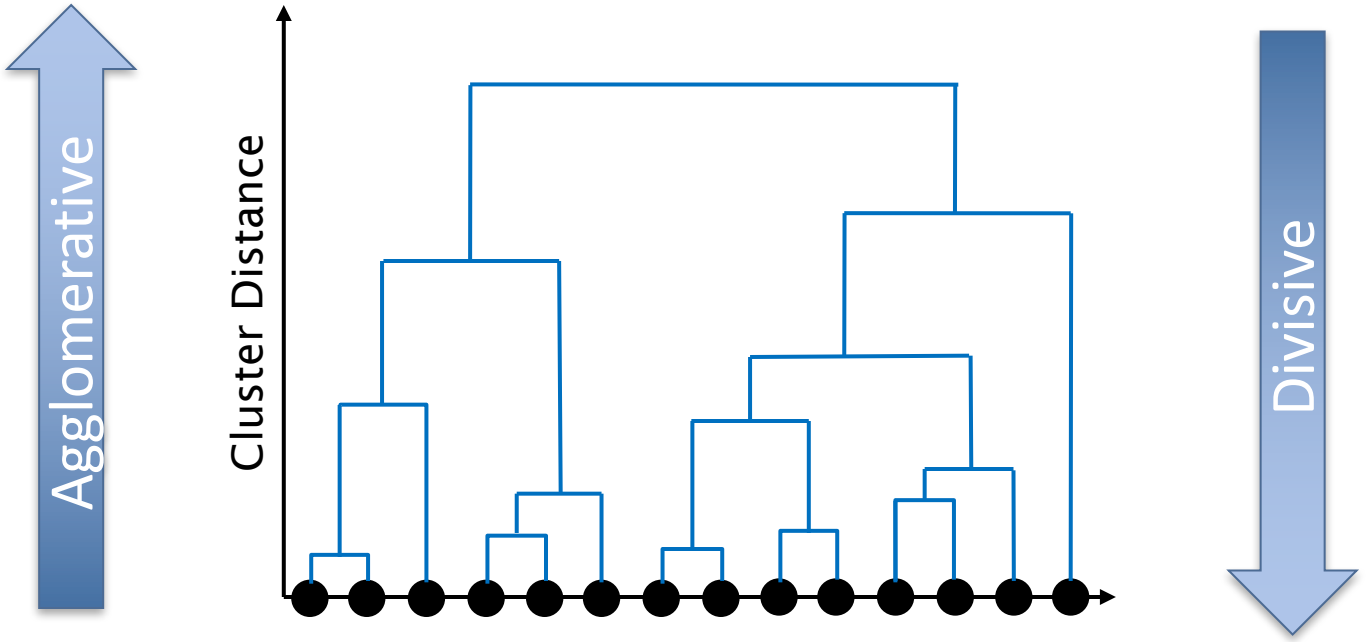
- If q is a neighbor of p_j but not a core point, it is called a non-core point.
- Outliers: All points not reachable from any other point are outliers.

Hierarchical Clustering

Grow clusters

Hierarchical Clustering

- Hierarchical clustering has a predefined order top-to-bottom or bottom-to-top.
- Similar to nested directories in a file system.
- Two types of hierarchical clustering
  - Agglomerative: Bottom up → Merge most similar clusters
  - Divisive: Top down → Partition cluster into the two least similar cluster.



Agglomerative Hierarchical Clustering

- **Required**: Distance metric between clusters, called linkage.
- Step 1: Assign each data point to its own data cluster.
- Step 2: The compute similarity between each of clusters
- Step 3: Join the two join the two most similar clusters
- Step 4: Go back to Step 2 until there is only one single cluster left.

Peusdocode:

```
Input:  Data points X = {x_1, ..., x_n}
        Linkage function L(c_i, c_j) to measure distance between clusters c_i and c_j
Output: C a set of sets of sets of ...
for i = 1 to n   -- create 1-element clusters
    c_i = { x_i }
end
C = {c_1, c_2, ..., c_n}   -- repeat until only one single cluster left
while |C| > 1 do
    (cmin_1, cmin_2) = argmin L(c_i, c_j) for all c_i, c_j in C
    remove cmin_1 and cmin_2 from C
    add new cluster {cmin_1, cmin_2} to C
end
```

Linkage

- **Linkage**: Distance between two clusters (sets of data points)
- Distance metric: Distance between two data points

- **Single Linkage**
  Shortest distance between two points in each cluster.

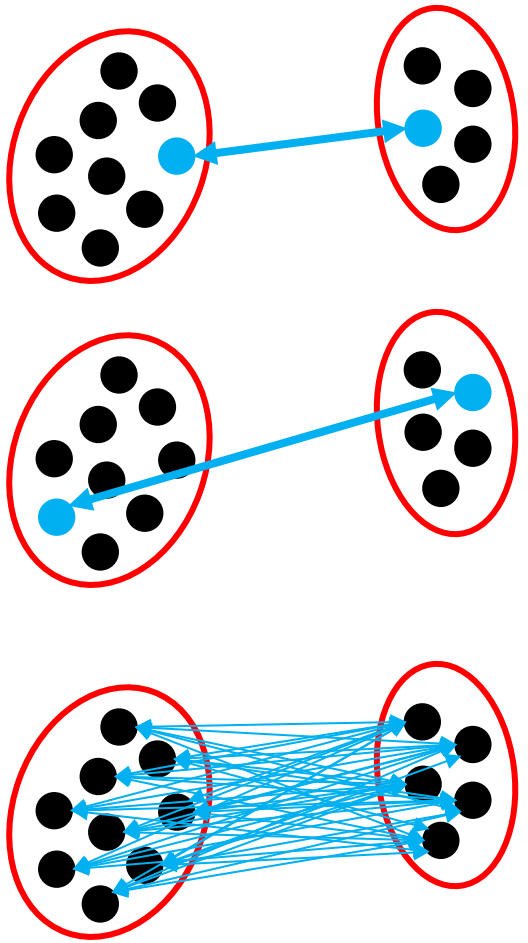
$$L(s,t) = \min\left(D(x_i^{(s)}, y_j^{(t)}\right)$$

- **Complete Linkage**
  Longest distance between two points in each cluster.

$$L(s,t) = \max\left(D(x_i^{(s)}, y_j^{(t)}\right)$$

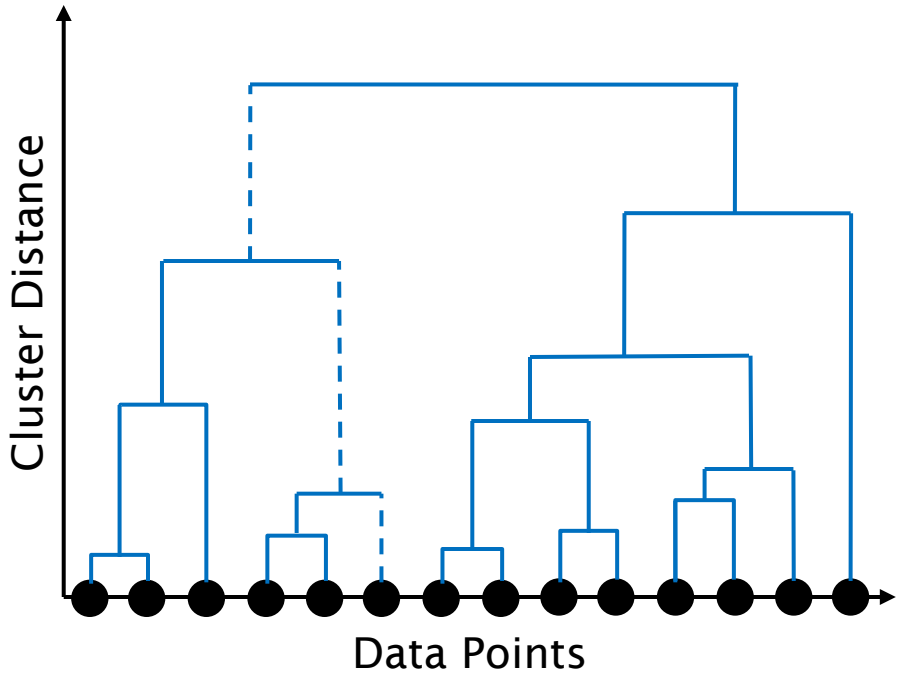
- **Average Linkage**
  Average distance between each point in one cluster and every point in the other cluster.

$$L(s,t) = \frac{1}{n_s n_t} \sum_{i=1}^{n_s} \sum_{j=1}^{n_t} D(x_i^{(s)}, y_j^{(t)})$$



Dendrogram (1)

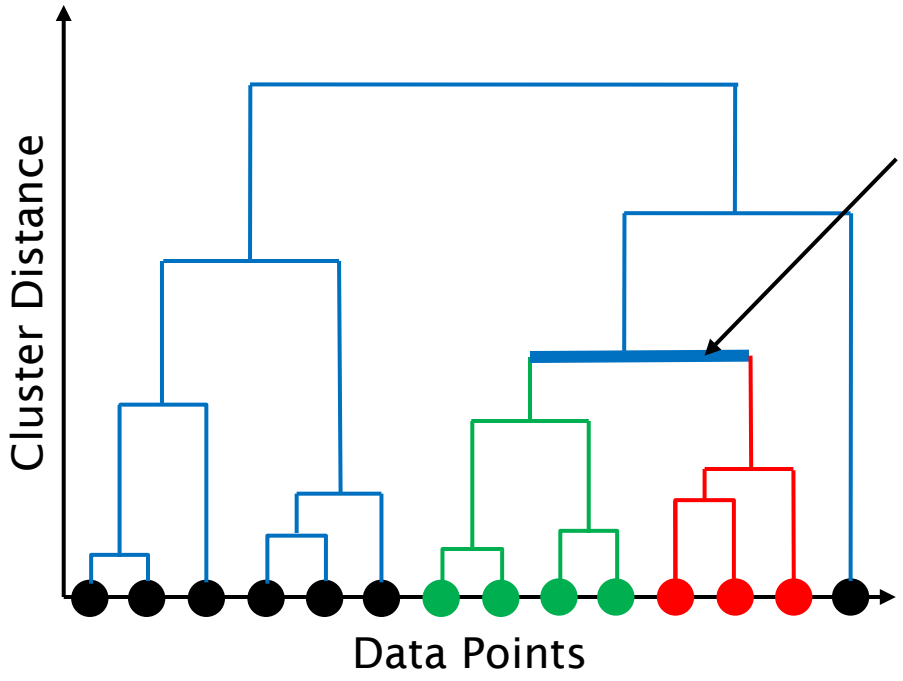
- Compact visual representation of the result of hierarchical clustering.
- Here shown for agglomerative clustering, but divisive clustering also works.
- Path from bottom to top shows the clusters a point belongs when clusters are merged (agglomerative clustering).
- Path from top to bottom shows the clusters a point belongs to when clusters are split (divisive clustering).



Data points on x-axis ordered.
y-axis shows distance between pairs of clusters

Dendrogram (2)

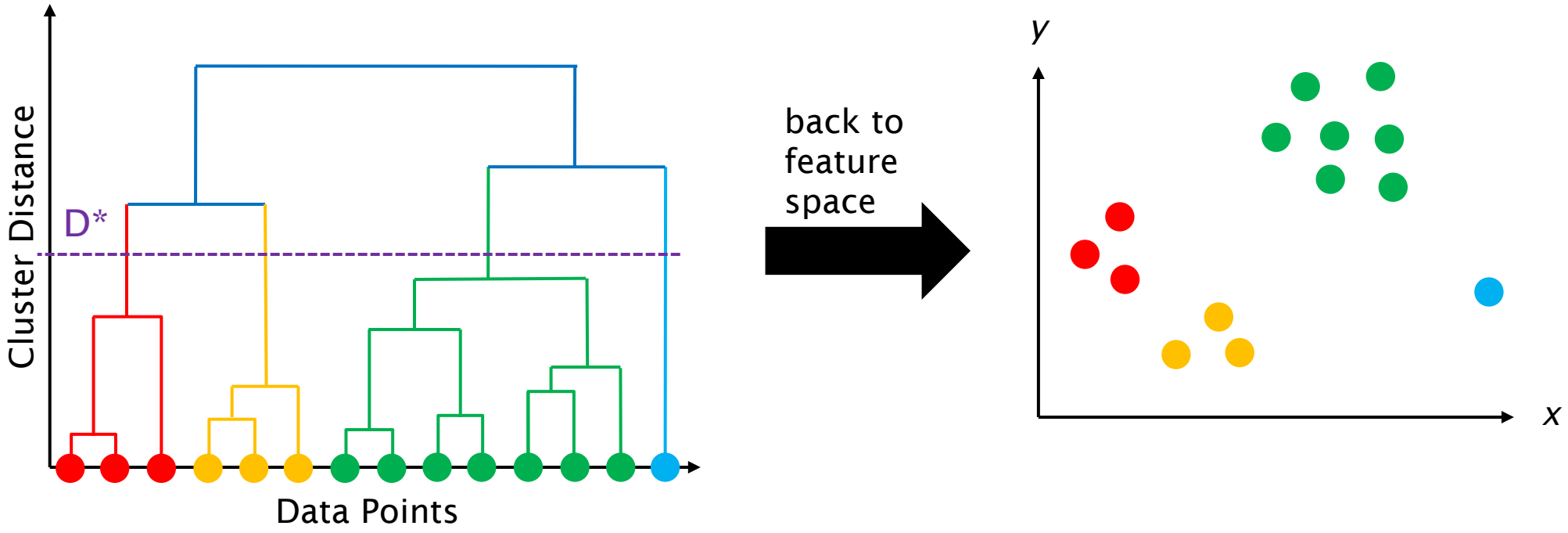
- The height of a ⊓ in a dendrogram shows the distance between clusters.



Assume single linkage is used here, then height indicates **min distance** between green points and red points in the two respective clusters.

Dendrogram (3)

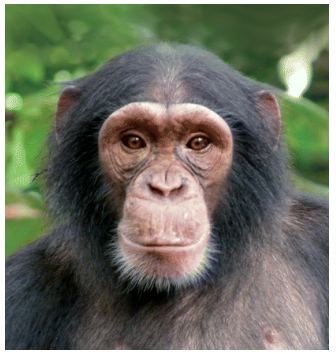
- Extracting a Partition *P* by cutting the dendrogram horizontally at D*.
- Resulting vertical sticks correspond to the different clusters
  → color data points accordingly in feature space.
- Cutting dendrogram at height D* ⇒ D* is the minimum distance between clusters at this level of the cut.

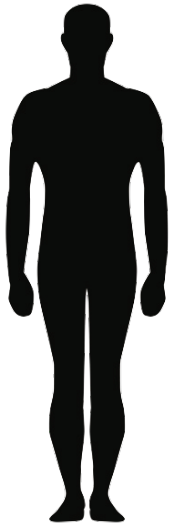


Case Study 2: Hierarchical Clustering of RNA Samples

mRNA tissue samples from different species (Homo sapiens, chimpanzee, orangutan, rhesus macaques from prefrontal cortex, heart, liver, kidney analyzed in microarray.

Goal: Group samples (tissue type and species) based on similarity in a hierarchy.





Application Setup

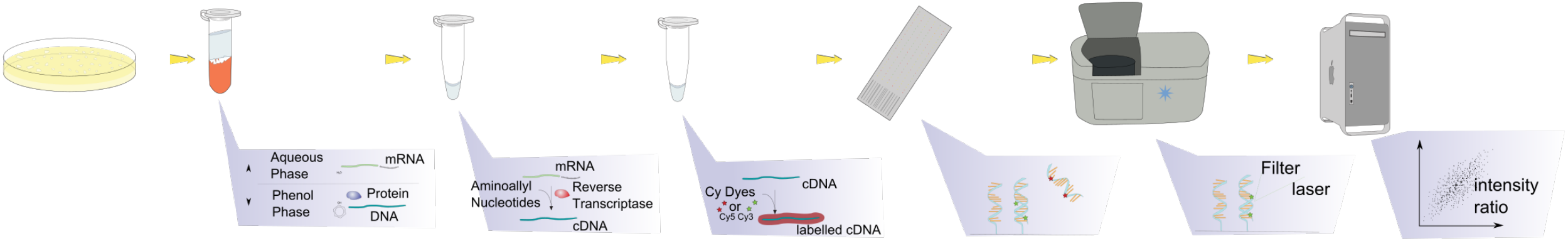
- mRNA (messenger RNA) from different species and tissues is analyzed on a microarray.
- The data set contains 31 tissue samples from:

Abbreviation	Species	tissue
c	Pan troglodytes - Chimpanzee	prefrontal cortex
h	Homo Sapiens - Human	prefrontal cortex
ob	Pongo pygmaeus - Orangutan	prefrontal cortex
oh	Pongo pygmaeus - Orangutan	heart
ol	Pongo pygmaeus - Orangutan	kidney
ol	Pongo pygmaeus - Orangutan	liver
r	Macaca mulatta - rhesus macaques	prefrontal cortex

Microarray: detect presence of certain genes

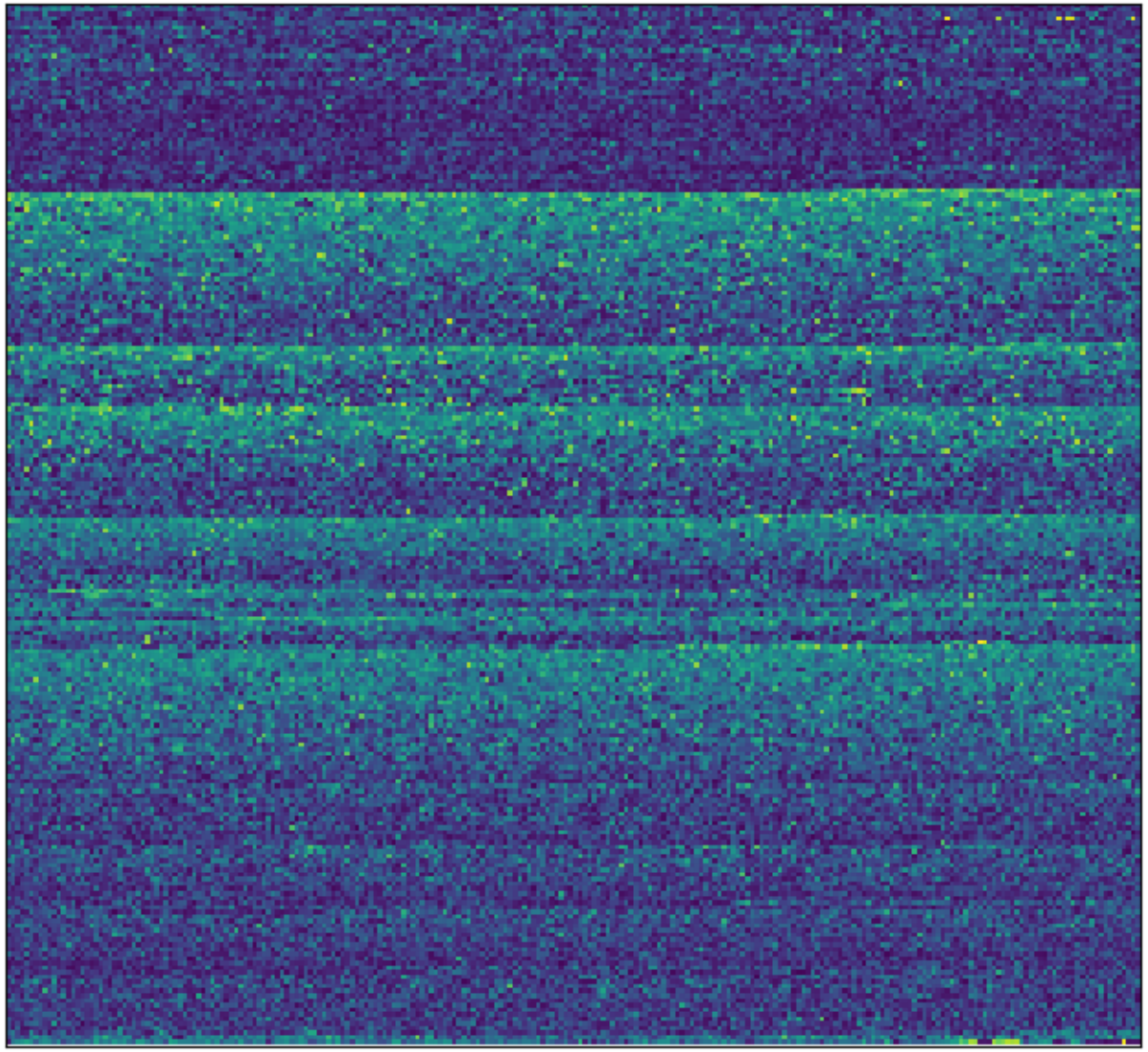
- A Microarray is a Lab-on-a-Chip. The present chip contains 54,674 micro-compartments, called spots.
- mRNA from the issue samples is processed using a number of steps, turned into complementary DNA (cDNA), labeled with a color dye, and added onto the chip.
- Each spot has a number of probe DNA samples which encode a specific gene we are looking for.
- The sample DNA with color dye bind to the spots if the the sample contains the specific genes.
- A laser reads out the spot if is received, the sample contains the gene.

Affymetrix GeneChip Microarray



source: wikipedia.com

Sample Readout



Sample: Homo Sapiens (h45)

About 54,650 real-valued data points from the spots on micro array.

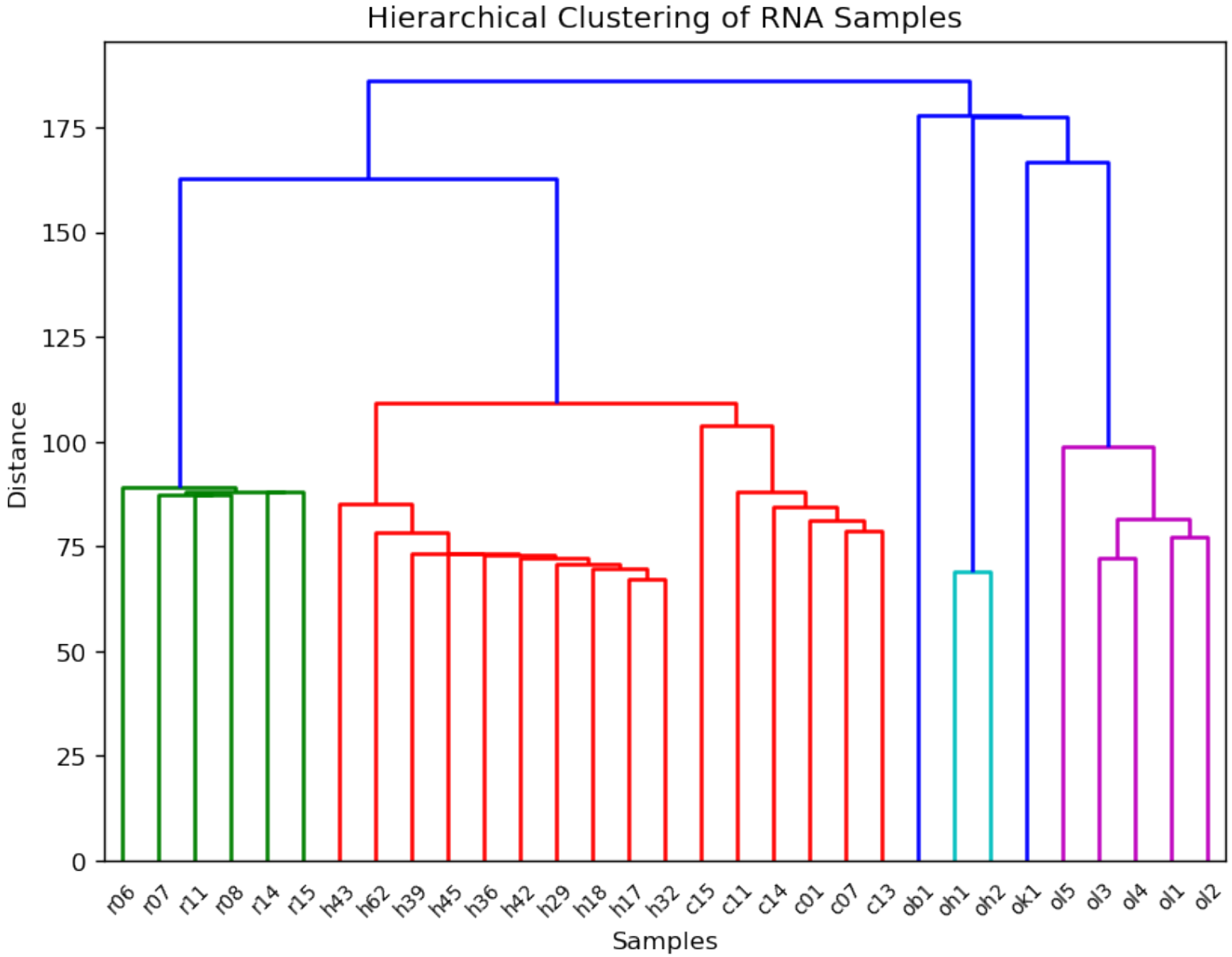
The color represents the strength of the signal within a spot, i.e., the amount of sample DNA present (tagged with a marker dye).

Note: The strength DNA hybridization (binding of the sample DNA with the target) depends on how similar sample and target are.

A weak binding causes the dye to wash off a lot of the dye before the readout → weaker signal.

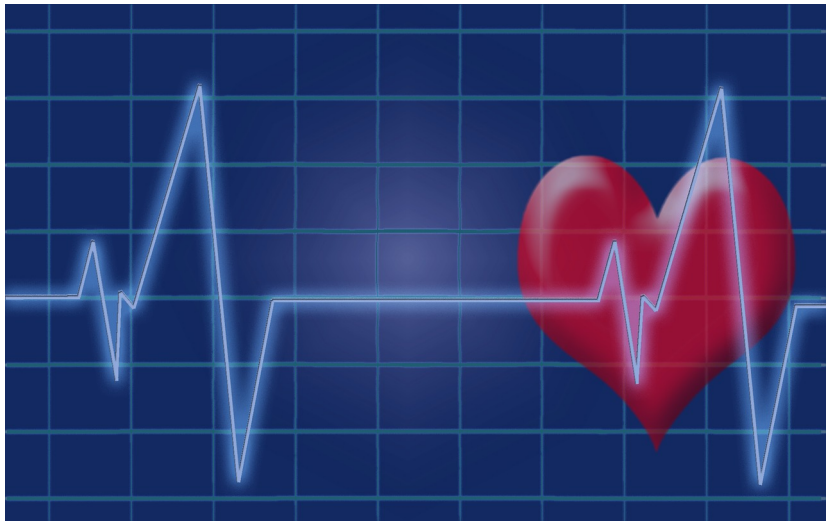
Result: Hierarchical Cluster

Legend
r: Rhesus Macaques
h: Homo Sapiens
c: Chimpanzee
ob: Orangutan (brain)
oh: Orangutan (heart)
ok: Orangutan (kidney)
ol: Orangutan (liver)



Case Study 3: EKG Anomaly time series, overlapping sequences with a windowing kernel

EKG time series is split into short overlapping sequences. The sequences are clustered and serve as a basis for signal construction. New signals are split into sequences and reconstructed using the base sequence from the nearest cluster. The error between the original signal and reconstructed signal is determined. If error too large we have an anomalous situation.

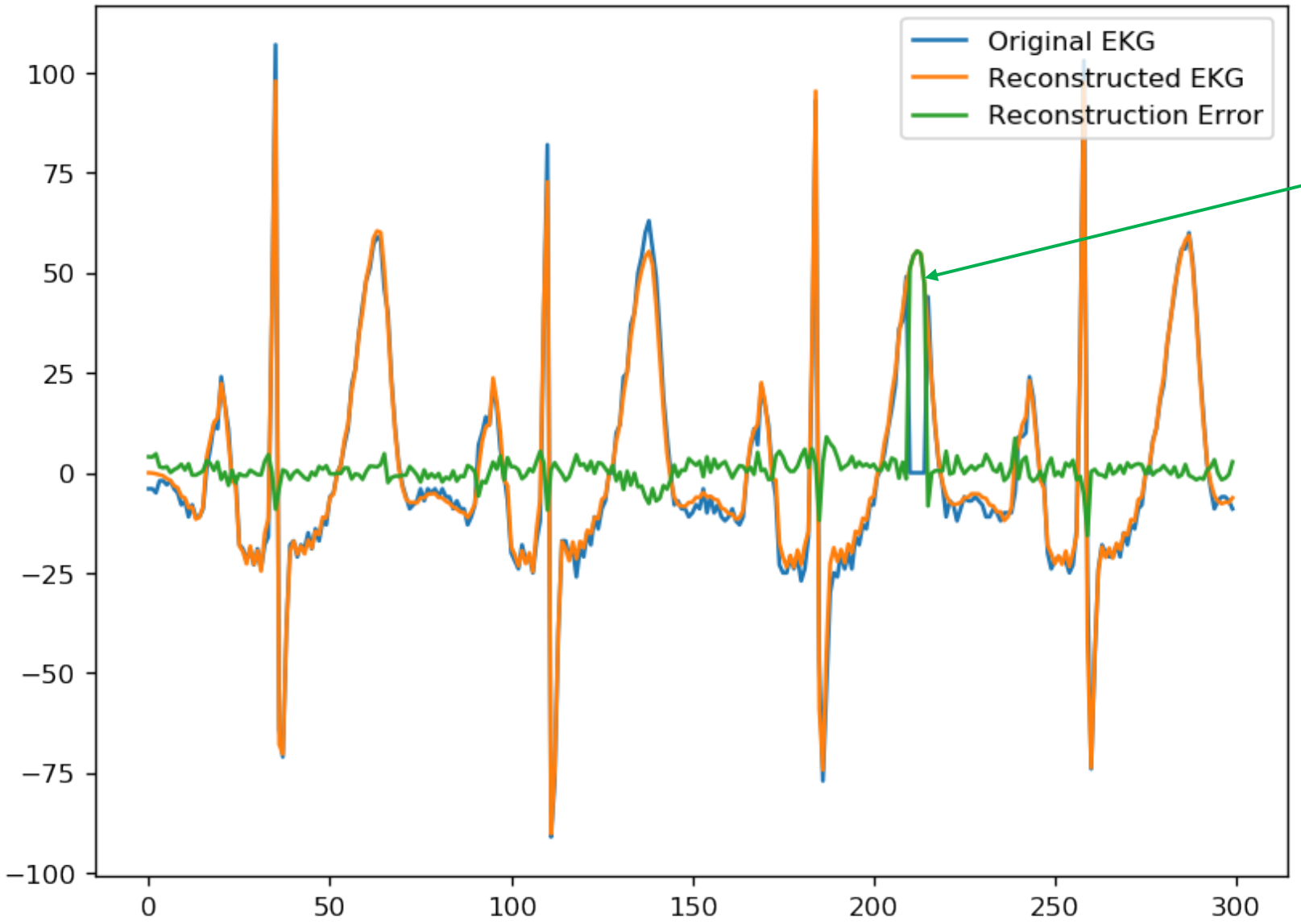


Data set from:
https://physionet.org/physiobank/database/apnea-ecg/

Based on book by

T. Dunning and E. Friedman: *Practical Machine Learning – A new Look at Anomaly Detection*. Chapter 4: More Complex, Adaptive Models. O'Reilly. 2014.

Outlier detection from Anomalous Signal



Peak in error
signal > threshold
→ outlier detected

Distribution-Based Methods

Gaussian Mixture Models

Gaussian Mixture Models

- Observed data points are regarded as a mixture of *k* multivariate Gaussian distributions.
- The number *k* of Gaussians in the mixture is called the number of components in the mixture.
- Each component *i* goes into the mixture with a weight w_i, which sum to 1.
- **Example**: Data Points and two fitted Gaussians.
  Task: Find, i.e., estimate, parameters of the distribution μ_1, μ_2, Σ_1, and Σ_2:

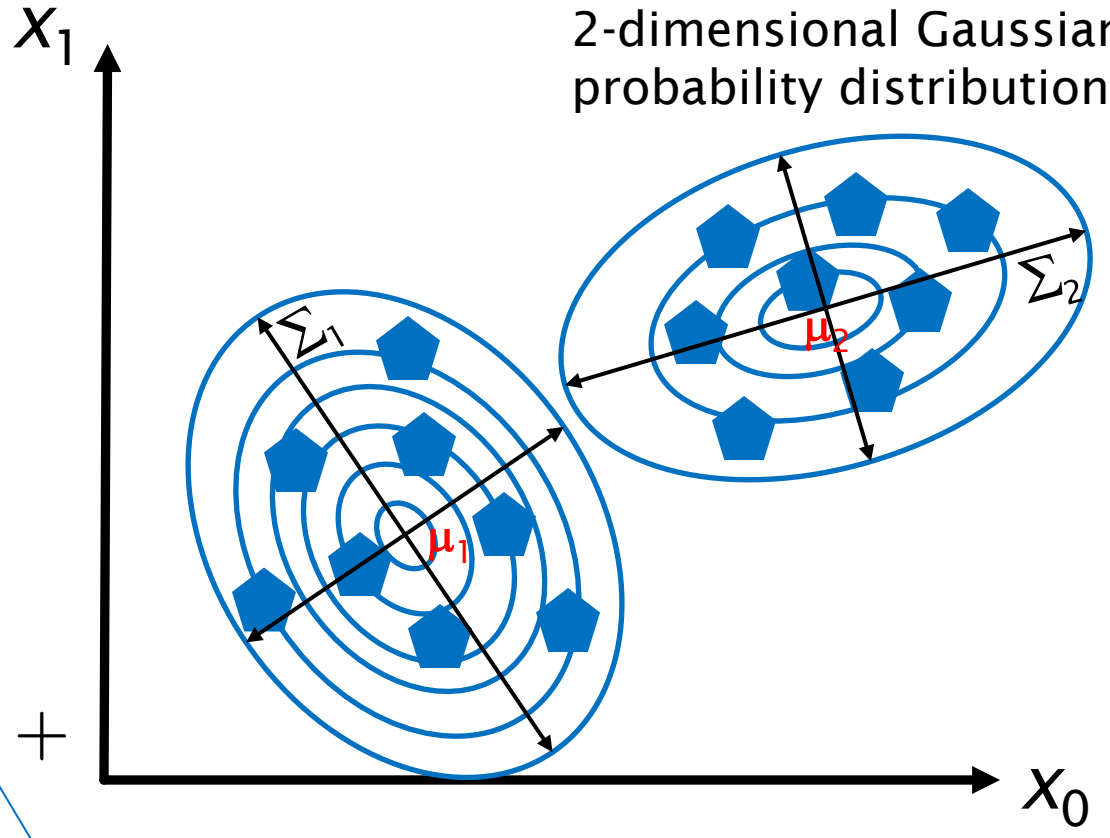
$$p(\mathbf{x}) = \frac{w_1}{\sqrt{|2\pi\boldsymbol{\Sigma}_1|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^{\mathrm{T}}\boldsymbol{\Sigma}_1^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)} + \frac{w_2}{\sqrt{|2\pi\boldsymbol{\Sigma}_2|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^{\mathrm{T}}\boldsymbol{\Sigma}_2^{-1}(\mathbf{x}-\boldsymbol{\mu}_2)}$$

Mean Vectors

Determinant

Covariance Matrices

contour plot of two 2-dimensional Gaussian probability distributions.

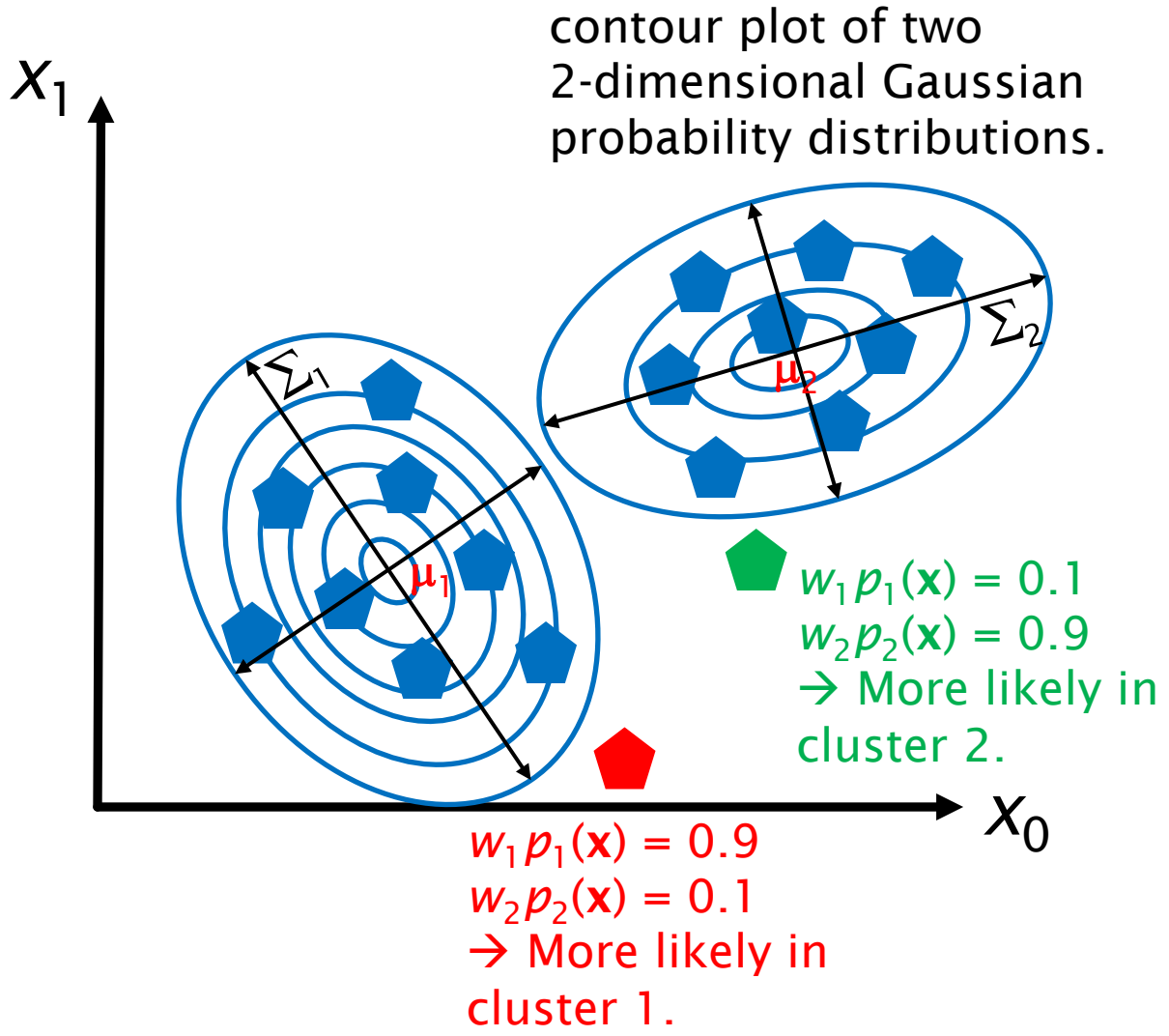


Gaussian Mixture Models for Clustering

- Once the parameters are learned, one can use the distribution for cluster assignment.

- **Soft-clustering** of a point **x**:
  Determine for $w_i\, p_i(\mathbf{x})$ for each of the k components.
  → The k probability densities signify the likelihood that the data point belongs to a particular cluster.

- **Hard-clustering** of a point **x**:
  Determine $w_i\, p_i(\mathbf{x})$ for each of the k components and pick that component with the highest probability density value.



EM Algorithm

- The EM Algorithm (expectation-maximization) is an iterative method to find maximum likelihood of parameters of a statistical model that is based on unobserved latent variables.
  - Example of observed data: set of data points $X = \{ \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \}$
  - Example of parameters to be estimated: μ_1, μ_2, Σ_1, and Σ_2
  - Example of unobserved latent variables: the cluster (component) assignments $c_1,\ldots, c_n$ for the data points $x_1, x_2, \ldots, x_n$.
    → After all, we are dealing with an unsupervised scenario!
    → If we had the cluster labels for training (supervised scenario) the calculation of μ_j and Σ_j is trivial: sample mean → μ_j and sample covariance → Σ_j.
- EM Algorithm: alternates between E and M step.
  - E-Step: Estimate probabilistic cluster (component) membership labels $P(c_i \mid \mu_j, \Sigma_j)$ for all $\mathbf{x}_i$ using current parameters μ_j, Σ_j of all components j.
  - M-Step: Given the current cluster labels ci find μ_j, Σ_j that maximizes the likelihood for $P(c_i \mid \mu_j, \Sigma_j)$

Gaussian Mixture Models in scikit-learn

full: each component has separate cov. matrix
diag: each component has diagonal cov. matrix
tied: all components share same cov. matrix
spherical: each component has single variance

```
from sklearn.mixture import GaussianMixture

k = 3
gmm = GaussianMixture(n_components=k, covariance_type='full')
gmm.fit(data)
```

```
gmm.means_
gmm.covariances_
gmm.weights_
```

[3,2] matrix with means (cluster centroids)

[3,2,2] tensor with 3 covariance matrices (one for each component)

[3] vector with component weights (sum up to 1)

```
cluster_ids = gmm.predict(data_points)
```

[n] vector, ID of cluster a point most likely belongs to

```
probs = gmm.predict_proba(data_points)
```

[n,k] log probability for each cluster and data point, i.e., element [i,j] means probability that data point i is member of cluster j.