PACKAGES FOR MACHINE LEARNING

The Scikit-learn package

- With the 4 packages studied and with the knowledge we have about Python language, it would already be possible to develop our ML models
  - *however, this would require a very high effort and a deep knowledge of all the complex mathematical foundation that supports the current ML algorithms*
- Fortunately, we don't have to implement them by hand; the ***Scikit-learn*** package provides us with most of these sophisticated – but also complex – algorithms
- Using the ***Scikit-learn*** package, instead of spending an important part of our time replicating algorithms already available and properly consolidated, we will be able to focus our effort on the critical tasks of preparing the datasets (preprocessing tasks), tuning the algorithms (adjusting their main configuration parameters) and evaluating and interpreting the results achieved

The Scikit-learn package

- Now we will see just a brief introduction to the Scikit-learn package, as it will be with it that we will explore all the machine learning algorithms in the important walk that we still have ahead
- In addition to the main algorithms and other essential tools for ML, Scikit-learn also provides us with some datasets that we can use as raw material in experimenting with ML models that we will learn to develop
    - *but it will be in online repositories that we will have a great deal of datasets that we can use for our studies*

Datasets for experimentation

- In the process of developing an ML model, a particular dataset is used to input the model itself
  - *if this model is being developed only in the context of learning ML techniques, with no real data set to be analysed, it is imperative to find datasets with a sufficient volume of data that make experimentation as realistic as possible*
    - some of these datasets are very popular with the Data Science community; we found them in many libraries we are using, as is the case with Seaborn and, in particular, Scikit-learn
    - but, as we said before, it is in cyberspace that we have at our disposal huge repositories of datasets for experimentation

Datasets for experimentation

- For a first foray into Scikit-learn and ML, let's start by analysing one of our most popular datasets
  - *More complete information on all datasets made available by Scikit-learn can be obtained from its official documentation:*

    http://scikit-learn.org/stable/datasets/index.html
- We then proceed with the implementation of a first ML model through the 'Scikit-Learn' library to illustrate all the development phases included in this type of project

The Iris dataset

- One of the most popular datasets among the Data Science community is Iris
- It is a dataset created by the British Ronald Fisher, containing records of three species of iris plants (versicolor, virginica, setosa)
  - *consisting of 150 lines, with the record of the characteristics of 50 specimens of each type of plant*
  - *and by 5 columns, with the length and width of both the petals and the sepals, and the classification of the plant into one of the three species*
- This dataset is widely used to build ML models that, based on the 4 attributes of each specimen, classify the plant into one of the 3 species

Scikit-learn's Iris dataset

- Iris is one of the datasets included in the Scikit_learn
  - *to achieve this, we call the **load_iris()** function of the 'datasets' module*

```
from sklearn import datasets
iris = datasets.load_iris()
```

```
type(iris)
```

```
sklearn.utils.Bunch
```

- The loaded object is not a DataFrame
  - *it is a special object that, having the structure of a dictionary, allows us to access each of its parts through attributes (keys)*

```
dir(iris)
```

```
['DESCR', 'data', 'feature_names', 'filename', 'frame', 'target', 'target_names']
```

- Through the **'feature_names'** attribute, we have access to the column name (features)

```
iris.feature_names
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

- and through the **'date'** attribute to the values of these columns (characteristics of each copy)

```
iris.data
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2
       [4.6
```

```
iris.data.shape
```

```
(150, 4)
```

Dataset *Iris from* Scikit-learn

- With the attribute 'target_names', we can know which types of plants (which are the distinct values of the target variable – categorical values that appear in the last column)

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

- and with the attribute 'target', we access the classification of each copy (values of the last column, in which: 0 - setosa; 1 - versicolor; 2 - virginica)

```
iris.target
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                                      2, 2, 2, 2, 2,
```

- But if necessary, we quickly converted the iris to the DataFrame format

```
import pandas as pd
iris_df = pd.DataFrame(iris.data)
iris_df.columns=iris.feature_names
iris_df['target']=iris.target
```

```
iris_df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
...	...	...	...	...	...
149	5.9	3.0	5.1	1.8	2

Other Scikit-learn datasets

- In addition to Iris, Scikit-learn offers us many other interesting datasets, such as:
  - *breast cancer data (for classification)*

```
ds=datasets.load_breast_cancer()
print('{} linhas e {} colunas.'.format(ds.data.shape[0],ds.data.shape[1]))
```

```
569 linhas e 30 colunas.
```

  - *diabetes data (for regression)*

```
ds=datasets.load_diabetes()
print('{} linhas e {} colunas.'.format(ds.data.shape[0],ds.data.shape[1]))
```

```
442 linhas e 10 colunas.
```

  - *handwritten digit images (for sorting)*

```
ds=datasets.load_digits()
print('{} linhas e {} colunas.'.format(ds.data.shape[0],ds.data.shape[1]))
```

```
1797 linhas e 64 colunas.
```

  - *characteristics of wines from 3 different producers (for classification)*

```
ds=datasets.load_wine()
print('{} linhas e {} colunas.'.format(ds.data.shape[0],ds.data.shape[1]))
```

```
178 linhas e 13 colunas.
```

Small incursion into ML with Scikit-learn

- The easiest way to start in ML is through **linear regression**
  - *it is a linear method that models the relationship between a numerical dependent variable (target variable) and one or more independent variables (explanatory variables)*
  - *is linear because the relationship between independent variables X and dependent variable y is linear:* $y=a+bx_1+cx_2+dx_3+...$

- For example, two variables that are indeed related are the study hours dedicated to a curricular unit and its grade
  - *if we assume that this relationship is approximately linear, we can then develop a linear regression model that helps us capture that relationship*
- In the following exposure, we will then use linear regression to solve the following problem:
  - *based on a history of grades and study hours spent by 15 students (value too low to be minimally realistic) in a given curricular unit, create a model that, based on that history, can predict which grade is expected for a student who will take the exam with 2 weeks of study*

The dataset with notes and hours of study

- Consider that the record of the study hours and the grades of 15 pupils is available

```
horas=[55.1,70.8,29.1,51.1,89.3,89.6,12.6,20.7,5.1,44.1,3.0,45.7,64.9,27.8,67.6]
notas=[8.3,14.8,6.8,11.6,18.6,16.0,4.8,3.4,2.3,9.3,4.5,11.4,11.6,4.4,11.4]
```

```
import matplotlib.pyplot as plt
plt.plot(horas, notas,'ob')
plt.xlabel('tempo de estudo (h)'); plt.ylabel('nota');
```

Note that in this example there is only 1 independent variable, but it could be several (to help explain the grade…)

Typically, Scikit-learn ML algorithms require data in a two-dimensional format

- so we convert our lists to 2D arrays, with 15 rows and 1 column

```
import numpy as np
horas=np.array(horas).reshape(-1,1)
horas.shape
```

```
(15, 1)
```

```
notas=np.array(notas).reshape(-1,1)
horas.shape
```

```
(15, 1)
```

Training data and test data

- An important step in creating an ML model is our training phase
  - *it is at this stage that the model learns from the data, adjusting its configuration automatically*
- In addition to building the model, we will then need to evaluate its effectiveness (its ability to achieve) in some way by comparing our predictions with knowing results
  - *This assessment will be more reliable if it is performed with data that is not used in the training of them,*
  - *that is, the model performance evaluation should be done with completely new data (data that the model has never seen)*
  - *Only in this way it will be able to properly measure its generalization capacity*
- Note that, it is natural for the model, to perform better with the training data, as it has adjusted to them
  - *However, what really matters is our ability to apply and adapt to new data the knowledge we have acquired with training data*
  - *that is, to have the ability to generalize*
- The more training data we use, the greater the generalization capacity of the model
  - *With little training data, the model will adjust too much to this data (overfitting), losing its generalization ability (good performance in training, but bad in the test)*

Partitioning the dataset

- Instead of using all the data available to train our model, we should then put some aside so that with them we will be able to properly test the final model
  - *Fortunately, Scikit-learn contains a function, in its **model_selection** module, which allows us to easily partition our dataset into 2 subsets*

```
from sklearn.model_selection import train_test_split
horas_train, horas_test, notas_train, notas_test = train_test_split(horas, notas, test_size=0.33)
```

As a rule, more data is used for training and less for testing (e.g., between 20 and 30%)

The trainning_test_split() function receives the array with the values of the explanatory variables (hours) and the array with the values of the target variable (notas), and returns each of these arrays unfolded in two, with the data that will be used for training and testing randomly separated in the proportion indicated by parameter test_size (33% for testing)

```
plt.plot(horas_train, notas_train,'ob', label='para treino')
plt.plot(horas_test, notas_test, '.r', label='para teste')
plt.xlabel('tempo de estudo (h)'); plt.ylabel('nota');
plt.legend();
```

Create, adjust, and predict

- By instantiating the LinearRegression class of linear_model scan of Scikit-learn, we create a linear regression model

```
from sklearn.linear_model import LinearRegression
modelo=LinearRegression()
```

- Then, through its fit() method, we train the model with the subsets of the hours and grades we selected for this purpose

```
modelo.fit(X=horas_train, y=notas_train)
```

- Once the model has been created and trained, we can already make predictions through its **predict()** method, finding the answer to the question initially posed

```
modelo.predict([[80]]) #nota prevista para duas semanas de estudo
```

```
array([[15.2]])
```

Not bad, for those who worked so little....

- But as or more important than the forecast itself, we can make reliable forecasts.

```
notas_estimadas=modelo.predict(horas_test)
```

- Only now the data left for testing should be used, putting the model to make the prediction with this data

```
horas_test
array([[89.6],
       [70.8],
       [20.7],
       [29.1],
       [ 3. ]])
```

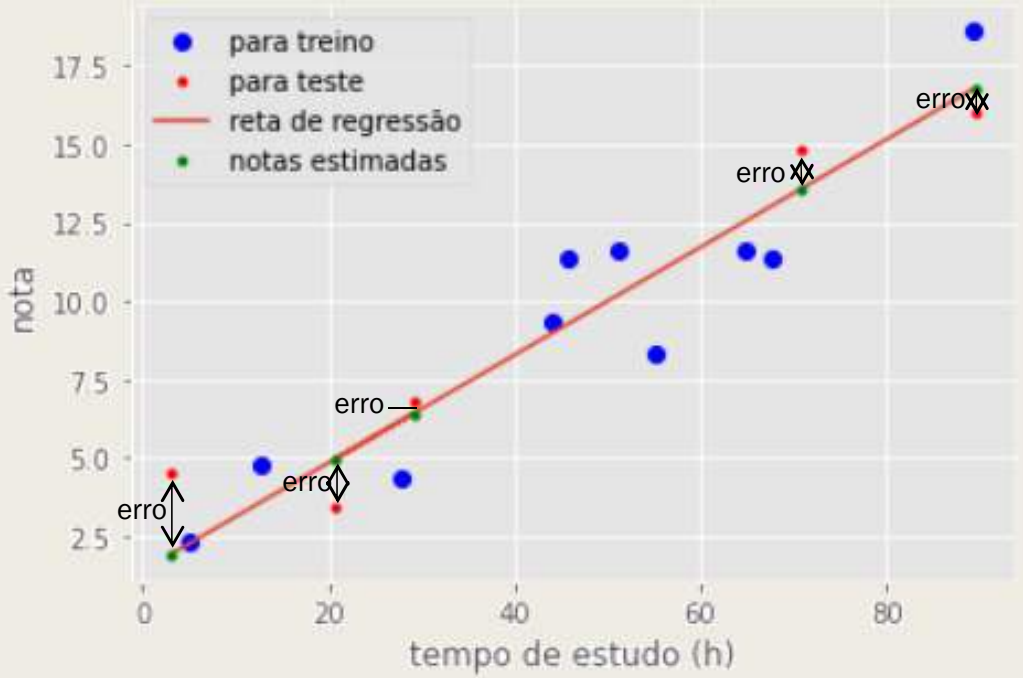
```
notas_test
array([[16. ],
       [14.8],
       [ 3.4],
       [ 6.8],
       [ 4.5]])
```

```
notas_estimadas
array([[16.8],
       [13.6],
       [ 5. ],
       [ 6.4],
       [ 1.9]])
```

Graphical interpretation of model data

- For a better understanding of the linear regression model that was created, we visualise, together with the training and test data, the regression line defined by the model and the grades we plan for the study hours of the training set

```
plt.plot(horas_train, notas_train,'ob', label='para treino')
plt.plot(horas_test, notas_test, '.r', label='para teste')
plt.plot(horas_test, notas_estimadas, label='reta de regressão')
plt.plot(horas_test, notas_estimadas, 'g.', label='notas estimadas')
plt.xlabel('tempo de estudo (h)'); plt.ylabel('nota');
plt.legend();
```



The performance of the model will be the higher the lower the errors made in its forecasts.

There are several ways to account for these errors, which are called metrics

Continue solving exercises #13, #14 e #15

from the book of exercises