Data Mining 2023/2024
Lab exercises

1 Python (do it on your own, no points)

I assume that you know the basics of Python. If you have only a little experience in Python, exercises in this section will guide you through the process of creating a convenient programming environment. At the end of this section you should be able to write and run Python code in Jupyter Notebook and also know basic Jupyter Notebook commands. You should also know how to install new packages and switch environments.

Exercise 1 — Download and install Anaconda. This is distribution of the Python and R programming languages for applications related to data science (see wiki). Read a quick user guide and conda-cheatsheet to learn how to create and switch environments and install new packages.

Exercise 2 — Using Anaconda run Jupyter Notebook and see *Help* → *User Interface Tour, Keyboard Shortcuts*. Learn how to crate, move and run cells. Learn how to get information about objects and methods and how to print their source code (e.g. can use *tab, shift+tab, shift+tab+tab or write '?' and '??' before a method name*).

Exercise 3 — Recall what data structures are available in Python. Pay special attention to list comprehensions as often they help to write more readable and more efficient code.

Exercise 4 — Install and familiarize yourself with `numpy`, `pandas`, `matpoltlib` packages.

a) Read official numpy user guide.

b) Read official pandas user guide.

c) Read official matplotlib user guide.

2 Word-count problem (deadline: 3rd lab)

Exercise 5 — Find the source of your favorite book and save it in UTF-8 format. Load the book and split it into single words. For example you can use a construction like: (5p)

```
with open("Catch_22.txt", encoding="UTF-8") as f:
words = [word
for line in f
for word in line.split()]
```

Change all words to lower case, remove punctuation and remove stop-words. You may try constructions like:

```
from string import punctuation
words = [word.lower().translate(str.maketrans('', '', punctuation)) for
                                                     ... ]

filtered_words = [w for w in words if not w in stopwords]
```

List of stop-words you can find in the Internet (e.g. here). You may also try using stemming procedure to reduce the different forms of a given word to a common form:

```
from stemming.porter2 import stem
filtered_words = [stem(word) for word in filtered_words]
```

Next, convert the obtained list of words into a list of pairs `(word,1)` of the type `(String,Int)`:

```
pairs = [(w,1) for w in filtered_words]
```

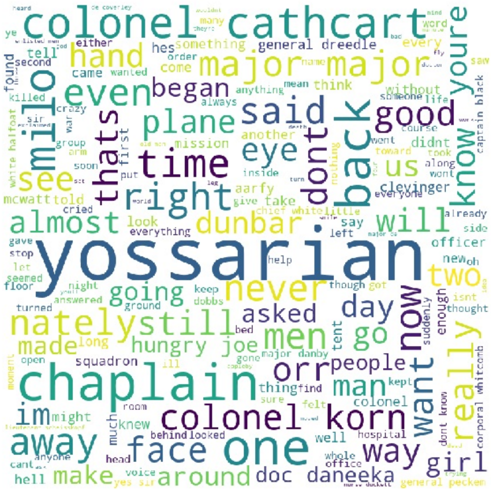
Group the list of pairs by different words and count the total number of occurrences of each word to get pairs `(word, occurrences)`. For example, you may use `groupby` method. However note that `groupby` method requires that input list is sorted by keys and such sorting might be computationally costly for large lists. Try to think up and implement a more efficient way of aggregating occurrences that does not require sorting.

```
from itertools import groupby
pairs.sort()
word = lambda pair : pair[0]
grouped_pairs = [(w, sum(1 for _ in g)) for w, g in groupby(pairs, key=word
                                                     )]
```

Sort obtained list of pairs by the second component in decreasing order:

```
occurrences = lambda pair: pair[1]
grouped_pairs.sort(key=occurrences, reverse=True)
```

Remove some of the initial elements (most common words) and save the result to a text file. Build a word-cloud from the obtained list. You can use the service `http://www.wordclouds.com/`.



Exercise 6 — This is the continuation of the previous task. (5p)

1. Divide your book into chapters. Treat each chapter as a document.
2. Split each documents into words (use lower case, stemming, etc.).

3. Determine the tf-idf weights of all words in all documents:

$$tf\text{-}idf(t,d,D) = tf(t,d) \times idf(t,D),$$

where t denotes a term (word), d denotes a document and D denotes the collection of all documents. Term frequency $tf(t,d)$ is the number of times a term t appears in document d. Inverse document frequency $idf(t,D)$ is often defined as

$$idf(t,D) = log\frac{\mid D \mid}{1 + \mid \{d \in D : t \in d\} \mid}.$$

There are packages that make it easy to find tf-idf weights, but try to implement the appropriate procedure yourself.

4. For each document separately build a word cloud using obtained tf-idf weights.

Exercise 7 — Write a function that takes a word as an input and use tf-idf weights to create the list of chapters of your book most matching to that word (i.e. it should return a list of chapters sorted according to appropriate tf-idf weights). (5p)

Exercise 8 — For each given word in your book make a list of five most common words that appear directly after considered word (but ignore stop-words). Use this summary to generate a random paragraph that resembles a paragraph of your book. (5p)

3 Linear Regression (deadline: 4th lab)

Exercise 9 — Using Python solve applied exercises 13 and 14 from Section 3.7 in ISL book . (20p)

Exercise 10 — Read lab1.ipynb and download Auto.csv from my webpage. Read it as dataframe and change the `origin` column to the `category` type. Split the data into the training and validation set. (20p)

a) Use *statsmodels* library for linear regression with mpg as the response and `horsepower` as the feature. Be prepared to explain parameters returned by `summary()` method that we have discussed, in particular: confidence intervals, p-values, T-statistic, F-statistic and R-squared.

b) Create a scatterplot matrix which includes all of the variables in the data set. You can use `pandas.plotting.scatter_matrix(...)`. Compute the matrix of correlations between the variables, you may use `corr()` function for *pandas* dataframe.

c) Perform a linear regression with mpg as the response and all other variables (except `name`) as the features. Try defining different models with *patsy* library, use symbols +, `*`, `:` and different transformations of the variables like for example `I(np.log(X))` or `I(np.sqrt(X))`. For which model you get the best generalization error?

d) Try to look for outliers and remove them from the data (see e.g.: residual plot, Z-Score). What are high leverage points? How can you detect them (for example see here)? Retrain your models on cleansed data and compare the results.

4 Classification (deadline: 5th lab)

Exercise 11 — Categorical predictors. Using Auto.csv data from previous list create and compare two linear regression models for predicting *mpg*. In the first model use *year* treated as a continuous variable. In the second use *year* treated as a categorical variable. Which model is better? What if there were more than 13 values for variable *year*? Which model is easier to train? (10p)

Exercise 12 — Download Credit.csv file. Dataset is described here. Create logistic regression models with possibly high prediction accuracy for predicting

a) if a given person has an income greater than 50 (hint: create new indicator variable),

b) how many credit cards a person has. (10p)

Exercise 13 — Repeat the previous exercise with the K-Nearest Neighbor and Decision Tree classification models. You may use scikit-learn implementations: KNN and DT. For KNN check different values of parameter *n_neighbors* - the number of considered neighbors. For DT check different values of parameter *max_depth* - the maximum depth of a tree.

What is the best model you get in case a) and b)? To get more reliable answer you may use cross-validation for making many experiments having only one dataset (e.g. use KFold method, see lab3.ipynb) (10p)

Exercise 14 — For problems a) and b) from Exercise 12 choose two continuous predictors that seems to be important and plot decision boundaries for different models (e.g. logistic regression, KNN, DT, RF). You may read this tutorial. (20p)

5 Introduction to Neural Networks (deadline: the last lab)

Exercise 15 — In the notebook lab4.ipynb presented during the lecture we have created a pipeline for text classification (data set: 20newsgroups). Try to increase test accuracy by improving the pipeline and by using stronger classification model. You can get $2(x - 70)$ points , where x is your average test accuracy (in percents) obtained from cross-validation. In the example we have used a single Decision Tree, at least you should try using Random Forest model. What's out-of-bag error?

Exercise 16 — In the notebook lab5.ipynb we have presented a simple neural network created with Keras library. Try to solve classification problem from the previous exercise with the similar neural network. Test different hyper-parameters (e.g. size and number of layers or batch size). If you don't have GPU you may try to use Google Colab. To avoid dealing with very large input vector spaces you may consider only top N most common words in the dataset. (20p)

Exercise 17 — In the notebook lab7.ipynb we have presented the procedure for training a deep neural network for an image classification problem using only a small dataset. We have used data augmentation, transfer learning and fine tuning. Try to carry out a similar procedure for a dataset of your choice. If you have no better ideas you can use flowers dataset. (20p)

J.L.