Introduction to PYTHON

Objective: The objective is to get acquainted with PYTHON

Material: The following on-line materials are recommended.

- `docs.python.org/tutorial` - Introduction into python environment, syntax and data structures. Recommended reading - sections 1, 2, 3, 4 and 5.
- `www.scipy.org/Tentative_NumPy_Tutorial` - Tutorial introducing the scientific computing in Python, array and matrix operations, indexing and slicing matrices.
- `www.scipy.org/NumPy_for_Matlab_Users` -Useful reference to scientific computing in Python if you have previous experience with Matlab programming.
- `matplotlib.sourceforge.net` - Documentation and examples related to matplotlib module, which we shall use extensively through the course to visualize data and results.
- `packages.python.org/spyder` - Overview of Spyder IDE for Python code editing/debugging.
- `https://www.youtube.com/course?list=PL6gx4Cwl9DGAcbMi1sH6oAMk4JHw91mC_` - Series of video tutorials covering basics of Python programming.

Preparation: None

Part 1: Group discussion

For the first group discussion please perform the following tasks:

- Select a *discussion leader* for next weeks group discussion. You should change discussion leaders between each exercise session. The purpose of the discussion leader is to ensure all team members understand the group discussion and the discussion leader should prepare answers for the two tasks under **"Group discussion"** for week 2.
- Find a dataset for the group reports. For more information see the information in today's lecture slides as well as the document "Finding a dataset for the reports" available in the folder "Instructions to group project 1, 2, and 3" on CampusNet.. We encourage that you find unique datasets, however keep in mind the machine-learning tasks for reports 1-3 should be feasible as outlined in the instructions for report 1. For this reason you should have your dataset approved by the exercise instructors.

Part 2: Programming exercises

PYTHON Help: You can get help in your Python interpreter by typing `help(obj)` or you can explore source code by typing `source(obj)`, where `obj` is replaced with the name of function, class or object.
Furthermore, you get context help in Spyder after typing function name or namespace of interest. In practice, the fastest and easiest way to get help in Python is often to simply Google your problem. For instance: `"How to add legends to a plot in Python"` or the content of an error message. In the later case, it is often helpful to find the *simplest* script or input to script which will raise the error.

Piazza discussion forum: You can get help by asking questions on Piazza:
https://piazza.com/dtu.dk/fall2017/02450

1.1 Introduction. Why Python? Which Python version?

Python is a powerful, free and open-source dynamic programming language. Python first appeared in 1991 as a multi-paradigm general-purpose programming language and is widely applied in many domains ranging from scripting, web applications and GUI programming and is one of the most popular languages which significantly differ from the more traditional languages such as C, C++ and JAVA.

In the recent years there has been an increasing interest in numerical computation in Python giving rise to scientific and numerical extensions such as SciPy and NumPy. The purpose of these extensions is to provide math functionality similar or surpassing that of Matlab.

While Matlab was created for easy numerical computation, Python was invented as a general-purpose language and advanced math functionality (e.g. matrices) is provided by third-party extensions. In other words, Python does not come with a build-in matrix type and different third-party matrix packages come with different functionality and may be incompatible. An important consequence is that in e.g. Matlab a 1×3 matrix, a row vector of 3 elements and a list of 3 floating-point numbers will behave the same way just as one would expect, however in Python these will be different *objects* and a function which expects a 1×3 matrix and is passed a 3-element list may not work as expected.

On the positive side, as numerical computation for Python is becoming more mature, these issues has become less severe in the recent years. Furthermore development on Python is happening at a rapid rate and the general-purpose nature of Python mean Python is often much easier to integrate in existing projects.

A very important point is Python exists in two major version (2.x and 3.x) and we will be using the 3.x versions, specifically we recommend version 3.5.2 as the most recent version (3.6.0) is not yet supported by the Anaconda Python distribution. If you already have Python v3.6.0 on your computer and do not intend to install the Anaconda Python distribution, then note that v3.6.0 is backwards compatible with v3.5.2 and the supplied script should run without issue. Python version 2.x is *not* compatible with some of the packages used in this course and some of the supplied scripts will not run. Typing `help()` in the python command prompt (see below)

will print the major version of python in the first line and this should be `Welcome to Python 3.5`.

1.2 Installing the Python environment

Since numerical functionality is supplied by third-party packages, and different packages provide different (possibly incompatible) functionality, it is important to install the right set of packages. Instead of downloading the packages one at a time, we strongly recommend installing a *Python distribution* which will install both Python and most relevant numerical Python packages.

We strongly recommend installing the **Anaconda** Python distribution. To install Anaconda, go to `http://continuum.io/downloads` and download the most recent version for your operating system (as of writing, v4.2.0). If asked, ensure you select Python 3.5 in the Anaconda installer and accept the default settings.

Anaconda will amongst other things install:

- **Spyder IDE**. Spyder offers many tools characteristic for mature IDEs, and thus will enhance your programming experience. These include: context help, text completion, syntax highlighting, enhanced editor, integrated interactive console, project navigator and more. Working with Spyder is quite similar to Matlab environment. See: `code.google.com/p/spyderlib`. In the following it will be assumed Spyder is used to run python commands and edit Python files. Notice the current file can be run in Spyder by pressing `F5`, the current line or selection can be run by pressing `F9`. Further, the Python kernel can be restarted by pressing `ctrl + .` in the iPython console.
- **iPython**: Interactive console with a number of enhancements, see: `www.ipython.org`
- **Jupyter**: The Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text. See `http://jupyter.org/`.
- **NumPy** and **SciPy** - Numerical Python (handles arrays and matrices in Python) and Scientific Python (scientific tools for signal processing, statistics, numerical methods): `www.scipy.org/Download`
- **Matplotlib** - toolbox for plotting and data visualization: `matplotlib.sourceforge.net`

1.3 Additional Packages

While Anaconda include many packages the course require a few additional packages. To install these, open the **"Conda Command Prompt"** and use `easy_install <packagename>`. For instance

```
easy_install neurolab
```

will install `neurolab` which will be used in week 7. To test if a package has been installed correctly use:

```
import neurolab
neurolab.__version__
```

which should output 0.3.5.

1.4 Installing the 02450 Toolbox

The course will make use of several specialized scripts and toolboxes not included with Python. These are distributed as a toolbox which need to be installed.

1.4.1 Download and unzip the 02450 Toolbox for Python, `02450Toolbox_Python.zip` (available from Campusnet). It will be assumed the toolbox is unpacked to create the directories:

```
<base-dir>/02450Toolbox_Python/Tools/     # Misc. tools and packages
<base-dir>/02450Toolbox_Python/Data/      # Datasets directory
<base-dir>/02450Toolbox_Python/Scripts/   # Scripts for exercises
```

For the exercises, you should work on the example scripts in `<base-dir>/02450Toolbox_Python/Scripts/` (notice the scripts are labelled according to exercise number) and not try to write the scripts from the bottom up.

1.4.2 To finalize the installation you need to update your path. In Spyder, go to `Tools -> PYTHONPATH Manager` and press `Add path`. Navigate to `<base-dir>/02450Toolbox_Python/Tools/` and press `Select folder`. Restart Spyder for changes to take effect. Test your path by typing:

```
from tmgsimple import TmgSimple
```

1.5 Python basics

The main goal of today exercises is to establish your Python development environment, as highlighted in the previous section. Secondly, you should get familiar with the basics of Python language and data structures. In particular, make sure you understand the following points.

1.5.1 You can define and display variables, plot data, and even define functions on-the-fly using interactive console. It is useful when you debug your code or experiment with small chunks of code, mathematical expressions, etc. Open a new interpreter in Spyder and type the following instructions:

```
a = 3
b = [1, 2, 3]
print('{0} times list {1} is {2}'.format(a,b,a*b))
```

1.5.2 In many cases you will rather write scripts, and use console only as a supporting/debugging tool. Python script is a file with `.py` extension, which contains instructions, functions, class definitions. Try to write the following code in the Spyder editor, and save it as `hello.py`:

```
import numpy as np

def hello(name, n):
     M = np.random.rand(n,n)
     print('\nHello {0}! This is your matrix:\n{1}'.format(name, M))

name = 'Morten'
matrix_size = 3
hello(name, matrix_size)
```

Now you can run it directly from Spyder (press F5 or choose Run/Run), or from your system command line (type `python hello.py`). Furthermore, you can import the 'hello' script in your other scripts, and access all the 'hello' functions and variables. Start another script, and type `import hello` and afterwards `hello.hello('Boss',3)`.

1.5.3 Note that in Python you need to import packages and external functions before you can use them:

```
from matplotlib.pyplot import *
import numpy as np
from scipy.io import loadmat
```

The commands above import all from the matplotlib module pyplot, import package numpy in namespace np, import function loadmat from package scipy, respectively. Note, that importing all definitions into your local namespace (i.e. `import *`) can overwrite your local definitions, therefore it is generally advised to either specify the functions or import the packages as a namespace.

1.5.4 `array` and `matrix` are two data structures added by NumPy package to the list of basic data structures in Python (lists, tuples, sets). We shall use both `array` and `matrix` structures extensively throughout this course, therefore make sure that you understand differences between them (multiplication, dimensionality) and that you are able to convert them one to

another (`asmatrix(), asarray()` functions). Generally speaking, `array` objects are used to represent scientific, numerical, N-dimensional data. `matrix` objects can be very handy when it comes to algebraic operations on 2-dimensional matrices.

```
import numpy as np
# multiply arrays
a = np.random.rand(2,2)
b = np.array([[2, 0], [0, 2]])
a*b
# convert to matrices and multiply
a = np.asmatrix(a)
b = np.asmatrix(b)
a*b
```

Note different result for array and matrix multiplication. How can you explain this difference? Try typing `a @ b`, how does it work for `array` and `matrix`?

To avoid ambiguity, it is important to keep track of the type of variables. Alternatively, you can use explicit functions:

```
np.dot(a,b)          # matrix multiplication
np.multiply(a,b)     # element-wise multiplication
```

1.5.5 There are many different ways to create arrays and matrices. Try several of the examples:

```
## Creating arrays a1-a8 and matrices m1-m3
a1 = np.array([[1, 2, 3], [4, 5, 6]])   # define explicitly
a2 = np.arange(1,7).reshape(2,3)        # reshape range of numbers
a3 = np.zeros([3,3])                    # zeros array
a4 = np.eye(3)                          # diagonal array
a5 = np.random.rand(2,3)                # random array
a6 = a1.copy()                          # copy
a7 = a1                                 # alias
m1 = np.matrix('1 2 3; 4 5 6; 7 8 9')   # define matrix by string
m2 = np.asmatrix(a1.copy())             # copy array into matrix
m3 = np.mat(np.array([1, 2, 3]))        # map array onto matrix
a8 = np.asarray(m1)                     # map matrix onto array
```

1.5.6 It is easy to extract and/or modify selected items from arrays/matrices. Here is how you can index matrix elements:

```
m = np.matrix('1 2 3; 4 5 6; 7 8 9')
m[0,0] # first element
m[-1,-1] # last element
m[:,0] # first column
m[0:2,-1] # view on selected rows&columns
```

Similarly, you can selectively assign values to matrix elements or columns:

```
m[-1,-1] = 10000
m[0:2,-1] = np.matrix('100; 1000')
m[:,0] = 0
```

1.5.7 Below, several examples of common matrix operations, most of which we will use in the following weeks.

```
m1 = 10 * np.mat(np.ones([3,3]))
m2 = np.mat(np.random.rand(3,3))
m1+m2               # matrix summation
m1*m2               # matrix product
np.multiply(m1,m2)  # element-wise multiplication
m1>m2               # element-wise comparison
m3 = np.hstack((m1,m2[:,0]))    # combine matrices
m3.shape            # shape of matrix
m3.mean()           # mean value of the elements
m3.mean(axis=0)     # mean values of the columns
m3.transpose()      # or: m3.T
m3.I                # compute inverse matrix
```

Try to understand those commands, and apply them to another matrices of your choice. What happens if `m1` and `m2` are `np.array`?

1.5.8 Once installed matplotlib, visualizing data is as simple as in Matlab. You can find many examples on the website/documentation of matplotlib tool. Below, example of plotting sine function with random noise.

```
from matplotlib.pyplot import (plot,show,title)
x = np.linspace(0,4*np.pi,100)
noise = np.random.normal(0,0.2,100)
y = np.sin(x) + noise
plot(x, y, '.-r')
title('Sine with gaussian noise')
show()
```

Can you modify the code to plot `exp()` function on range `x=[0,2]` ?

1.5.9 Python standard library and NumPy/Scipy packages offer many frequently used functions and utilities. There are several ways to **seek help** for the functions and objects you need.
In Spyder IDE you can get immediate context help for the functions you need. You will get list of functions from imported module if you type the namespace followed by dot, for instance `np.` after previous `import numpy as np`. Furthermore, you will get function description in Spyder's Object Inspector, if you start to type the name of the function and the first opening bracket, for example `np.sin(`.
Alternatively, you can get help in your console by typing `help(obj)` or you can explore source code by typing `source(obj)`, where `obj` is replaced with the name of function, class or object.
Note that in practice, often the fastest way is to google your problem.

1.6 Loading data from an Excel file in Python

In this exercise we will consider how to load data from an excel file into Python. As an example dataset we will consider chemical sensor data obtained from the NanoNose [1] project, see also [2]. The data contains 8 sensors named by the letters A–H measuring different levels of concentration of Water, Ethanol, Acetone, Heptane and Pentanol injected into a small gas chamber. The data will be represented in matrix form such that each row contains the 8 sensors measurements (i.e. sensor A-H) of the various compounds injected into the gas chamber.

1.6.1 Inspect the file `<base-dir>/02450Toolbox_Python/Data/nanonose.xls` and make sure you understand how the data is stored in Excel.

Load the data into Python using the `xlrd` package, and get it into the standard data matrix form as described in the beginning of this document.

See `ex2_1_1.py` for details.

Script details:

- *You can read data from excel spreadsheets after installing and importing xlrd module. In most cases, you will need only few functions to accomplish it:* (`open_workbook(), col_values(), row_values()`)
- *If you need more advanced reference, or if you are interested how to write data to excel files, see:*
  `http://www.python-excel.org/`

There are 90 data objects with 8 attributes each. Can you get the correct data matrix X of size 90×8?

1.7 Select a dataset for the reports

The course will include three written group reports. Each report will cover one of the three sections of the course:

- Data: Feature extraction, and visualization
- Supervised learning: Classification and regression
- Unsupervised learning: Clustering and density estimation

Each group will find one dataset they will use for the reports. This can either be your own dataset or a dataset you find yourself. Additional details about where you can find a dataset and formal requirements can be found in the document **Finding a dataset for the reports.pdf** on campusnet.

As a guideline your dataset should have at least 60 observations and 5 attributes with at least two of the attributes being interval or ratio. Once you have found a dataset you need to have the dataset approved by one of the teaching assistants of the course. Please select the dataset before the end of week 3 of the course.

As part of project 1 you will have to carry out various machine-learning tasks for the dataset and this exercise will briefly touch upon these questions. Discuss the following two question taken from project 1:

1.7.1 Load the dataset into Python, for instance using the last exercise if the dataset is in Excel format. Use the plot command to plot different columns of your dataset against each other and try to answer the following two questions taken from project 1:

1.7.2 *What the problem of interest is (i.e. what is your data about)*

1.7.3 *What the primary machine learning modeling aim is for the data, i.e. which attributes you feel are relevant when carrying out a classification, a regression, a clustering, an association mining, and an anomaly detection in the later reports and what you hope to accomplish using these techniques. For instance, which attribute do you wish to explain in the regression based on which other attributes? Which class label will you predict based on which other attributes in the classification task? If you need to transform the data to admit these tasks, explain roughly how you might do this (but don't transform the data now!)*

1.8 OPTIONAL

There is much more to scientific scripting in Python, and above we highlighted only several important points. It is therefore highly recommended that you read through the tutorials suggested earlier, especially the Python tutorial (sections 1-5) and NumPy tutorial. The more you get acquainted with Python now, the easier it will be for you to solve machine learning problems in the following weeks. You will benefit from this course most if you try to implement the solutions on your own, before checking the correct answers. Nevertheless, if you run into problems, the guidelines and the correct scripts will be always provided for your reference.

1.9 Tasks for the report

Loading data from Excel into Python. Discussion of how the various machine-learning tasks can be carried out for your dataset.

References

[1] Nanonose project.

[2] Tommy S Alstrøm, Jan Larsen, Claus H Nielsen, and Niels B Larsen. Data-driven modeling of nano-nose gas sensor arrays. In *SPIE Defense, Security, and Sensing*, pages 76970U–76970U. International Society for Optics and Photonics, 2010.