UPPSALA UNIVERSITET

Statistical Machine Learning

Introduction to Python and scikit-learn

Niklas Wahlström
Johan Wågberg
Division of Systems and Control
Department of Information Technology
Uppsala University

niklas.wahlstrom@it.uu.se
johan.wagberg@it.uu.se

About Python

- Implementation started 1989
- Python 2.0 released 2000 (end of life as of January 1, 2020), Python 3.0 released 2008
- High-level general-purpose language
- Open source
- A lot of modules available
- Popular in the Machine Learning community

About Python

Ranked most popular programming language in 2019 by IEEE

Rank	Language	Type	Score
1	Python	web, desktop, embedded	100.0
2	Java	web, mobile, desktop	96.3
3	C	mobile, desktop, embedded	94.4
4	C++	mobile, desktop, embedded	87.5
5	R	desktop	81.5
6	JavaScript	web	79.4
7	C#	web, mobile, desktop, embedded	74.5
8	Matlab	desktop	70.6
9	Swift	mobile, desktop	69.1
10	Go	web, desktop	68.0

https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2019

The Python environment

- Anaconda (https://www.anaconda.com/, Includes Python + a few IDEs)
  - Many modules are included by default, others can be installed using `conda`
  - Jupyter Notebook
    - Alternatives: Spyder, Atom, PyCharm, Visual Studio Code, ...

or use run the code via the cloud using Google colab

- We have provided direct links from the course homepage
- Requires that you have a Google account

Important modules

- numpy
- scipy
- scikit-learn
- matplotlib.pyplot
- pandas

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.linear_model as linear_model
```

UPPSALA UNIVERSITET

Outline

niklas.wahlstrom@it.uu.se, johan.wagberg@it.uu.se

UPPSALA UNIVERSITET

Outline

Variables

You do *not* need to declare variable types in Python.

Python syntax for creating a variable and assign a value to it:

```
In [1]: x = 2
```

Variable types: string, float, bool, str, complex, list, tuple, ...
(numpy adds additional numeric types, such as np.int8, np.int16, np.float64, ...)
Check type with `type()`, e.g., `type(x)`

Basics

Add two numbers $x = 2 + 2$

```
In [1]: x = 2 + 2
```

and print the result to the terminal

```
In [2]: print(x)
Out[2]: 4
```

or

```
In [3]: x
Out[3]: 4
```

UPPSALA UNIVERSITET

Help resources

- `help()` shows the documentation for a command, e.g., `help(np.sum)`
- Internet
  - `https://wiki.python.org/moin/`
  - `https://docs.scipy.org/doc/numpy/`
  - `https://scikit-learn.org/stable/user_guide.html`
  - ...

UPPSALA UNIVERSITET

Outline

Vectors (numpy)

A vector $v = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$ is created with numpy (`np`) as

```
In [1]: v = np.array([1,2,4])
In [2]: v
Out[2]: 1 2 4
```

or alternatively

```
In [3]: v = np.array((1,2,4))
```

Matrices (I/II, numpy)

- There are matrices in numpy, but multidimensional arrays are preferred
- A matrix $A = \begin{bmatrix} 2 & 1 & 5 \\ 3 & 6 & 1 \\ 4 & 2 & 4 \end{bmatrix}$ is created as a multidimensional array

```
In [1]: A = np.array([[2,1,5],[3,6,1],[4,2,4]])
```

or

```
In [2]: u = np.array([2,1,5,3,6,1,4,2,4])
In [3]: B = np.reshape(u,[3,3])
```

Matrices (II/II, numpy)

- *or*

```
v0 = np.array([2,3,4])
v1 = np.array([1,6,2])
v2 = np.array([5,1,4])
C = np.column_stack([v0,v1,v2])
```

- A matrix $D = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ is written

```
In [4]: D = np.ones([3,3])
```

Matrix indexing

Python uses 0 based indexes (not 1)

- Consider the same matrix

```
In [1]: A
Out[1]:
array([[2, 1, 5],
       [3, 6, 1],
       [4, 2, 4]])
```

- Access element (0,1):

```
In [2]: A[0,1]
Out[2]: 1
```

- Access first row:

```
In [3]: A[0,:]
Out[3]: array([2, 1, 5])
```

- Access second column:

```
In [4]: A[:,1]
Out[4]: array([1, 6, 2])
```

- Access last column:

```
In [5]: A[:,-1]
Out[5]: array([5, 1, 4])
```

- Access all but last column:

```
In [6]: A[:,:-1]
Out[6]:
array([[2, 1],
       [3, 6],
       [4, 2]])
```

Vector and matrix operations

- Matrix transpose A^T
  `In [1]: A.T` *or* `np.transpose(A)`
- Matrix inverse A^{-1}
  `In [2]: np.linalg.inv(A)`
- Elementwise multiplication $E_{ij} = A_{ij}C_{ij}$
  `In [3]: E = A*C`
- Matrix multiplication $F = AC$
  `In [4]: F = A@C` *or* `np.matmul(A,C)`
- Solve (well-determined) linear system of equations $Ax = v$
  `In [5]: x = np.linalg.solve(A,v)`

UPPSALA UNIVERSITET

Outline

niklas.wahlstrom@it.uu.se, johan.wagberg@it.uu.se

Plotting (matplotlib.pyplot)

Now, we want to plot the following data describing the length of an infant at different ages

Age (months)	0	6	12	18	24
Length (cm)	51	67	74	82	88

 # Insert the data:

```
In [1]: age = np.array([0,6,12,18,24])
In [2]: length = np.array([51,67,74,82,88])
```

 # Plot data:

```
In [3]: plt.plot(age,length,'ro',label="Data")
In [4]: plt.xlabel('age (months)')
In [5]: plt.ylabel('length')
In [6]: plt.legend()
In [7]: plt.show()
```

UPPSALA UNIVERSITET

Outline

niklas.wahlstrom@it.uu.se, johan.wagberg@it.uu.se

Linear regression with numpy example (I/II)

- We would like to fit a straight line to the data with the model

$$y = \theta_0 + \theta_1 x + \varepsilon, \qquad x: \texttt{age}, \qquad y: \texttt{length}$$

- The normal equations

$$\mathbf{X}^\mathsf{T}\mathbf{X}\widehat{\boldsymbol{\theta}} = \mathbf{X}^\mathsf{T}\mathbf{y} \qquad \text{where} \qquad \mathbf{X} = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_5 \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_5 \end{bmatrix}$$

- *# Solve normal equations*

```
In [1]: X = np.column_stack([np.ones(5),age])
In [2]: theta = np.linalg.solve(X.T@X,X.T@length)
In [3]: theta
Out[3]: array([54.6, 1.483])
```


niklas.wahlstrom@it.uu.se, johan.wagberg@it.uu.se

Linear regression with numpy example (II/II)

- Compute the prediction according to

$$\widehat{y} = \widehat{\theta}_0 + \widehat{\theta}_1 x$$

 # Do predictions

```
In [1]: length_hat = theta[0]+age*theta[1]
```

- Plot a line corresponding to these predictions

 # Plot

```
In [2]: plt.plot(age,length_hat,label="model")
```

UPPSALA UNIVERSITET

Outline

Linear regression in scikit-learn (I/III)

- Consider the same regression model as before

$$y = \theta_0 + \theta_1 x + \varepsilon$$

  where x is age and y is length.
- We now use the scikit-learn function `linear_model.LinearRegression()` function to do linear regression
  # Learn the model

```
In [1]: lr = linear_model.LinearRegression()
In [2]: lr.fit(X=np.column_stack([age]),y=length)
```

- Note, the intercept-term θ_0 will always be included (if we don't exclude it explicitly)

Linear regression in scikit-learn (II/III)

- `lr` is an object containing the learned model. We can look at $\widehat{\theta}_0$

```
In [1]: lr.intercept_
Out[1]: 54.6
```

- and $\widehat{\theta}_1, \ldots, \widehat{\theta}_p$

```
In [2]: lr.coef_
Out[2]: array([1.483])
```

Linear regression in scikit-learn (III/III)

- With the learned model `lr` we can do predictions

- *# Do predictions*
  ```
  In [1]: length_hat_lr = lr.predict(X=np.column_stack([age]))
  ```
- *# Plot*
  ```
  In [2]: plt.plot(age,length_hat_lr,'g',label="model")
  ```

UPPSALA UNIVERSITET

Outline

Pandas data frames (I/I)

- Pandas data frames can be used to store data tables.

- *# Create a data frame*

```
In [1]: infantdata = pd.DataFrame(np.column_stack([age,length]),\
                                  columns=['age', 'length'])
In [2]: infantdata
Out[2]:
   age length
0    0     51
1    6     67
2   12     74
3   18     82
4   24     88
```

- Each column in a data frame may be of a different type, and may also have a descriptive name.

Pandas data frames (II/II)

- A data frame can be indexed with number or names

- *# Either ...*

```
In [1]: infantdata['length']
Out[1]:
0    51
1    67
2    74
3    82
4    88
Name: length, dtype: int32
```

 # ... or

```
In [2]: infantdata.iloc[:,1]
Out[2]:
0    51
1    67
2    74
3    82
4    88
Name: length, dtype: int32
```

- *# Plot the data*

```
In [1]: plt.plot(infantdata['age'],infantdata['length'],\
                 'b+', label="Data")
```

Outline

Working with data sets: Example (I/III)

If you load a data set into python, data frames can be handy to use

 # Read csv as data frame (can also load directly from internet address)

```
In [1]: Auto = pd.read_csv('Auto.csv')
```

 # Look at the column names in the data frame

```
In [2]: Auto.info()
Out[2]: <class 'pandas.core.frame.DataFrame'>
RangeIndex: 397 entries, 0 to 396
Data columns (total 9 columns):
mpg             397 non-null float64
cylinders       397 non-null int64
displacement    397 non-null float64
...
```

 # See a statistical summary

```
In [3]: Auto.describe()
```

 # Plot the data

```
In [4]: pd.plotting.scatter_matrix(Auto)
```


niklas.wahlstrom@it.uu.se, johan.wagberg@it.uu.se

Working with data sets: Example (II/III)

- Do linear regression with `displacement` and `weight` as input and `mpg` as output
- *# Do linear regression*

```
In [1]: lr = linear_model.LinearRegression()
In [2]: lr.fit(X=Auto[["displacement","weight"]],y=Auto["mpg"])
```

- *# Predict*

```
In [3]: mpg_hat = lr.predict(X=Auto[["displacement","weight"]])
```

Working with data sets: Example (III/III)

- To evaluate we compute the mean-square-error $\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \widehat{y}_i\right)^2$ for the predictions $\widehat{y}_i$ on the training data.
- *# Compute MSE to evaluate*

```
In [1]: np.mean((mpg_hat - Auto["mpg"])**2)
Out[1]: 18.45
```

- *# Try another set of inputs*

```
In [2]: lr.fit(X=Auto[["displacement","weight","acceleration"]],\
               y=Auto["mpg"])
In [3]: mpg_hat_new = lr.predict(X=Auto[["displacement","weight",\
                                 "acceleration"]])
In [4]: np.mean((mpg_hat_new - Auto["mpg"])**2)
Out[4]: 18.30
```


niklas.wahlstrom@it.uu.se, johan.wagberg@it.uu.se

UPPSALA UNIVERSITET

Outline

Functions (I/III)

To write more structured and efficient, you sometimes want to wrap some code in a function.

Let us say we want to compute the MSE $\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \widehat{y}_i\right)^2$ for the prediction $\widehat{y}_i$

```
def MSE(yhat, y):
    mse = np.mean((yhat-y)**2)
    return mse

In [1]: MSE(mpg_hat,Auto["mpg"])
In [2]: MSE(mpg_hat_new,Auto["mpg"])
```

Functions (II/III)

General structure for Python functions:

```
def myfunction(arg1, arg2, ... ):
    statements
    return object
```

If you want to return multiple results, return a list or vector!

niklas.wahlstrom@it.uu.se, johan.wagberg@it.uu.se

Functions (III/III)

```
def myfunction(arg1, arg2, ... ):
    statements
    return object
```

- All variables created within the function are local to the function. Variables created in the notebook (or similarly) are also readable from within functions. Local variables take precedence in case of name clashes.
- Define your functions either at the beginning of your notebook/script, or in a separate Python file and import it as a module

UPPSALA UNIVERSITET

Outline

Random number generation (I/II)

- The function `random.normal()` in numpy generates realizations of a random Gaussian variable.
- *# Draw 1000 samples from a normal distribution with mean 4 and standard deviation 2*
  `y = np.random.normal(size=1000,loc=4,scale=2)`
- *# Plot*
  `plt.hist(y)`

Random number generation (II/II)

- We want to randomly split the `Auto` data set into two data sets (called training and test).
- First check the number of data sample in Boston

```
In [1]: Auto.shape
Out[1]: (397,9)
```

- Select 200 index values

```
In [2]: trainI = np.random.choice(Auto.shape[0],\
                                  size=200, replace=False)
In [3]: trainIndex = Auto.index.isin(trainI)
```

- Put those in the training data, the other in the test data

```
In [4]: train = Auto.iloc[trainIndex]
In [5]: test = Auto.iloc[~trainIndex]
```

Outline

For-loops

- If you want to repeat a calculation multiple times
- Compute u_i^2, $i = 1, \ldots, 30$, where $u_i \sim \mathcal{N}(0, 1)$

```
u = np.random.normal(size=30)

usq = np.zeros(30)

for i in range(0,30):
   usq[i] = u[i]*u[i]
```

Conditional statements

- If you want to take different actions depending on a previous outcome
- `d = np.random.uniform()`

```
if d < 0.5:
     print("d was smaller than 0.5")
else:
     print("d was bigger than or equal to 0.5")
```