



Introduction to Python and Scikit-Learn

Machine Learning 2022

Slides P. Zanuttigh

Material from: M. Huenerfauth, G. van Rossum, R.P. Muller, P. Dragone, A. Passerini



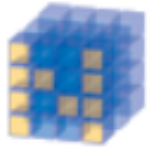
- ❑ Interpreted high-level general-purpose programming language
- ❑ It is open source !
- ❑ Object Oriented programming model
- ❑ Current stable version is 3.9
 - There are relevant changes from Python 2.x to 3.x
 - For this course we'll use **Python 3.x**

Resources:

- ❑ Website: <http://www.python.org>
- ❑ Documentation: <http://www.python.org/doc/>

Modules: SciPy ecosystem

SciPy (pronounced "Sigh Pie") is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:



NumPy
Base N-dimensional
array package



SciPy library
Fundamental library for
scientific computing



Matplotlib
Comprehensive 2D
Plotting

IP[y]:
IPython

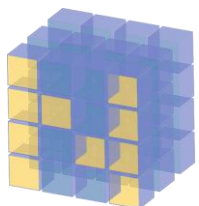
IPython
Enhanced Interactive
Console



Sympy
Symbolic mathematics



pandas
Data structures &
analysis



NumPy

- ❑ Scientific computation capabilities within Python
 - Similar to Matlab functionalities
- ❑ Fast array operations
- ❑ 2D arrays, multi-D arrays, linear algebra, etc...

Modules: scikit-learn



- ❑ Machine Learning library in Python
- ❑ Based on numpy and scipy
- ❑ Open source
- ❑ We'll use this library for the labs !!
- ❑ Includes linear ML models, SVM, Neural Networks, clustering tools, ...
- ❑ Documentation: <http://scikit-learn.org/stable/documentation.html>
- ❑ Reference Manual: <http://scikit-learn.org/stable/modules/classes.html>

Setup: Your Home PC or Laptop



For your PC:

- ☐ Install **Anaconda** (with Python 3)
- ☐ Install **scikit-learn** (if not already installed by Anaconda)
 - Install scikit-learn with anaconda: `conda install scikit-learn`
 - or install with pip: `pip install -U scikit-learn`
 - It requires: Python (≥ 3.4), NumPy ($\geq 1.8.2$), SciPy ($\geq 0.13.3$)
 - If required install the dependencies with pip or conda
- ☐ Install **jupyter notebook**
 - ☐ With anaconda it is installed by default
 - ☐ Can be launched with : `jupyter notebook` or `jupyter-lab`



- ❑ Start the computer under **linux**
- ❑ To login you can use your DEI account or the temporary account provided by the instructor if you do not have a DEI account
- ❑ Setup Anaconda 3 environment with Python 3:
`source /nfsd/opt/anaconda352/anaconda352.sh`
- ❑ Launch jupyter notebook or lab
`jupyter notebook` or `jupyter-lab`

How to use: Jupyter notebook / lab



- ❑ Run with : `jupyter notebook` or `jupyter-lab`
 - Jupyter lab has some extra features
- ❑ Interactive environment inside the web browser
- ❑ You can run each block of code and see the output
- ❑ Can combine code and text (comments / description)
- ❑ ***We'll use jupyter notebooks for the lab deliveries***

If you need a tutorial:

- ❑ See the provided `python_intro_labs` script
- ❑ Jupyter notebook tutorial at: <https://github.com/kuleshov/cs228-material/blob/master/tutorials/python/cs228-python-tutorial.ipynb>

Operators and Variables

- ❑ Assignment uses `=` and comparison uses `==`
- ❑ For numbers: `+` `-` `*` `/` `%` work as expected
 - Special use of `+` for string concatenation
 - Special use of `%` for string formatting (as with *printf* in C)
 - Logical operators are words (`and`, `or`, `not`) *not* symbols
- ❑ The basic printing command is `print`
 - ❑ For strings can use `" "` or `' '` to specify : `"abc"` `'abc'` are the same
- ❑ The first assignment to a variable creates it
- ❑ Variable types don't need to be declared (weakly typed)
- ❑ Python figures out the variable types on its own

Assignments

- ❑ Binding a variable in Python means setting a name to hold a reference to some object
- ❑ Assignment creates references, not copies
- ❑ Names in Python do not have an intrinsic type
 - Objects have types !
 - Python determines the type of the reference automatically based on the data object assigned to it
- ❑ You create a name the first time it appears on the left side of an assignment expression: (e.g., `x = 3`)
- ❑ A reference is deleted via garbage collection after any names bound to it have passed out of scope

- ❑ Handled through the numpy library
- ❑ A numpy array is a grid of values, all of the same type
- ❑ It is indexed by a tuple of non-negative integers
- ❑ The *shape* of an array is a tuple of integers giving the size of the array along each dimension

Examples:

```
import numpy as np
a = np.array([1, 2, 3])    # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"
```

```
b = np.array([[1,2,3],[4,5,6]]) # Create a rank 2 array
print(b.shape)                 # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0]) # Prints "1 2 4"
```

Sequence Types

1. Tuple

- A simple **immutable** ordered sequence of items
- Items can be of mixed types, including collection types

2. Strings

- **Immutable**
- Conceptually very much like a tuple

3. List

- **Mutable** ordered sequence of items of mixed types

4. (*Dictionaries*)

- Store a mapping between a set of keys and a set of values

Whitespaces and Functions

- ❑ **Whitespace** is meaningful in Python (especially indentation)
 - ❑ No braces { } to mark blocks of code in Python, ... use consistent indentation instead !
 - ❑ The first line with more indentation starts a nested block, the first line with less indentation is outside of the block
- ❑ **Functions:**
 - *def* creates a function and assigns it a name, *return* sends a result back to the caller
 - Arguments are passed by assignment
 - Arguments and return types are not declared
 - If no *return* statement is present, the function returns a **None** instance

Example:

```
def <name>(arg1, arg2, ..., argN):  
    <statements>  
    return <value>
```

```
def times(x,y):  
    return x*y
```



Your First Program in Python

Develop a simple application in the last part of the lab:

1. Load the provided .csv file with the used car data
2. Use a linear regression to estimate the car prices from the year, kilometers or engine power
 - You can make a simple 1D regression from each one of the parameters independently
 - *(optional) If you like to experiment try a 2D or 3D regression combining multiple cues*
3. Firstly use the scipy *linregress* function
 - Alternatively you can use the `sklearn.linear_model.LinearRegression` class
4. Have a look at the correlation coefficient to see which of the 3 features works better
5. *(optional) Try to manually implement the least square algorithm*
 - You should get exactly the same solution of *linregress* !
 - If never used least squares you can do it later after the lectures on linear models
6. Plot the data and the lines representing the output of the *linregress* and least square algorithms



Linear Regression with scikit-learn

scipy.stats.linregress

- ❑ The function calculates a linear least-squares regression for two sets of measurements
- ❑ `scipy.stats.linregress(x, y=None)[source]`

Parameters:

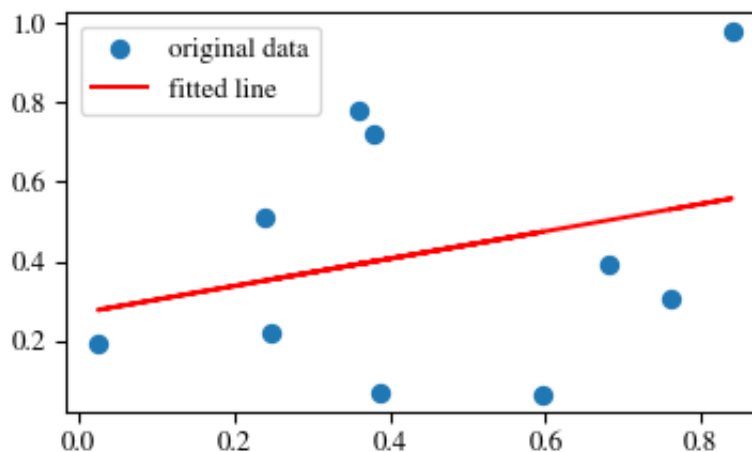
- ❑ `x, y : array_like` Two sets of measurements. Both arrays should have the same length. If only `x` is given (and `y=None`), then it must be a two-dimensional array where one dimension has length 2. The two sets of measurements are then found by splitting the array along the length-2 dimension

Returns:

- ❑ `slope : float` slope of the regression line
- ❑ `intercept : float` intercept of the regression line
- ❑ `rvalue : float` correlation coefficient (see box, ± 1 : total correlation, 0 no correlation)
- ❑ `pvalue : float` two-sided p-value for a hypothesis test whose null hypothesis is that the slope is zero, using Wald Test with t-distribution of the test statistic
- ❑ `stderr : float` Standard error of the estimated gradient

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

Plot Data with matplotlib



Plot the data along with the fitted line using matplotlib

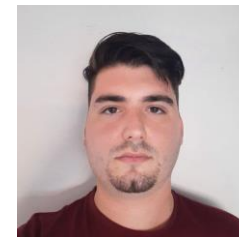
```
from matplotlib import pyplot as plt
plt.plot(x, y, 'o', label='original data')
plt.plot(x, intercept + slope*x, 'r', label='fitted line')
plt.legend()
plt.show()
```

Task for Lab 0

1. Load a dataset with used car data
2. Use a linear regression to estimate the car prices from the year, kilometers or engine power
3. Understand which of the 3 features works better and visualize your results

For lab 0 there is no homework, it is just to get used with Python

For help ask to the instructor or to the TAs



TA:

F. Barbato



M. Mel

Manually implement Least Squares (optional, presented later)

- Compute gradient of MSE on training set and set to 0

$$L_s = \frac{1}{m} \sum_{i=1}^m (< \mathbf{w}, \mathbf{x}_i > - y_i)^2 \rightarrow \frac{\partial L_s}{\partial \mathbf{w}} = \frac{2}{m} \sum_{i=1}^m (< \mathbf{w}, \mathbf{x}_i > - y_i) \mathbf{x}_i = 0$$

- Set

$$\mathbf{A} = \left(\sum_{i=1}^m \mathbf{x}_i \mathbf{x}_i^T \right) \quad \mathbf{b} = \sum_{i=1}^m y_i \mathbf{x}_i$$

- The solution is:

$$\mathbf{w} = \mathbf{A}^{-1} \mathbf{b}$$

- $w[0]$: intercept $w[1]$: slope
- The computation is done using homogeneous coordinates
- Python: 1D array and $m \times 1$ 2D array are different objects
- Inverse of a matrix: [np.linalg.inv\(M\)](#)