Softwarewerkzeuge der Bioinformatik

Prof. Dr. Volkhard Helms
PD Dr. Michael Hutter, Markus Hollander,
Andreas Denger, Marie Detzler, Larissa Fey

Saarland University
Department of Computational Biology

Winter semester 2020/2021

Tutorial 10

February 4, 2021

Data visualization & Machine Learning

In this tutorial, you will learn about some of the basics of Data Science. You will manipulate and visualize data with *pandas*, and finally create your own machine learning model with the *scikit-learn* library. The third project, which will be given out later today, will build on the methods described in this tutorial, and go into more detail.
The code that is used in this tutorial is available in a text file on the website, so it can be easily copied. Pasting text that was copied from a PDF can introduce unwanted symbols or line-breaks.

Exercise 10.1: Preparation

For this exercise, we are going to use the website kaggle.com. Kaggle is a social network for data scientists that allows for easy collaboration, and sharing of datasets and analyses. They also host coding competitions that award up to seven figure prize money for the best analysis of a given dataset.
Most importantly, it allows us to run our own calculations in the cloud, without installing any additional software.

(a) Visit kaggle.com and create a user account. You can either sign up with a Google-Account or with your email address and a password. In the latter case, you will receive a confirmation email. Open your email inbox and click the included link in order to verify your email address.

(b) Log in to your account. Now it is time to create the first notebook. On the left hand side of the website, you will find the option **Code** (with the symbol <>). Click on that option, and select **+ New Notebook**. Choose *Python* as the language, and *Notebook* as the type. These should be the default options. Click on *create*. You can click on the name of the notebook on the top left in order to change it. Change the name to something you will recognize, for example *SWW Tutorial 10*.

(c) Your notebook is now saved in your user account. You can access it from the start page by clicking on your profile picture in the top right, selecting *Your Profile* and then the tab *Code*.

Exercise 10.2: Getting familiar with Jupyter Notebooks

What you are looking at now is a *Jupyter Notebook*. A notebook is comprised of *cells*. A cell can either be a *Code cell*, which contains Python code, or a *Markdown cell*, which contains written text, for example to explain what you are doing in your analysis. You can add new code cells by pressing the + icon on top, you can execute the contents of a cell by selecting it and pressing the arrow/triangle next to the plus, or you can run all cells from top to bottom by pressing *Run All*.

(a) Your notebook will already contain a cell. Select it and press the trashcan icon on its top right to delete it, so we can start from scratch.

(b) Type *2+2* into the empty cell, and click on the arrow button.
The result will show up under the cell.

(c) Create a new code cell with the + icon, type *print("Hello World")* into the cell, and execute it. The string *Hello World* should now be printed underneath the cell.

(d) You can assign values to variables. Those values are saved in the environment of the notebook, and can be recalled later. Write *a=5* into an empty code cell, and execute it. Open a new cell, and simply write *a*. That will show you the value of variable *a* after its execution. If the last line of a cell returns or contains a value then that value will be printed underneath the cell.

(e) Create, edit, execute and delete a few cells, until you feel familiar with Jupyter Notebooks. Here are a few useful keyboard shortcuts:

- CTRL+Enter (Command+Enter on Mac) executes the currently selected cell.
- Shift+Enter executes the current cell moves the cursor to the next cell. If there is no next cell, it creates a new, empty cell underneath.
- Pressing ESC lets you enter *command mode*. In this mode you can move around the notebook with the arrow keys and use additional shortcuts, such as *c* and *v* for copy and paste, *a* and *b* for adding new cells above or below the current cell, or *dd* to delete a cell. You can go back to *edit mode* by pressing Enter.

Exercise 10.3: Loading the dataset

In this exercise, we will analyze a *Non-Small-Cell Lung Carcinoma* (NSCLC) dataset with the GEO accession GSE74706. It contains samples of cancer patients, and samples from healthy patients for reference. The website Cumida collects Microarray data from GEO, and prepares it for machine learning analysis. By using their version of the dataset, you don't need to do any preprocessing before starting the analysis.

(a) Click on **+Add data** on the top right. Choose *Upload* in the top right of the window, next to the X. Select the second icon from the top on the left. Enter *lungcancer* as the title of your dataset. Copy the following URL and paste it under **Remote Files**:

https://sbcb.inf.ufrgs.br/data/cumida/Genes/Lung/GSE74706/Lung_GSE74706.csv

Click on **+Add Remote Files**, then on **Create**. Uploading the data will take about 1-2 minutes.

Alternatively, you can download the dataset through the link, and upload it to the notebook manually.

(b) Now, we will load the dataset into a a *DataFrame*, a data structure that is provided by the *pandas* package for Python. A DataFrame is essentially a table with rows and columns, similar to an Excel sheet or a SQL database. A reference can be found here.

(1) First, you have to find out the location of the document you just uploaded on the server. On the top right, there is a dropdown menu containing the input and output data. You will find the dataset under *input* → *lungcancer* → *Lung_GSE74706.csv*. Here, you can copy the path to the file.

(2) This cell will read the *.csv* file into a DataFrame called *lungcancer_df*:

```
import pandas as pd
file_path = "../input/lungcancer3/Lung_GSE74706.csv"
lungcancer_df = pd.read_csv(file_path)
```

(3) Create a new cell, simply write *lungcancer_df* into it, and execute the cell. Now you can see the table underneath. How many rows and columns does it have?

(c) Next, we will clean up the data a bit. The first step is to save the labels (i.e. whether a sample is a normal- or a cancer sample) in a separate list. Next, any column that doesn't contain the gene expression of a gene is removed. That makes the data easier to work with. Create a cell that looks like this:

```
labels_df = lungcancer_df["type"]
features_df = lungcancer_df.drop(["type", "samples"], axis=1)
```

(d) Finally, we calculate a few statistics on the data to get an overview.

(1) Count how often each label occurs:

```
labels_df.value_counts()
```

How many *normal* and *NSCLC* (lung cancer) samples are present in the dataset?

(2) Calculate statistics on the gene expression data:

```
features_df.T.describe()
```

A normalized dataset has the property that the mean values and standard deviations (std) of the samples are very close to each other. Is the dataset sufficiently normalized or is a normalization step necessary?

(3) Create a histogram from the average expression of the genes:

```
features_df.mean().hist()
```

Shortly describe what you see on the plot.

Exercise 10.4: Linear Support Vector Machine

Our goal is to write a machine learning program that is trained on samples from known cancer patients and healthy control samples. Once it has learned the differences in gene expression, it will be able to tell if a previously unknown sample is taken from a tumor or from healthy tissue, from its gene expression alone.

(a) First, create a cell that imports all the functions and libraries that you are going to need:

```
import numpy as np
from sklearn.svm import LinearSVC
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from mlxtend.plotting import plot_decision_regions
```

(b) The machine learning packages we are going to use want the data to be in a specific format. The DataFrames have to be converted to *numpy arrays*, which is another commonly used data structure, and the labels of the samples (i.e. *normal* and *NSCLC*) have to be converted to numbers (0 and 1). It is a convention to call the variable containing the samples and features **X**, and the list of corresponding labels **y**:

```
X = features_df.to_numpy()
y = np.where(labels_df == "normal", 0, 1)
```

(c) Currently, the table with the genes (**X**) has 29148 dimensions per sample, i.e. one for each gene. In order to create a plot that shows what the algorithm is doing, you need to reduce the number of dimensions from 29148 to two. For that, you are going to use one of the most commonly used methods for dimensionality reduction: Principal Component Analysis (PCA). PCA works best if the features are standardized, which is what the *scale* function does:

```
pca = PCA(n_components=2)
X_std = scale(X)
X_pc2 = pca.fit_transform(X_std)
```

(d) Now, you are going to train the machine learning algorithm. A linear support vector machine (SVM) tries to find an optimal hyperplane that separates the data into two areas, based on the samples it is trained on. In this case, the SVM will draw a line, i.e. a hyperplane in two-dimensional space. SVMs assume that the data is standardized, so we need to use *scale* again, the training function is called *fit*.

```
X_pc2 = scale(X_pc2)
svm = LinearSVC()
svm.fit(X_pc2, y)
```

(e) Finally, create a plot of the trained SVM and the dataset:

```
plot_decision_regions(X=X_pc2, y=y, clf=svm)
```

Interpret the plot. What do the axes stand for? Does a linear SVM work well in this case?

Have fun!