Softwarewerkzeuge der Bioinformatik

Prof. Dr. Volkhard Helms
PD Dr. Michael Hutter, Markus Hollander,
Andreas Denger, Marie Detzler, Velik Velikov

Saarland University
Department of Computational Biology

Winter semester 2021/2022

Tutorial 10

January 20, 2021

Data visualization & Supervised Machine Learning

In this tutorial, you will analyze another gene expression dataset using Jupyter Notebooks on Kaggle.com. The data has already been normalized and cleaned. It contains 39 tumor samples from liver cancer patients, and 36 samples from adjacent non-tumorous tissue. The goal is to train a basic machine learning model on the data to predict whether a given sample was taken from tumor tissue or healthy tissue, and evaluate its classification performance.
Create a new cell in your notebook for each piece of code you add.
Note: The issues with copy-pasting code from the PDF (missing underscores and additional spaces) are mostly resolved now. Some PDF viewers, like the ones in Edge and Chrome, still remove line-breaks. Try opening the PDF in Adobe Reader, SumatraPDF or Firefox if you have any issues.

Exercise 10.1: Preparation

(a) Create a new Notebook in your Kaggle account, and upload the dataset from the following link as you have done in the previous exercises (+**Add data**):

https://sbcb.inf.ufrgs.br/data/cumida/Genes/Liver/GSE57957/Liver_GSE57957.csv

Give the dataset a name you will recognize, like "*livercancer*".

(b) Read the data into a pandas DataFrame:

```
import pandas as pd
file_path = "../input/livercancer/Liver_GSE57957.csv"
df = pd.read_csv(file_path, index_col=0)
```

Split the data into *features* and *labels*:

```
labels = df["type"]
features = df.drop("type", axis=1)
```

(c) Call the function *labels.value_counts()*. How many tumor and normal samples does the dataset contain?

(d) Print the table *features*. How many samples and microarray probes does it contain?

Exercise 10.2: Preprocessing

In this exercise, we will get the data ready for the machine learning library we will be using (scikit-learn).
A machine learning model is trained and evaluated on a matrix **X**, where the rows correspond to *samples* (in our case tissue samples), and the columns correspond to *features* (here: spots on the microarray).
This matrix is accompanied by a vector **y**, which contains the labels for each sample (here: cancer and normal). The algorithm then tries to find patterns in the data (here: the luminance values of the spots on the microarray), so it can learn to distinguish between label **0** (normal liver tissue) and label **1** (tumorous liver tissue).

(a) The Python package *numpy* provides a data structure for matrices and vectors, called *array*. Pandas objects can be converted to numpy arrays with the *to_numpy()* function:

```
X = features.to_numpy()
print(X)
y = labels.to_numpy()
print(y)
```

(b) The labels in **y** are still strings (i.e. "HCC" and "normal"). It is common practice to convert class labels to positive integer numbers. Scikit-learn provides a convenient object called *LabelEncoder*, which can convert our string labels to integer labels and back:

```
from sklearn.preprocessing import LabelEncoder
label_enc = LabelEncoder()
y = label_enc.fit_transform(y)
print(y)
print(label_enc.inverse_transform(y))
```

(c) Next, we will split our dataset (meaning **X** and **y**) into a *training set* (**X_train, y_train**) and a *test set* (**X_test, y_test**). The algorithm will be trained on the training data (**X_train, y_train**), without knowing anything about the test data. After training, we will use the features of the test set (**X_test**) to predict the labels of the test set. The predicted labels (**y_pred**) will then be compared to the actual labels (**y_test**). Scikit-learn provides a function called *train_test_split()* for splitting the data:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=1
)
```

What is the purpose of the *test_size* and *stratify* parameters? *Hint: look at the official documentation.*

(d) Some of the algorithms we are going to use (specifically PCA and SVM) work best when the features are standardized. You can print the mean and standard deviations for the features in the training set like this:

```
print(X_train.mean(axis=0).round(2))
print(X_train.std(axis=0).round(2))
```

Is the data already standardized?

(e) Scikit-learn provides an object called *StandardScaler* for standardizing our two datasets:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Like many other objects provided by scikit-learn, StandardScaler provides a *fit_transform* function, which is called on the training data, and a *transform* function, which is called on the test data. Look up the term data leakage in the context of machine learning. Why is it important to only fit the StandardScaler to the training data, not to the test data?

Exercise 10.3: Dimensionality Reduction

Currently, each of our 75 samples has datapoints for 47,324 features. The term "curse of dimensionality" refers to the disadvantages of a large number of dimensions. These include a higher computational cost, as well as more redundancy and noise in the data, which can be detrimental to classification performance of the machine learning algorithm.

(a) We are going to perform a *Principal Component Analysis* to reduce the number of dimensions from 47,324 to two. The scikit-learn library provides a class for this:

```
from sklearn.decomposition import PCA

pca_decomp = PCA(n_components=2, random_state=1)

X_train = pca_decomp.fit_transform(X_train)
X_test = pca_decomp.transform(X_test)

print(X_test)
```

(b) Another advantage of dealing with two-dimensional data is that we can easily create a scatter plot of our samples:

```
import matplotlib.pyplot as plt

plt.scatter(
    x=X_train[y_train == 0, 0],
    y=X_train[y_train == 0, 1],
    color="blue",
    label="HCC"
)
plt.scatter(
    x=X_train[y_train == 1, 0],
    y=X_train[y_train == 1, 1],
    color="red",
    label="normal"
)
plt.xlabel("PrincipalComponent1")
plt.ylabel("PrincipalComponent2")
plt.legend()
plt.show()
```

Interpret the plot. Can the preprocessed and transformed data be clearly separated into two classes? Are there any outliers?

Exercise 10.4: Support Vector Machine

Now it is time to train and evaluate the support vector machine classifier (SVC).

(a) First, the data needs to be standardized again:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

(b) We create a SVC object, fit it to the training data, and predict the labels of the test data:

```
from sklearn.svm import LinearSVC

svc = LinearSVC(C=1, class_weight="balanced", random_state=1)
svc.fit(X_train, y_train)

y_pred = svc.predict(X_test)
print(y_pred)
```

(c) The predicted integer numbers we get back from the classifier (**y_pred**), as well as the true labels (**y_test**), can be converted back to text labels with the LabelEncoder from Exercise 10.2:

```
print(label_enc.inverse_transform(y_pred))
print(label_enc.inverse_transform(y_test))
```

Did the classifier perform well on the dataset? How many wrong predictions do you see?

(d) Scikit-learn implements several metrics for measuring classification performance. Print the accuracy of our classification:

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_true=y_test, y_pred=y_pred))
```

Interpret the result of *accuracy_score*.

(e) The *mlxtend* python package provides a function for plotting your classifier after fitting it to the training data. Plot the SVM hyperplane, together with the training data:

```
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(X_train, y_train, clf=svc, legend=2)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

Repeat the same with the test data:

```
plot_decision_regions(X_test, y_test, clf=svc, legend=2)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

How many outliers can you find? Did the algorithm place the hyperplane in a good position?

Have fun!