Introduction to AI: assignment 1 - EUR DS4H

Jupyter, Numpy and Scikit Learn

Part I: classification of machine learning problems

In the table below, different machine learning problems are given. Based on their description, give the corresponding type of problem: regression, classification, clustering or dimensionality reduction.

Solutions

Description	**Type**
Given a dataset of steering angles and front camera images from a moving car, predict the steering angle to be used given an image.	Regression
Use a dataset of a credit company to predict if a borrower will be able to pay back a loan based on his financial situation	Classification
Group books based on their content only.	Clustering
Based on a dataset of segmented and annotated images, detect if an object is present in an image or not.	Classification
Group clients of a service based on their demand profile.	Clustering
Visualize a complex dataset with many features in a 2D graphic.	Dimensionality reduction
Based on a dataset with sensor readings in a car and behavior of the driver (driving normally, eating, talking in the cellphone), predict its behavior from sensor readings.	Classification
Given a corpus of news articles with their corresponding subjects (sports, politics, finance), predict the subject of an article outside the corpus.	Classification
Given a corpus of news articles without subject indication, separate the articles in different groups.	Clustering
Using a dataset of house attributes such as sale price, area in square meters, number of rooms and neighborhood, predict the sale price as a function of the other attributes.	Regression
Predict if a tumor is malignant or benign based on different types of medical imagery. The prediction model is built upon a dataset containing images of tumors along with conclusions from medical diagnosis.	Classification

Part II: Jupyter

For the solutions see notebook "labwork_1_ds4h.ipynb"

For the following parts it will be required to have Anaconda installed in your computer with a recent version of Python (*e.g.* 3.7 or newer). You can download and read installation instructions at the following websites:

`https://www.anaconda.com/products/individual`

`https://docs.anaconda.com/anaconda/install/`

Normally, Jupyter, the numerical algebra library, *numpy*, the machine learning library, *scikit-learn*, the scientific programming library, *scipy*, and the data analysis library, *pandas* are all already installed with Anaconda. If that is not your case, or you if you find problems using these libraries later in the labworks, you will mostly find a solution for your specific problem in forums in the internet.

Jupyter is an interactive programming environment which can be used for different languages such as Python, R, Scala and Julia. Jupyter notebooks, as we will see next, can contain, at the same time, text, equations, code and rich outputs, such as graphics and videos. In this part we will briefly start exploring the Jupyter environment with Python [1].

[1] In case you do not want to install Anaconda, a similar environment to jupyter can be found in *google colab* `https://colab.research.google.com`

- Open Jupyter following the instructions in

  `https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/execute.html`

- To create folders and notebooks, click on *New* on the top right (see Figure 1). To rename a folder or a notebook, click on the check box on its left, then on *Rename* just above (see Figure 2).

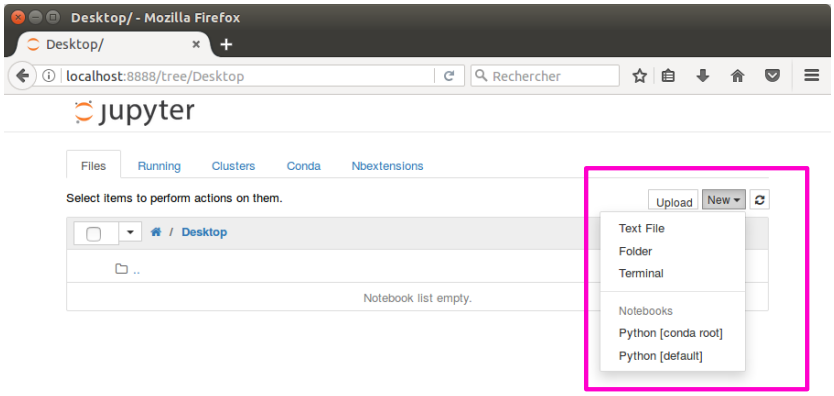

Figure 1: Indication on how to create new folders and notebooks

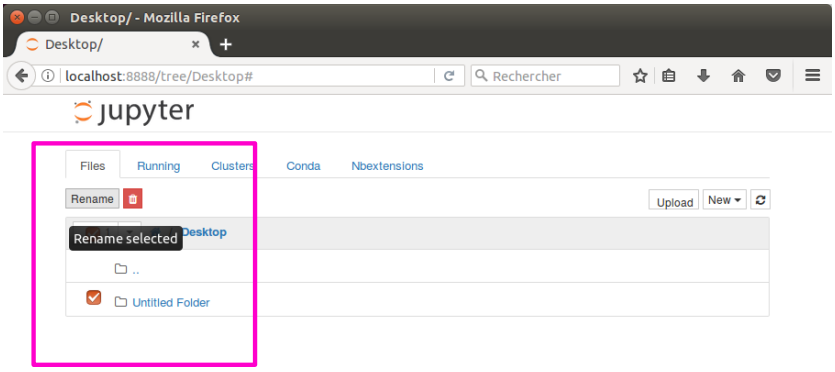

Figure 2: Indication on how to rename a folder or a notebook

Create a folder and rename it as *ML_intro*, for example in your desktop, then inside it, create a subfolder and rename it *Labwork_1*. Within this folder create a notebook and rename it *labwork_1*.

- Once you have created the notebook, you should see something similar to Figure 3, which shows an empty cell.

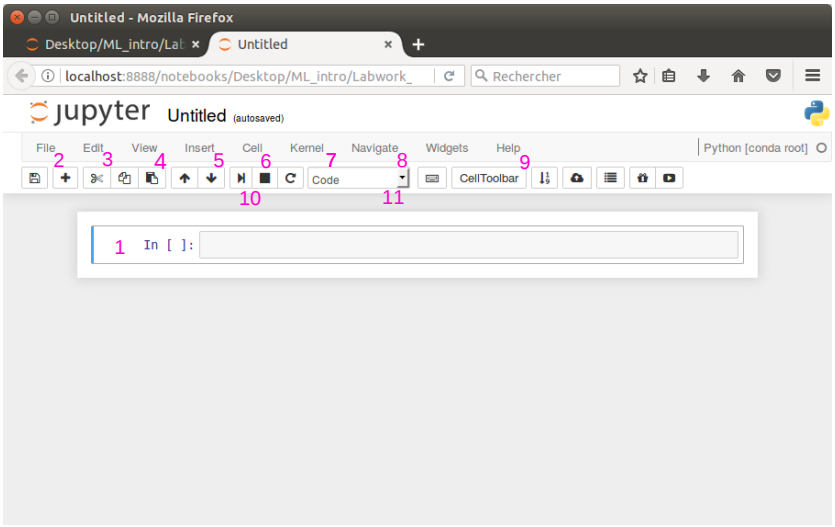

Figure 3: Window displaying an opened notebook

Numbers have been inserted in Figure 3 to indicate the important elements of the Jupyter user interface. They are the following:

1. *Cell content*: the content of Jupyter notebooks is made of cells. Here is where you enter text to be displayed or code to be executed. Text cells can be formatted using Markdown syntax[2], you can also add mathematical expressions using *$.$*, for example $y=ax$ will give $y = ax$ after executing the cell. Note that you need to use LaTex typesetting[3] to write mathematical expressions.
2. *File*: in this menu you can create, open, save and export notebooks. For example, to export your notebook as a PDF file, click on *Download as*, then *PDF via LaTex (.pdf)*. This can be useful when you want to develop a machine learning algorithm and quickly have a PDF report on what you are working on.
3. *Edit*: you can edit cells with the options in this menu.
4. *View*: turn on/off options on the user interface.
5. *Insert*: cell insertion options.
6. *Cell*: options to execute a cell (*Run*) and to change its type.
7. *Kernel*: Jupyter runs on a kernel, the kernel is responsible for running the code part of the Jupyter notebook either on your machine (*http://localhost:8888*) or somewhere else. When you interrupt (*Interrupt*) the kernel, it is equivalent to stop running the code, you can restart it again later (*Restart*). If you want to clear all variables from your notebook, so that you can test again parts

[2] https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

[3] https://en.wikibooks.org/wiki/LaTeX/Mathematics

of it, you simply click on *Restart & Clear Output*, and then run your selected parts. If you want to rerun the entire notebook from the beginning, you click on *Restart & Run All*.

8. *Navigate*: this allows you to navigate through different chapters and sections of your notebook.
9. *Help*: here you can find documentation and tutorials not only about Jupyter but also about Python.
10. *Run cell*: this is the shortcut to run a selected cell. Note that if you run a code cell with variables on it, the values of the variables are kept stored.
11. *Cell type*: here you can choose the different types of cell. For example, *Code* for a code cell and *Markdown* for a text cell.

    If you want to start a chapter in your notebook, then you simply insert a chapter heading in a text cell by starting the cell with #. Similarly, for a section, you start the cell with ##. After running the cells, you can navigate through the different parts of your notebook in *Navigate* as previously explained and see the structure of your notebook in the table of contents.

    The meaning of the other buttons in the interface can be easily obtained by hovering the mouse pointer above their icons.

- Reproduce the notebook shown in Figure 4. Test the Kernel menu options *Restart & Clear Output* and *Restart & Run All* to see their effects and generate a PDF file of your notebook.

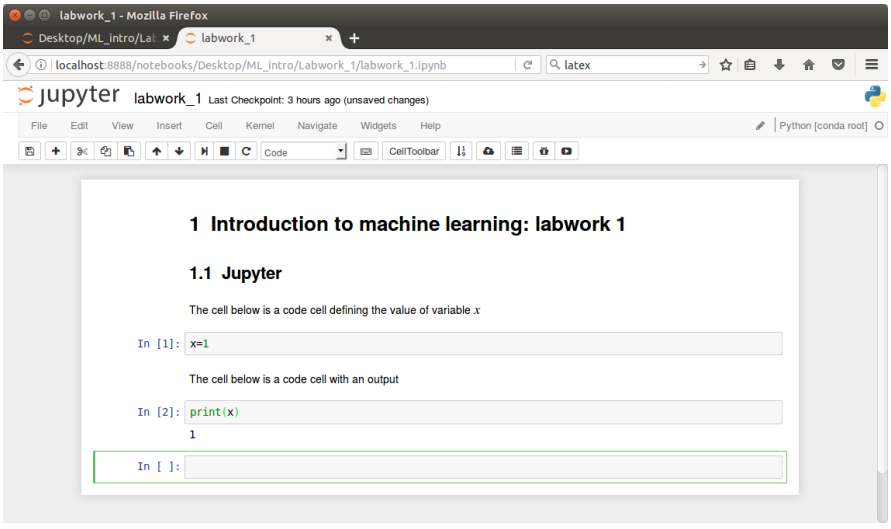

Figure 4: Example of a simple notebook with different types of cells.

Part II: Numpy (and matplotlib)

Numerical operations such as array creation and manipulation and linear algebra operations are often required in machine learning, either during data manipulation/pre-processing or for programming the algorithms. Since these operations are not all built-in functions of Python, we need to use a library. The standard Python library for numerical operations is *numpy*. To use *numpy* functions, you need to import the library, this can be done with the command *import*, to simplify its use we are going to rename it with a shorter name *np* thus leading to the following code line:

```
import numpy as np
```

Create a subheading named *Numpy (and matplotlib): artificial generation of a dataset* and add a new cell with the import command above.

In the following parts, we are going to illustrate the use of some Numpy functions by generating an artificial dataset. We will later apply some algorithms from the Scikit-learn library on this dataset.

We are going to create a dataset with 2 input features, $\mathbf{X} = [\mathbf{x}_1 \quad \mathbf{x}_2]$, and an discrete-valued output $\mathbf{y}$ (two classes: $y^1 = 0$ and $y^2 = 1$). There will be $N = 400$ observations, $N_1 = 200$ observations corresponding to class $y^1 = 0$ and $N_2 = 200$ corresponding to class $y^2 = 1$.

1. To generate artificially the data, we will first set some parameters of the dataset generator: the numbers of observations in each class $N_1 = 200$ and $N_2 = 200$, the distance between the class centers in feature space $d = 1$ and their dispersion (standard deviation) $\sigma_1 = 0.5$ and $\sigma_2 = 0.5$. Add a cell with the parameters definition.

2. We will assume that 2 observations lie at the center of each class feature space, their feature values are $\mathbf{x}^1 = \frac{d}{2}[-1 \; -1]$ and $\mathbf{x}^2 = \frac{d}{2}[1 \quad 1]$. To start generating the matrix of feature values add the following code cell which uses numpy's array function:

   ```
   X_T=(d/2)*np.array([[-1.,1.],[-1.,1.]])
   ```

3. Print the array and the results of the functions *np.shape*, *np.size* and *len* applied to the array. What do these functions do?

4. The array we have created is a transposed version of the feature matrix we really want. Generate the correct feature matrix $\mathbf{X}$ using numpy's transpose function.

5. How do we print on the screen element $x_{2,2}$ of this matrix? How do we print on the screen the second row and the second column of this matrix?

6. Generate 2 random matrices $\mathbf{X}_1$ and $\mathbf{X}_2$ representing the two features and $N_1 - 1$ and $N_2 - 1$ observations of the two classes. Use *np.random.normal* function with means given by $\mathbf{x}^1$ and $\mathbf{x}^2$ and standard deviations given by σ_1 and σ_2.

7. Generate a scatter plot of the observations of the two matrices using the following code cell:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter(X_1[:,0],X_1[:,1]);
plt.scatter(X_2[:,0],X_2[:,1],c='r');
plt.title("Scatter plot of the two classes");
plt.xlabel("$\mathbf{x}_1$");
plt.ylabel("$\mathbf{x}_2$");
```

   You should see a figure similar to Fig. 5.

   Explain what this cell do by commenting each code line on the right. Note that this scatter plot represents visually what information we would have in a classification problem.

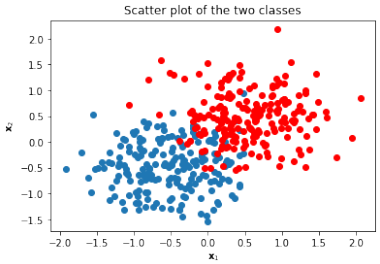

Figure 5: Scatterplot of the simulated dataset.

8. Stack the previously defined matrices $\mathbf{X}$, $\mathbf{X}_1$ and $\mathbf{X}_2$ in one single matrix $\mathbf{X}$ using the command *np.vstack*.

9. Generate one single scatter plot with all points in blue color. Note that this scatter plot represents visually what information we would have in a clustering problem (no class information).

10. Using the functions *np.array*, *np.zeros*, *np.ones* and *np.vstack*, generate a vector $\mathbf{y}$ with the discrete labels, 0s and 1s, corresponding to the classes of the observations in $\mathbf{X}$. Verify its shape, size and length and print it on the screen.

Part III: Scikit-learn

In this part we are going to use the previously generated dataset $(\mathbf{X}, \mathbf{y})$ to illustrate how to use the Scikit-learn library. We are going to focus on two tasks, classification with logistic regression and clustering with k-means.

Scikit-learn documentation can be found at: `http://scikit-learn.org/stable/tutorial/index.html`

1. For classification, you need first to import a set of models with the following command:

```
from sklearn import linear_model as lm
```

2. Machine learning tasks in Scikit-learn mainly correspond to three steps: first a prediction model is specified along with possible customization of some of its parameters, then the model is fitted to the

data, this part corresponds in fact to what we call “learn from data". The fitted model is then applied in a given prediction task.

In the case of logistic regression, we define a logistic regression model with the command

```
logreg = lm.LogisticRegression()
```

then fitting is done with *logreg.fit(X,y)* and prediction is done with *logreg.predict(X_{test})*.

Fit a logistic regression model to the artificially generated dataset and test the prediction model in the same dataset. Compare the predicted outputs $\hat{\mathbf{y}}$ with the original $\mathbf{y}$ to see if there are errors. To print both the predictions and the original $\mathbf{y}$ side by side, you can use the numpy's command *column_stack*.

3. Import the *k*-means clustering method with

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

and define the clustering model with two classes

```
kmeans = KMeans(n_clusters=2)
```

Fit the model on the feature matrix $\mathbf{X}$ and compare the predictions given by the clustering method $\hat{\mathbf{y}}$ with the original $\mathbf{y}$.

4. The accuracy $\mathrm{Acc}_{\hat{\mathbf{y}}(\mathbf{x})}(\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$ of a classification algorithm on a given test dataset $(\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$ is the frequency of predicted outputs $\hat{\mathbf{y}}(\mathbf{x})$ equal to y_{test} obtained when applying the algorithm to the observations of $\mathbf{X}_{\text{test}}$:

$$\mathrm{Acc}_{\hat{\mathbf{y}}(\mathbf{x})}(\mathbf{X}_{\text{test}}, y_{\text{test}}) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \mathbb{1}\left[\hat{y}(\mathbf{x}^i_{\text{test}}) = y^i_{\text{test}}\right]$$

where N_{test} is the number of observations in the test dataset, the couple $(\mathbf{x}^i_{\text{test}}, y^i_{\text{test}})$ is the i-th observation of the testing dataset and the indicator function $\mathbb{1}\left[\hat{y} = y_{\text{test}}\right]$ is defined as follows:

$$\mathbb{1}\left[\hat{y} = y_{\text{test}}\right] = \begin{cases} 1, & \text{if } \hat{y} = y_{\text{test}}, \\ 0, & \text{otherwise.} \end{cases}$$

The accuracy is a measure of the quality of prediction of the model on a given dataset.

Using the function *sklearn.metrics.accuracy_score*, evaluate the accuracy of the previously obtained logistic regression model using as test dataset, the same dataset we have used to fit it. Comment the result (is it closer to zero, 1/2 or 1?).

5. Evaluate the accuracy of the prediction model obtained with k-means using again as test dataset, the same dataset we have used to fit it. Comment the results. Is this accuracy measure available in practical clustering problems?