Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

CAS Practical Machine Learning
Introduction

Project: First ML Steps with Python

Prof. Dr. Jürgen Vogel (juergen.vogel@bfh.ch)

scikit-learn

Machine Learning with Python

Bern University of Applied Sciences

Machine Learning with scikit-learn

scikit-learn (http://scikit-learn.org/stable/index.html)

- provides many ML algorithms
- plus data preprocessing, feature selection, evaluation, visualization
- open source with BSD license
- installation
  - pre-installed with Anaconda
  - or see https://scikit-learn.org/stable/install.html
- general documentation see http://scikit-learn.org/stable/documentation.html

Basic Workflow with scikit-learn (1)

1) loading data (http://scikit-learn.org/stable/datasets/index.html)
   - dataset `ds` needs to have a specific format
     - `ds.data` is an array `[n_samples, n_features]` for storing the relevant features (numeric)
     - for supervised training: `ds.target` is an array `[n_samples]` with pre-assigned classes (numeric)
   - prepared random ("generated"), small ("toy") and large ("real-world") datasets exist

2) training the model (http://scikit-learn.org/stable/user_guide.html)
   - each supervised ML algorithm `ml` ("estimator") implements the method `fit(ds.data, ds.target)` that takes a training dataset `ds`

3) using the model (http://scikit-learn.org/stable/user_guide.html)
   - each ML algorithm `ml` implements the method `predict(ds.data)` that takes an array `ds.data` and returns the predicted classes as array `[n_samples]`

Basic Workflow with scikit-learn (2)

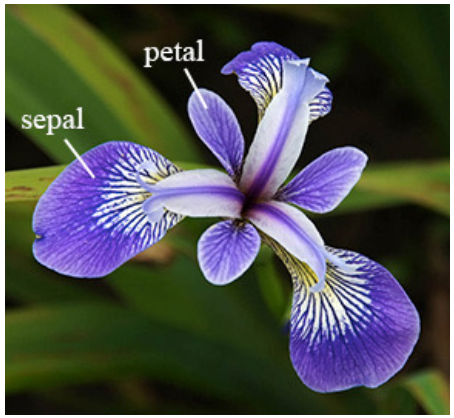
4) evaluating the model (http://scikit-learn.org/stable/modules/model_evaluation.html)
   - each ML algorithm `ml` has a method `score(ds.data, ds.target)` to calculate a model-specific, commonly-used evaluation metric
   - more elaborate evaluation tools support different metrics

5) saving and loading the trained model
   - via joblib (https://pythonhosted.org/joblib/persistence.html)
   - `joblib.dump(ml,filename)` and `joblib.load(filename)`

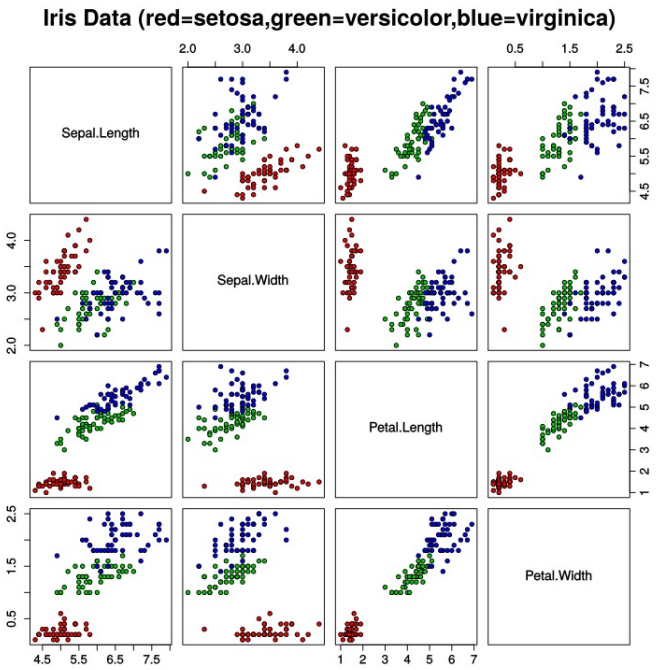
Decision Tree

Baseline Supervised Classification Algorithm

Iris Flower Dataset



- samples of different species of iris flowers
  - 3 species: Iris setosa, Iris virginica and Iris versicolor)
  - 4 features: sepal/petal length/width in centimeters
- introduced in the article: *R. A. Fisher (1936). "The use of multiple measurements in taxonomic problems". Annals of Eugenics. 7 (2): 179–188*
- see https://en.wikipedia.org/wiki/Iris_flower_data_set
- dataset is predefined in scikit-learn (http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html)



Decision Tree with scikit-learn (1)

Jupyter Notebook

1) loading the dataset

```
from sklearn import datasets
iris = datasets.load_iris()
```

2) exploring and visualizing the dataset

- raw dataset

```
print(iris.data)
```

- features

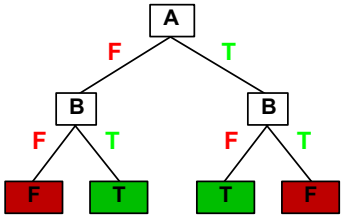
```
print(iris.feature_names)
```

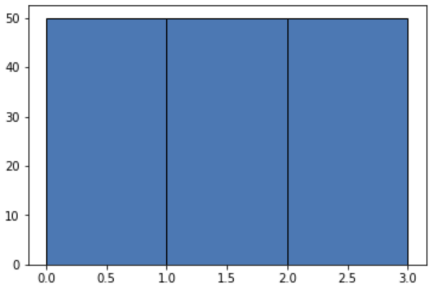
- classes

```
print(iris.target)
print(iris.target.size)
print(iris.target_names)
```

- classes histogram

```
import matplotlib.pyplot as plt
plt.hist(iris.target, bins=(0,1,2,3),
     edgecolor='black')
plt.show()
```

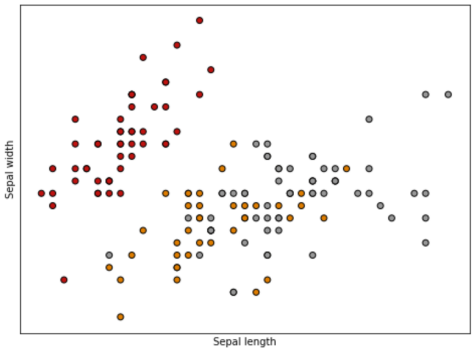




Decision Tree with scikit-learn (2)

2) exploring and visualizing the dataset
   - scatterplot with sepal length and width

```
import matplotlib.pyplot as plt
X = iris.data[:, :2]  # only the first two features
y = iris.target
plt.figure(2, figsize=(8, 6))
plt.clf()
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1,
edgecolor='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xticks(())
plt.yticks(())
plt.show()
```



Decision Tree with scikit-learn (3)

3) training the model (http://scikit-learn.org/stable/modules/tree.html)
   - scikit-learn implements the CART DT algorithm
     - recursively splits the dataset into subsets based on conditions wrt feature values
     - minimizes the Gini impurity to decide on the best split
       - measures the error rate that would result from classifying the remaining data samples randomly according to the distribution of labels in the candidate subset
       - i.e., the error rate that would result if the tree does not grow any further
   - running the classifier

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
# use all but the last sample for training
classifier.fit(iris.data[:-1], iris.target[:-1])
```

4) evaluating the model

```
classifier.predict(iris.data[-1:])
```

Decision Tree with scikit-learn (4)

5) visualizing the model (optional)
   - for rendering, we use Graphviz (http://www.graphviz.org/)
     - general tool for visualizing graphs and networks
     - download from http://www.graphviz.org/download/
     - install and verify via `dot -V`
   - option 1: direct display

```
from sklearn import tree
tree.plot_tree(classifier,
               feature_names=iris.feature_names,
               class_names=iris.target_names,
               filled=True, rounded=True)
```

   - option 2: render image file via command line
     - in Python, save tree in Graphviz file format ("tree.dot")

```
from sklearn import tree
tree.export_graphviz(classifier,out_file='tree.dot',
                     feature_names=iris.feature_names,
                     class_names=iris.target_names,
                     filled=True, rounded=True)
```

     - then render file via command line

```
dot -Tpng tree.dot -o tree.png
```