AIML 20-21 - Python crash course

Francesco Cappio Borlino: francesco.cappio@polito.it

```python
print("Hello world")
```

Content

- Introduction to Python:
  - https://www.python.org/about/gettingstarted/
- Introduction to Numpy: *The fundamental package for scientific computing with Python*
  - https://numpy.org/devdocs/user/quickstart.html
- Introduction to Scikit-learn: python library for machine learning. Exercises on Naive Bayes and Linear Regression
  - https://scikit-learn.org/stable/getting_started.html

Introduction to python

- installation and development
- python basics

Installation

- in many operating systems python comes preinstalled
- Windows/Mac/Linux: everything can be easily installed and managed through Anaconda: https://www.anaconda.com/products/individual

Development

- various IDEs are available (Spyder, PyCharm, …);
- python code does not need to be compiled! A simple text editor can be used to write code that is then executed by the `python` interpreter;
- python can be easily integrated with text and images inside notebooks (like this one): https://jupyter.org/
- many online interpreters/notebooks can be used to directly start programming: https://jupyter.org/try

Libraries

- a lot of libraries have been developed to extend Python capabilities

- they can be managed through `conda` or python's own package manager: `pip`

```
pip install numpy
```

- you can use `conda` or `pip` also to install jupyter notebooks!

```
conda install -c conda-forge notebook

pip install notebook
```

```
import numpy as np
np.random.random((2,3))
```

```
array([[0.70793746, 0.67409154, 0.04455432],
       [0.47994953, 0.27602007, 0.32024161]])
```

Python basics: features

- highly readable code, simple syntax
- dynamically typed and garbage collected
- high flexibility. It supports procedural, object-oriented and functional programming.

Python basics: control flow

- `if` statements
- `for` loops
- other useful statements (`break`, `continue`, `pass`, …) and functions (`range`, `enumerate`, `zip`)

```
a = 5
b = None
if a > 5:
    print("a>5")
elif a == 5:
    print("a=5")
else:
    print("a < 5")
for a in range(5):
    print(a)
```

```
a=5
0
1
2
```

```
3
4
```

Python basics: data types

- dynamic typing

```
x = 3
#type(x)
x = 'Hello'
type(x)
if isinstance(x, str):
    print("This is a string")
else:
    print(type(x))
```

```
This is a string
```

- default types (https://www.w3schools.com/python/python_datatypes.asp):

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`

- new types can be easily defined using classes:

```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
    def pretty_print(self):
        print('"' + self.title + '" by ' + self.author)

b = Book("The Lord of the Rings", "J.R.R. Tolkien")

b.pretty_print()
```

```
"The Lord of the Rings" by J.R.R. Tolkien
```

Because of dynamic typing sometimes objects definitions are a bit ambiguous.

Python basics: strings definition and manipulation

- strings support indexing and slicing
- strings are immutable
- concatenation and formatting is really easy
- useful predefined methods are available

```
# string definition and indexing
string = "Hello World"
hello = string[:6]
world = string[6:]
print(hello + world)
```

```
Hello World
```

```
# strings are immutable
string[5] = "_"
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-11-f4028c4d4802> in <module>
      1 # strings are immutable
----> 2 string[5] = "_"


TypeError: 'str' object does not support item assignment
```

```
# string concatenation and formatting
print(hello+world)
print("{}{}".format(hello, world))
print("%s%s %d" % (hello, world, 1))
```

```
Hello World
Hello World
Hello World 1
```

```
# predefined methods
splitted = string.split(" ")
'_'.join(splitted)
```

```
'Hello_World'
```

Python basics: lists definition and manipulation

```
my_numbers = [4,6,7,5,3,2]
squares = []
even_numbers = []
for el in my_numbers:
    squares.append(el**2)
    if el%2 == 0:
        even_numbers.append(el)
print("Squares:", squares)
print("Even numbers:", even_numbers)
```

```
Squares: [16, 36, 49, 25, 9, 4]
Even numbers: [4, 6, 2]
```

List comprehensions and filtering

```
my_numbers = [4,6,7,5,3,2]
squares = [el**2 for el in my_numbers]
even_numbers = [el for el in my_numbers if el%2==0]
print("Squares:", squares)
print("Even numbers:", even_numbers)
```

```
Squares: [16, 36, 49, 25, 9, 4]
Even numbers: [4, 6, 2]
```

Python basics: functions

```
import math

def euclidean_distance(x1, y1, x2, y2):
    return math.sqrt((x1 - x2)**2 + (y1 - y2)**2)

print(euclidean_distance(0,0,1,1))
```

```
1.4142135623730951
```

Introduction to numpy

- definition of multidimensional arrays

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print("a = \n",a)
print("Shape:", a.shape)
```

```
a =
 [[1 2 3]
 [4 5 6]]
Shape: (2, 3)
```

- useful tools for working with arrays (e.g: element-wise multiplication with lists vs numpy arrays)

```
# lists:
a = b = [[1,2],[3,4]]
c = [[0,0],[0,0]]
for i in range(len(a)):
    for j in range(len(b)):
        c[i][j] = a[i][j] * b[i][j]
```

```
# numpy arrays
a = b = np.array([[1,2],[3,4]])
c = a * b
```

- indexing and slicing

```
print(a)
#a[0]
#a[0,1]
#a[0][1]

#a[:,1]
#a[1,:]
```

```
[[1 2]
 [3 4]]
```

- operations:

- element-wise operations
- broadcasting (operations between different arrays of different sizes)

```
print(a+b)
print(a-b)
print(a*b)
```

```
[[2 4]
 [6 8]]
[[0 0]
 [0 0]]
[[ 1  4]
 [ 9 16]]
```

```
print(a+1)
print(a*2)
```

```
[[2 3]
 [4 5]]
[[2 4]
 [6 8]]
```

Exercise 1: random crop image

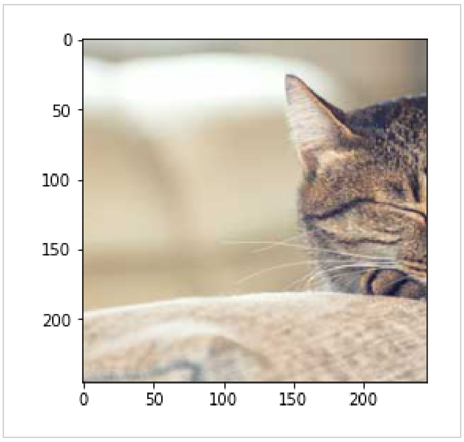
```
import matplotlib.pyplot as plt
import random

cat = plt.imread('cat.jpg')
print("Type", type(cat), "Shape", cat.shape)
plt.imshow(cat)
```

```
Type <class 'numpy.ndarray'> Shape (490, 735, 3)
```

```
<matplotlib.image.AxesImage at 0x7fa621eedf70>
```

```
type(cat)
h, w = cat.shape[0], cat.shape[1]
side = int(0.5*(min(h,w)))
i = random.randint(0,h-side)
j = random.randint(0,w-side)
cat_crop = cat[i:i+side,j:j+side]
plt.imshow(cat_crop)
```

```
<matplotlib.image.AxesImage at 0x7fa62068ba00>
```

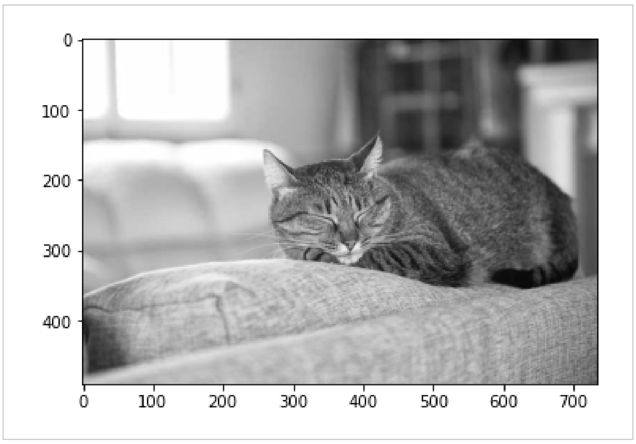


Exercise 2: gray scale image

```
print(cat.shape)
gray_cat = cat.sum(2)
print(gray_cat.shape)
plt.imshow(gray_cat, cmap='gray')
```

```
(490, 735, 3)
```

```
(490, 735)

<matplotlib.image.AxesImage at 0x7fa620324610>
```



Introduction to scikit-learn

Machine learning library built on top of numpy. Features:

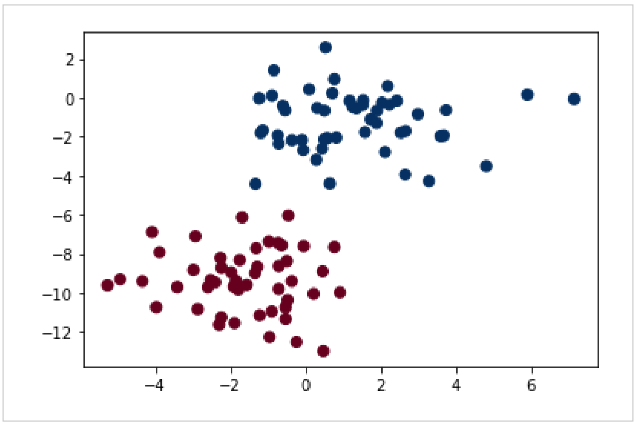
- includes definitions of machine learning algorithms for classification, clustering, regression… e.g.: SVM, k-means, random forest, linear regression…
- it also contains classes and functions useful to simplify research: data preprocessing, model selection, …

Naive Bayes

Problem:

- we have a set of points with 2 dimensions. Each point is assigned a label between 0 and 1;
- we want to build a Gaussian Naive Bayes classifier able to predict the label of new points.

```
from sklearn.datasets import make_blobs
X, y = make_blobs(100, 2, random_state=2, centers=2, cluster_std=1.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu');
```



We use the Bayes' theorem:

$$P(L|\textit{features}) = \frac{P(\textit{features}|L)P(L)}{P(\textit{features})}$$

For each label L_i we want to compute $P(\textit{features}|L_i)$.

If we use a *Gaussian* Naive Bayes approach we make the assumption that *data from each label is drawn from a simple Gaussian distribution*. So we need to estimate for each label the mean and the variance for the values of the various features. We also assume that there is no covariance between the features.

```
mask_l0 = y==0
mask_l1 = y==1
mean_l0 = X[mask_l0].mean(0)
mean_l1 = X[mask_l1].mean(0)
var_l0 = X[mask_l0].var(0)
var_l1 = X[mask_l1].var(0)
print("Mean l0: {}. l1: {}".format(mean_l0, mean_l1))
print("Var l0: {}. l1: {}".format(var_l0, var_l1))
```

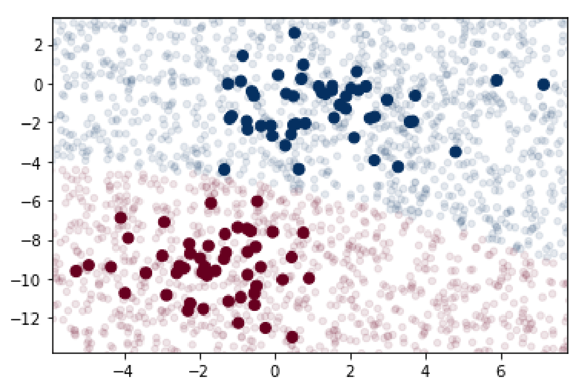
```
Mean l0: [-1.64939095 -9.36891451]. l1: [ 1.29327924 -1.24101221]
Var l0: [2.06097003 2.4771687 ]. l1: [3.33164805 2.22401382]
```

Now when we receive a new point we can compute the probability for the two classes and make a prediction.

```
from scipy.stats import norm

dist_l0_f0, dist_l0_f1 = norm(mean_l0[0], math.sqrt(var_l0[0])), norm(
    mean_l0[1], math.sqrt(var_l0[1]))
dist_l1_f0, dist_l1_f1 = norm(mean_l1[0], math.sqrt(var_l1[0])), norm(
    mean_l1[1], math.sqrt(var_l1[1]))
```

```
def prob_l0(point):
    return dist_l0_f0.pdf(point[0])*dist_l0_f1.pdf(point[1])
def prob_l1(point):
    return dist_l1_f0.pdf(point[0])*dist_l1_f1.pdf(point[1])
def predict_p(point):
    result = prob_l0(point)/prob_l1(point)
    if result > 1:
        return 0
    return 1
def predict_set(points):
    points = [predict_p(point) for point in points]
    return np.array(points)
```

```
Xnew = [-6, -14] + [14, 18] * np.random.rand(2000, 2)
ynew = predict_set(Xnew)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu')
lim = plt.axis()
plt.scatter(Xnew[:, 0], Xnew[:, 1], c=ynew, s=20, cmap='RdBu', alpha=0.1)
plt.axis(lim);
```



GaussianNB is already implemented in `scikit-learn`!

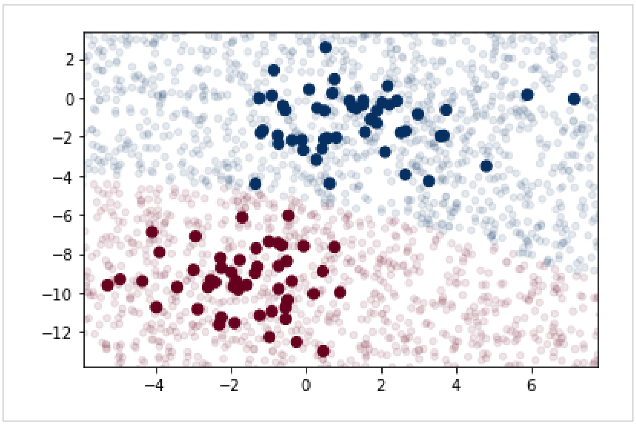
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X, y);
```

It internally computes the same statistics:

```
print("GNB mean:",model.theta_)
print("Mean l0: {}. l1: {}".format(mean_l0, mean_l1))
```

```
GNB mean: [[-1.64939095 -9.36891451]
 [ 1.29327924 -1.24101221]]
Mean l0: [-1.64939095 -9.36891451]. l1: [ 1.29327924 -1.24101221]
```

```
ynew = model.predict(Xnew)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu')
lim = plt.axis()
plt.scatter(Xnew[:, 0], Xnew[:, 1], c=ynew, s=20, cmap='RdBu', alpha=0.1)
plt.axis(lim);
```



Linear regression

Problem:

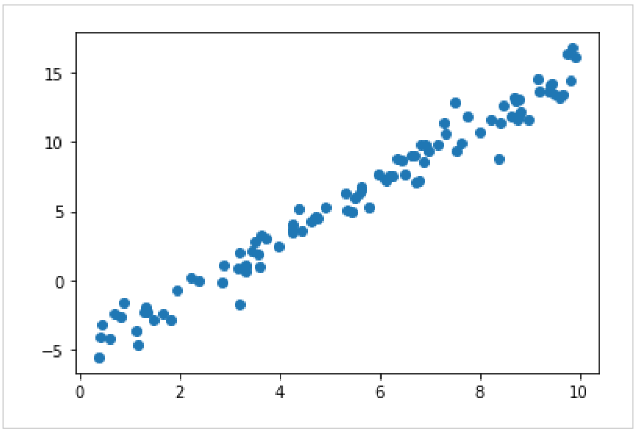
- we have a dataset containing a number of observations of a value x and a corresponding target value y;
- given a new x we want to be able to predict y.

Exercise 1: straight line

We use the linear regression to model a function like:

$$y = a_0 + a_1 x$$

```
n_points=100
x = 10 * np.random.rand(n_points)
y = 2 * x - 5 + np.random.randn(100)
plt.scatter(x, y);
```

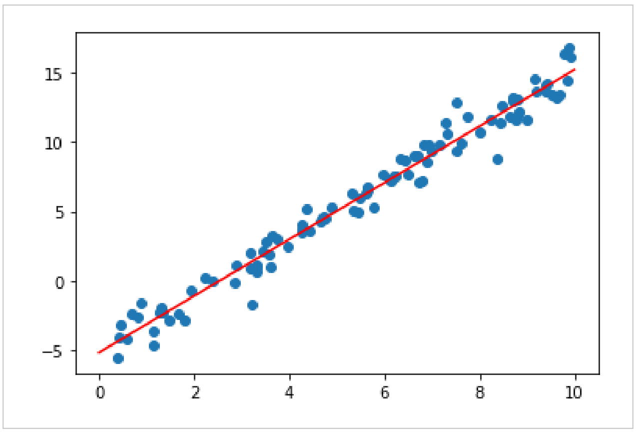


We can use `scikit-learn`'s `LinearRegression` estimator to fit our data and construct the best-fit line

```
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)

# the reshape is necessary: sklearn needs to know if this array contains
    many points
# with a single value or a single point with many values
model.fit(x.reshape(-1, 1), y);
```

We can plot the estimated line together with the training data:

```
xfit = np.linspace(0, 10, 1000)
yfit = model.predict(xfit.reshape(-1,1))

plt.scatter(x, y)
plt.plot(xfit, yfit, color='red');
```



We can also obtain the parameters of our line:

```
print("Model slope: {:.4f}".format(model.coef_[0]))
print("Model intercept: {:.4f}".format(model.intercept_))
```

```
Model slope: 2.0436
Model intercept: -5.2002
```

Exercise 2: non linear relationships

A linear regression could be used also to fit a function like:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots$$

In the name *Linear Regression* the term *linear* refers to the fact that the coefficients a_n never multiply or divide each other.

In order to apply our Linear Regression model in this situation we have to transform our data. In practice we consider a multidimensional linear model:

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + \ldots$$

And we build the x_1, x_2, x_3 from our single dimensional input x: - $x_1 = x$; - $x_2 = x^2$; - …

We can do this really easily using a *transformer* integrated in `scikit-learn`.

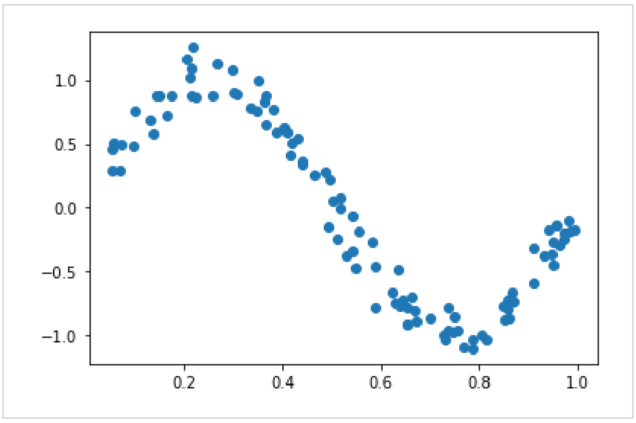
Example of application of the `PolynomialFeatures` transformer:

```
from sklearn.preprocessing import PolynomialFeatures
x = np.array([2, 3, 4])
poly = PolynomialFeatures(4, include_bias=False)
poly.fit_transform(x[:, None])
```

```
array([[  2.,   4.,   8.,  16.],
       [  3.,   9.,  27.,  81.],
       [  4.,  16.,  64., 256.]])
```

In our problem we consider data generated using a sin function with added noise. This is an useful example as there is a global and general regularity (that we wish to learn) but local observations are corrupted by noise.

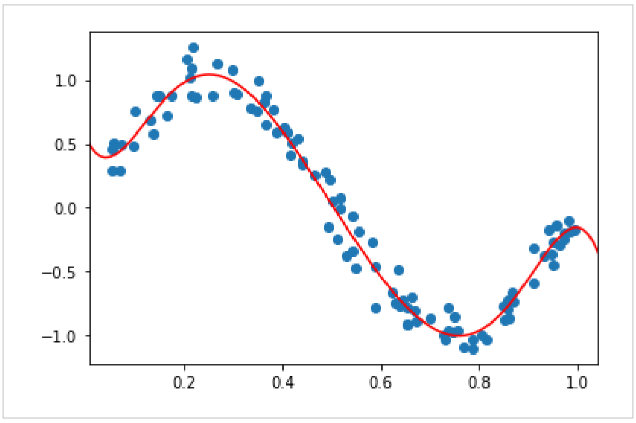
```
x = np.random.rand(n_points)
y = np.sin(2 * np.pi * x) + 0.1 * np.random.randn(n_points)
plt.scatter(x=x, y=y)
plt.show()
```



To solve the problem we want to use both the `PolynomialFeatures` transformer and the `LinearRegression` model. We can use the `make_pipeline` function provided by `scikit-learn` so that we do not need to perform the data transformation and model fitting in two separated steps

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures

poly_model = make_pipeline(PolynomialFeatures(7),
                           LinearRegression())
poly_model.fit(x.reshape(-1,1), y);
```

```
yfit = poly_model.predict(xfit.reshape(-1,1))

plt.scatter(x, y)
lim = plt.axis()
plt.plot(xfit, yfit, color='red');
plt.axis(lim);
```

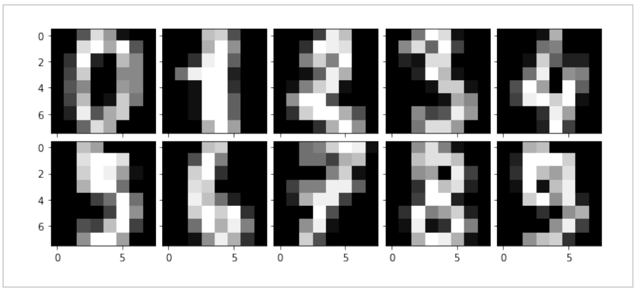


Bonus problem: classification on handwritten Digits using GNB

```
import sklearn
from sklearn.datasets import load_digits
X, y = load_digits(return_X_y=True)
print(X.shape)
```

```
(1797, 64)
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid

fig = plt.figure(figsize=(10., 4.))
grid = ImageGrid(fig, 111, nrows_ncols=(2, 5), axes_pad=0.1)

for idx, ax in enumerate(grid):
    ax.imshow(X[idx].reshape(8,8), cmap='gray')

plt.show()
```



Define **training set** and **test set**.

```
X_train, y_train = X[:-200], y[:-200]
X_test, y_test = X[-200:], y[-200:]
```

Normalize data: many machine learning algorithms work better on standardized data (0 meand and unit variance)

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Instantiate model and **train it**

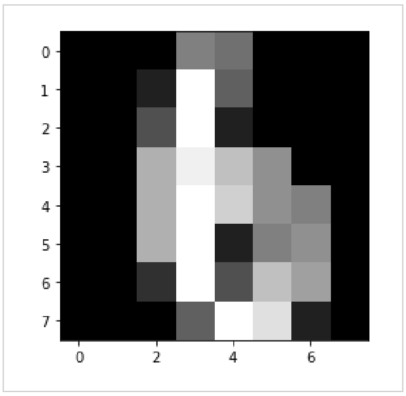
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
```

```
model.fit(X_train, y_train);
```

Test model and **compute accuracy**

```
print(model.predict(X_test[4].reshape(1,-1)))
plt.imshow(scaler.inverse_transform(X_test[4]).reshape(8,8), cmap='gray')
```

```
[6]
<matplotlib.image.AxesImage at 0x7fa618f89b80>
```




```
print("Accuracy:", model.score(X_test, y_test))
```

```
Accuracy: 0.74
```

References:

Python Data Science Handbook

Linear models and Pytorch Datasets