

Université d’Alger 1 – Benyoucef Benkhadda

Faculté des Sciences

Département d’Informatique

|  |
| --- |
| Master 1 - ISII |
| Artificial Intelligence |
| Machine Learning Project |

|  |
| --- |
| Team N° 10   * Taleb Mehdi * Sennoun Merouane * Zouai Serine Maria * Aouanouk lmane * Zemmouri Arslane |

|  |
| --- |
|  |

Table of Contents

[1. Introduction 1](#_Toc96803207)

[2. Project Descriptions 1](#_Toc96803208)

[2.1. Regression Problem 1](#_Toc96803209)

[2.2. Classification Problem 1](#_Toc96803210)

[3. Real Estate Valuation 2](#_Toc96803211)

[3.1. Dataset Description 2](#_Toc96803212)

[3.2. Preprocessing 2](#_Toc96803213)

[3.2.1 Graphic Representation 3](#_Toc96803214)

[3.2.2 Missing values 3](#_Toc96803215)

[3.2.3 Remove an unnecessary attribute such as the feature "No” 3](#_Toc96803216)

[3.2.4 Encoding 4](#_Toc96803217)

[3.2.5 Duplicate Values 4](#_Toc96803218)

[3.2.6 Remove outliers 4](#_Toc96803219)

[3.2.7 Feature scaling 5](#_Toc96803220)

[3.3. Linear Regression 6](#_Toc96803221)

[3.4. Normal Equation 8](#_Toc96803222)

[3.4.1 Preparing the X and Y in order to apply the previous formula: 8](#_Toc96803223)

[3.4.2 Splitting the data in train and test datasets 8](#_Toc96803224)

[3.4.3 Calculate the θ 8](#_Toc96803225)

[3.4.4 function calculate-hypothesis 9](#_Toc96803226)

[3.4.5 function Cost 9](#_Toc96803227)

[3.5. Neural Network 10](#_Toc96803228)

[3.5.1. Splitting the Dataset 10](#_Toc96803229)

[3.5.2. Tuning parameters for the model 10](#_Toc96803230)

[3.5.3. Final Results 11](#_Toc96803231)

[4. Immunotherapy Treatment Result 12](#_Toc96803232)

[4.1. Dataset Description 12](#_Toc96803233)

[4.2. Preprocessing 12](#_Toc96803234)

[4.2.1 Graphic Representation 12](#_Toc96803235)

[4.2.2 Missing values 12](#_Toc96803236)

[4.2.3 Removing Categorical values 13](#_Toc96803237)

[4.2.4 Detecting Outliers 14](#_Toc96803238)

[4.2.5 Feature scaling 14](#_Toc96803239)

[4.3. Logistic Regression 15](#_Toc96803240)

[4.3.1 Splitting the dataset 15](#_Toc96803241)

[4.3.2 definition of the functions 15](#_Toc96803242)

[4.3.3 Implementation of the Gradient Descent 15](#_Toc96803243)

[4.3.4 Testing different values of alpha 16](#_Toc96803244)

[4.4. Neural Network 17](#_Toc96803245)

[4.4.1. Splitting the dataset 17](#_Toc96803246)

[4.4.2. Tuning parameters for the model 17](#_Toc96803247)

[4.4.3. Final Results 18](#_Toc96803248)

[4.4.4. Confusion Matrix - Plotting the results 18](#_Toc96803249)

[4.5 SVM 19](#_Toc96803250)

[4.5.1 Train and Test Datasets 19](#_Toc96803251)

[4.5.2 Graphical representation 19](#_Toc96803252)

[4.6 Decision Trees 20](#_Toc96803253)

[4.6.1 Train and Test Datasets 20](#_Toc96803254)

[4.6.2 Building the Decision Tree 20](#_Toc96803255)

[4.6.3 Feature importance in the Decision Tree 20](#_Toc96803256)

[4.6.4 Graphical representation 21](#_Toc96803257)

[4.7 Comparing models 21](#_Toc96803258)

[Appendix 1](#_Toc96803259)

[A.1 Detailed set of loops example (Neural Network for Classification) 1](#_Toc96803260)

[A.2 GridSearchCV implementation (Neural Network for Classification) 2](#_Toc96803261)

[A.2.1 Initializing the algorithm 2](#_Toc96803262)

[A.2.2 The final results 2](#_Toc96803263)

[A.3 Confusion matrix and results plotting code (Neural Network for Classification) 3](#_Toc96803264)

[B.1 Regularization in Normal Equation 4](#_Toc96803265)

[C.1 Accuracy function 5](#_Toc96803266)

[C.2 Accuracy test with predefined function 5](#_Toc96803267)

[References 6](#_Toc96803268)

**Division of work**

**Taleb Mehdi:**

[Preprocessing](#_Toc96800523) (Classification)

[SVM](#_Toc96800560) (Classification)

[Decision Trees](#_Toc96800563) (Classification)

**Sennoun Merouane:**

[Preprocessing](#_Toc96800523) (Classification)

[Logistic Regression](#_Toc96800550) (Classification)

[SVM](#_Toc96800560) (Classification)

**Zouai Serine Maria:**

[Preprocessing](#_Toc96800523) (Regression)

[Normal Equation](#_Toc96800532) (Regression)

**Aouanouk lmane:**

[Preprocessing](#_Toc96800523) (Regression)

[Linear Regression](#_Toc96800531) (Regression)

**Zemmouri Arslane:**

[Introduction](#_Toc96800517) + [Project Descriptions](#_Toc96800518)

[Neural Network](#_Toc96800538) (Classification)

[Neural Network](#_Toc96800538) (Regression)

# Introduction

The project that we had to undertake is based on a subcategory of machine learning and artificial intelligence, which is Supervised Learning. Essentially, supervised learning is when the computer is taught by example. It learns from past data and applies the learning to present data to predict future events. In this case, both input and desired output data provide help to the prediction of future events.

Supervised learning can also be divided into two subcategories, classification and regression. Which are the two sorts of problems that we had to operate.

We had to apply the specific analysis methods to each category, to solve the two problems and train different models capable of predicting the best possible results.

On the other hand, we also had to comply with specific working methods to solve a machine learning problem. The use of a notebook is therefore essential for this, because it is used to present the analysis process step by step by arranging the code, images, text, output etc. in a step-by-step manner. Jupyter Notebook provides a full set of features that makes it one the best components of Python Machine Learning. Another tool that we used, this time for the arrangement and organization of work in general, is Git and GitHub, such as each member of the group made changes on the part of the notebook that corresponded with a specific part of the project (preprocessing, graphic visualization etc.)

In the following parts, we are going to discuss the achieved tasks in each of the two problems, explaining the dataset content, the different steps of preprocessing realized to improve its quality, and testing and visualizing the obtained results produced from the built models.

# Project Descriptions

## Regression Problem

One of the two problems that we had to process is a regression problem. A regression analysis must be considered when we deal with a problem that requires a prediction of a continuous value. Our regression problem is about valuating real estate properties given certain characteristics about them and their surroundings. Basically, the main task is to predict a house price, which is continuous value, considering a group of numerical/categorical features describing this house.

## Classification Problem

The second problem is a classification problem. This is the kind of problems that require the result of the prediction to be a categorical value. In our case it is a binary classification, for the reason that the model can only predict a 0 or 1 value for a given set of feature values concerning an instance. Our classification problem is about predicting if Immunotherapy, when used for wart treatment, produces a positive or negative result on a patient, given different information about this one. A prediction of “1” being a result of a successful treatment and “0” of an unsuccessful one.

# Real Estate Valuation

## Dataset Description

The real estate valuation is a regression problem. The market historical data set of real estate valuation are collected from Sindian Dist., New Taipei City, Taiwan.

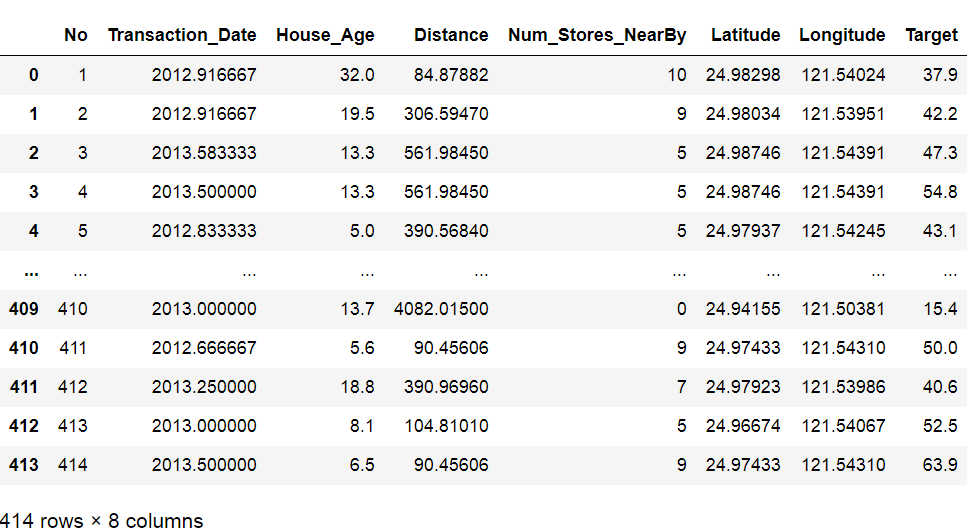
The data set was randomly split into the training data set (2/3 samples) and the testing data set (1/3 samples).

**Attribute Information:**

The inputs are as follows:  
X1=the transaction date (for example, 2013.250=2013 March, 2013.500=2013 June, etc.)  
X2=the house age (unit: year)  
X3=the distance to the nearest MRT station (unit: meter)  
X4=the number of convenience stores in the living circle on foot (integer)  
X5=the geographic coordinate, latitude. (Unit: degree)  
X6=the geographic coordinate, longitude. (Unit: degree)  
  
The output is as follows:  
Y= house price of unit area (10000 New Taiwan Dollar/Ping, where Ping is a local unit, 1 Ping = 3.3 meter squared)

## Preprocessing

We start our preprocessing by displaying our dataset df1 and using the Pandas method as follows:

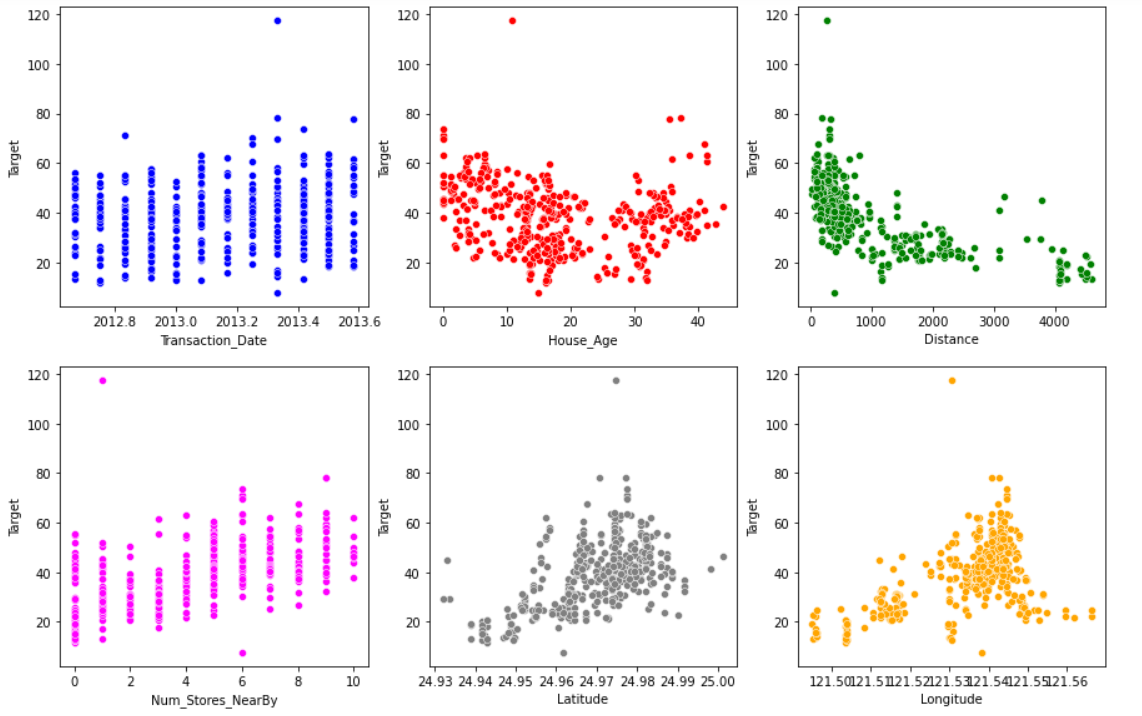


The shape of our Dataset df1 is:



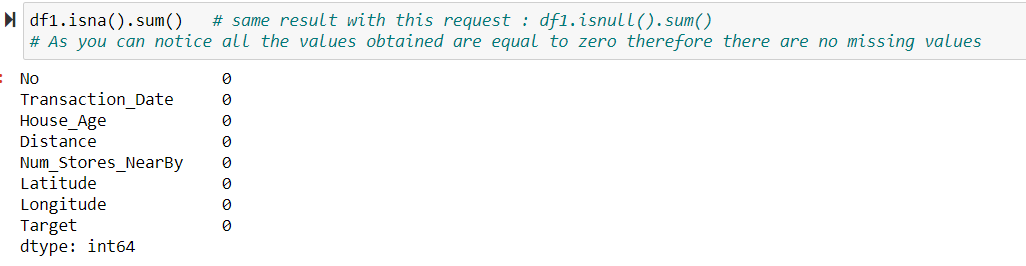
### 3.2.1 Graphic Representation

We use Scatter subplot for each feature as follows:

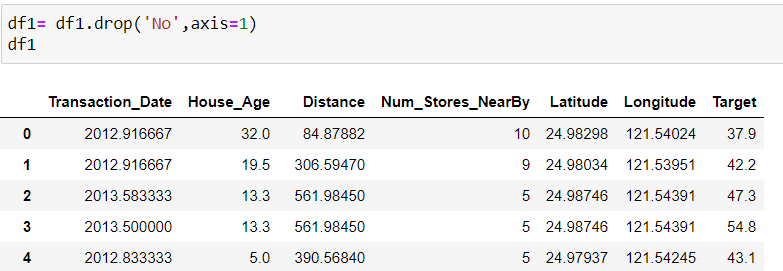


### 3.2.2 Missing values

In the description of the regression dataset, it is mentioned that there is no missing values and there is a command that allows us to determine the number of missing values : df1.isna().sum()

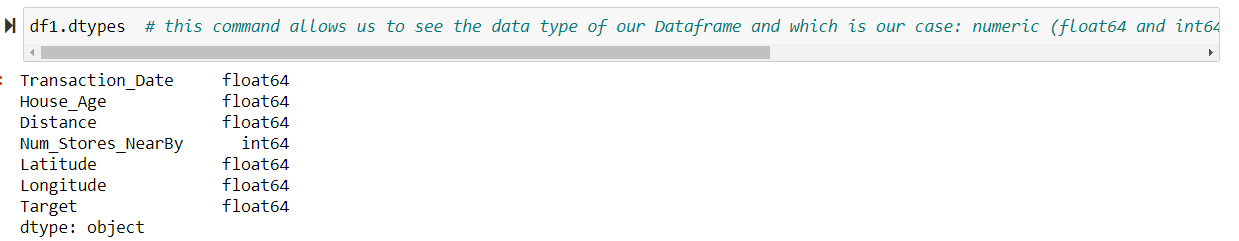


### 3.2.3 Remove an unnecessary attribute such as the feature "No”

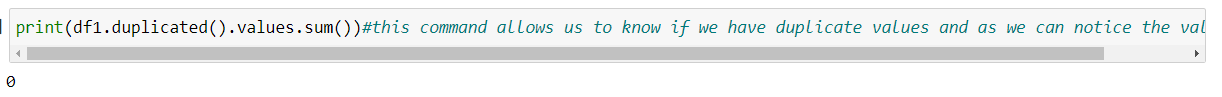


### 3.2.4 Encoding

We show the type of our data by the command df1.dtypes and we found that the dtype of all our features is not categorical.

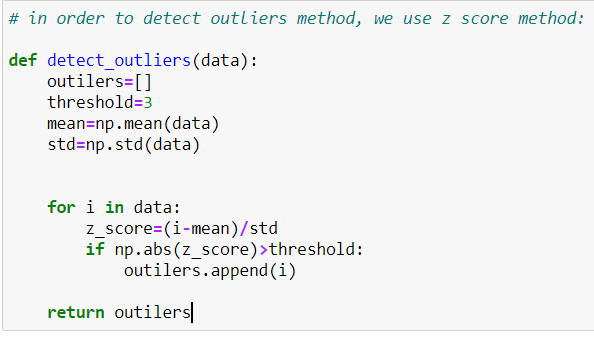
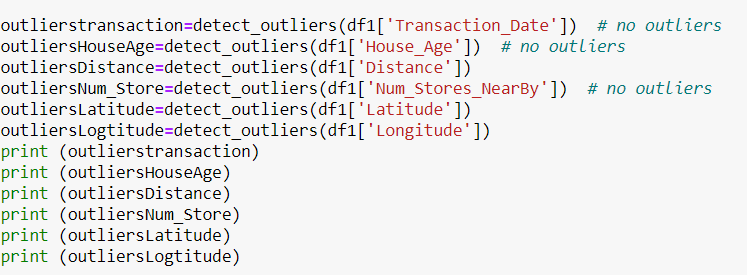


### 3.2.5 Duplicate Values



### 3.2.6 Remove outliers

We use the z\_score method to detect outliers:

Result:

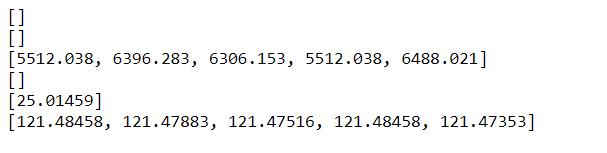
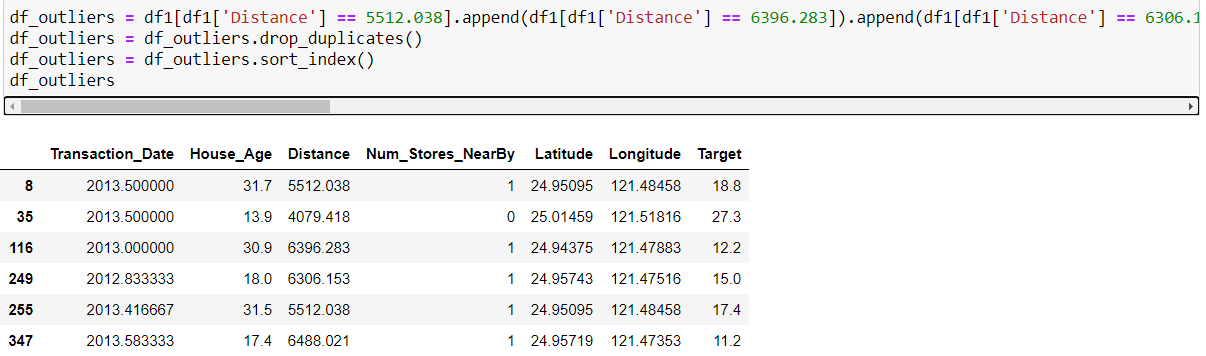
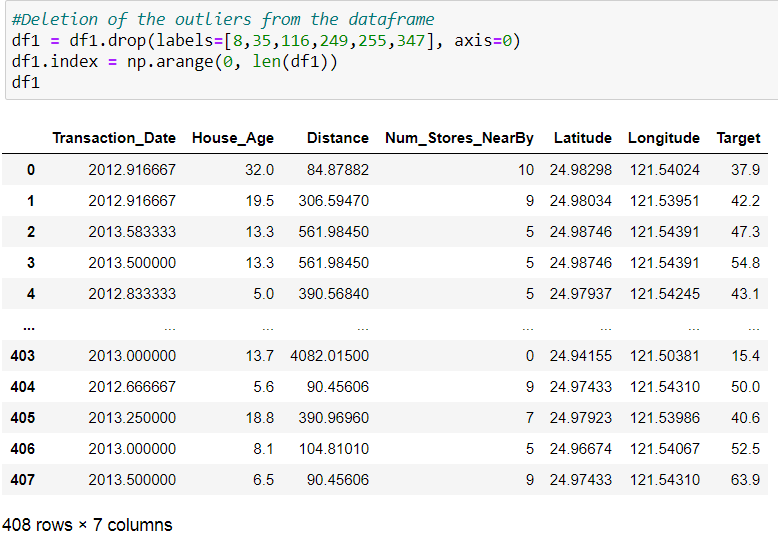


Table of outliers of our data df1:

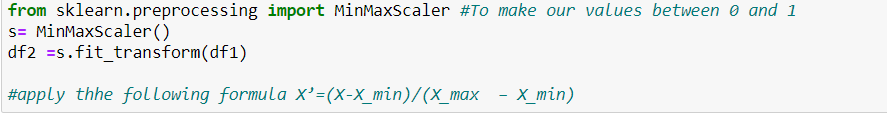


Df1 after removing outliers:

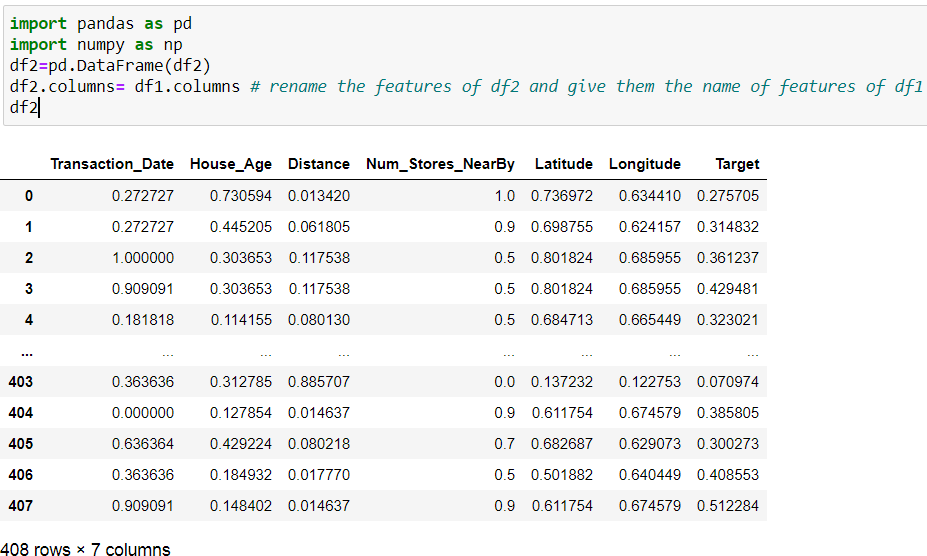


### 3.2.7 Feature scaling

In order to make our values in one interval [0 1] , we use the feature scaling method MinMaxScaler :



Df1 after scaling:

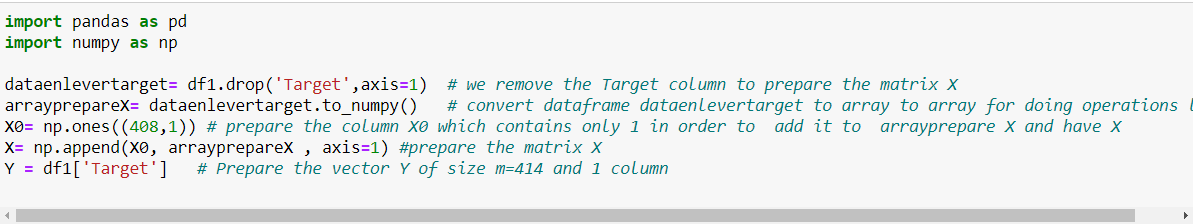


## Linear Regression

## Normal Equation

The Normal equation is a method that allows us to find the Teta θ analytically and will give us a much better way to solve for the optimal value of θ in our regression problem. We don’t need to choose **α** and we don’t need for feature scaling. We just need to calculate: inv(X.T \* X) \* X.T\*Y.

### 3.4.1 Preparing the X and Y in order to apply the previous formula:

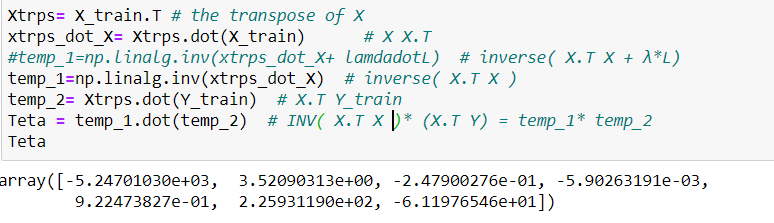


### 3.4.2 Splitting the data in train and test datasets



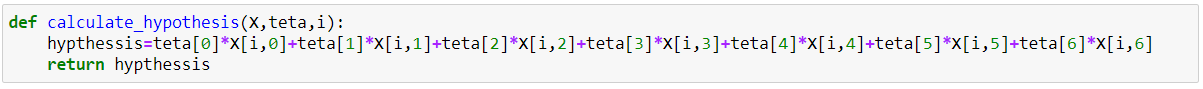
After many tests, we found that the test\_size=0.44 give us the best value of cost.

### 3.4.3 Calculate the θ



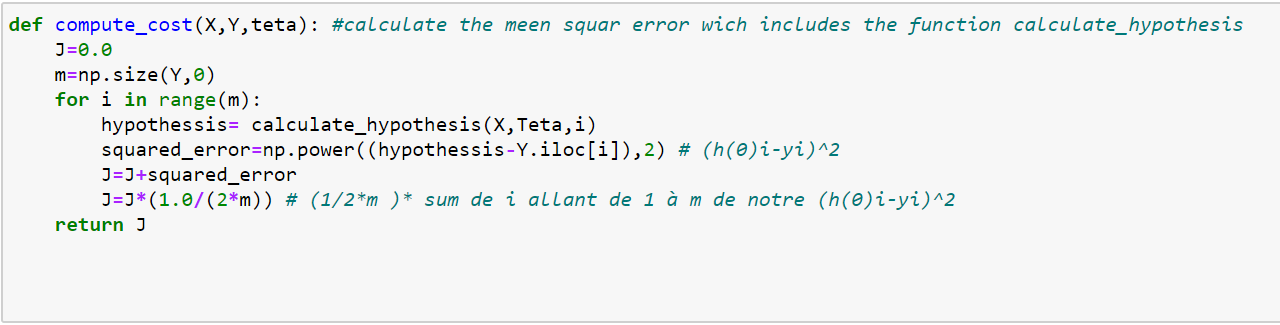
### 3.4.4 function calculate-hypothesis

In order to calculate the hypothesis, which are used later in order to calculate the mean-square error (cost), we implement this method:

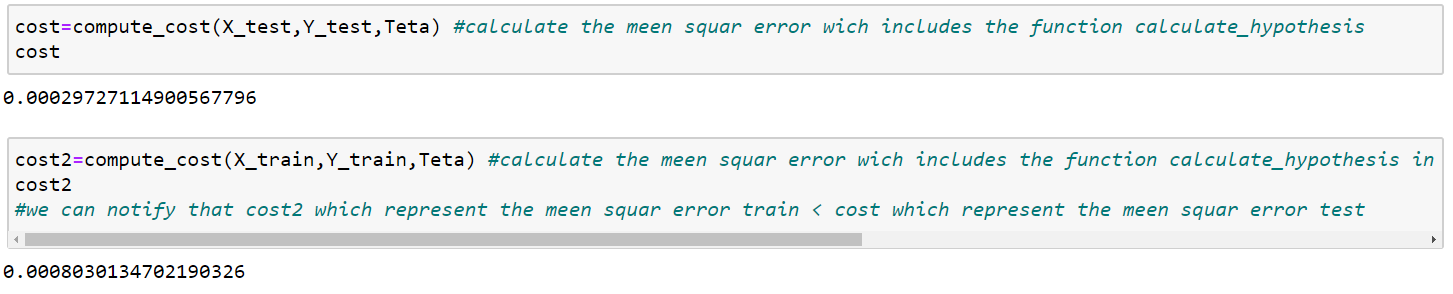


### 3.4.5 function Cost

Now, we will calculate the cost between the Y\_test and the prediction of our X\_test using the previous function



We will now calculate the cost\_test and the cost\_train in order to compare between the two values and see if we have the possibility to have overfitting in our dataset .



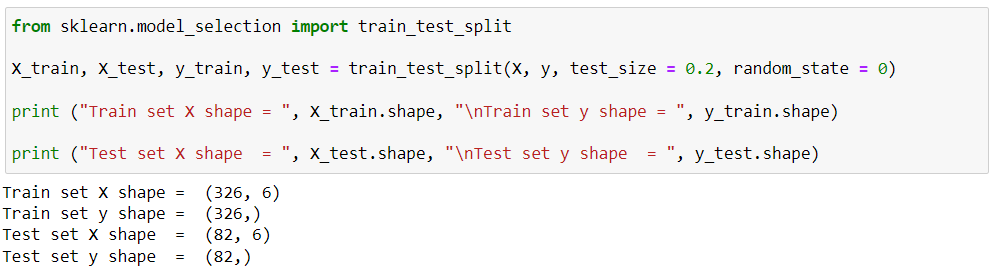
As we can see , the cost(X\_test, Y\_test, Teta) < cost (X\_train, Y\_train, Teta) , so we have a good result in our cost\_test and it is not just in cost\_train .

**NB :**

<you find the regularization in Appendix B.1 Regularization in Normal Equation >

## Neural Network

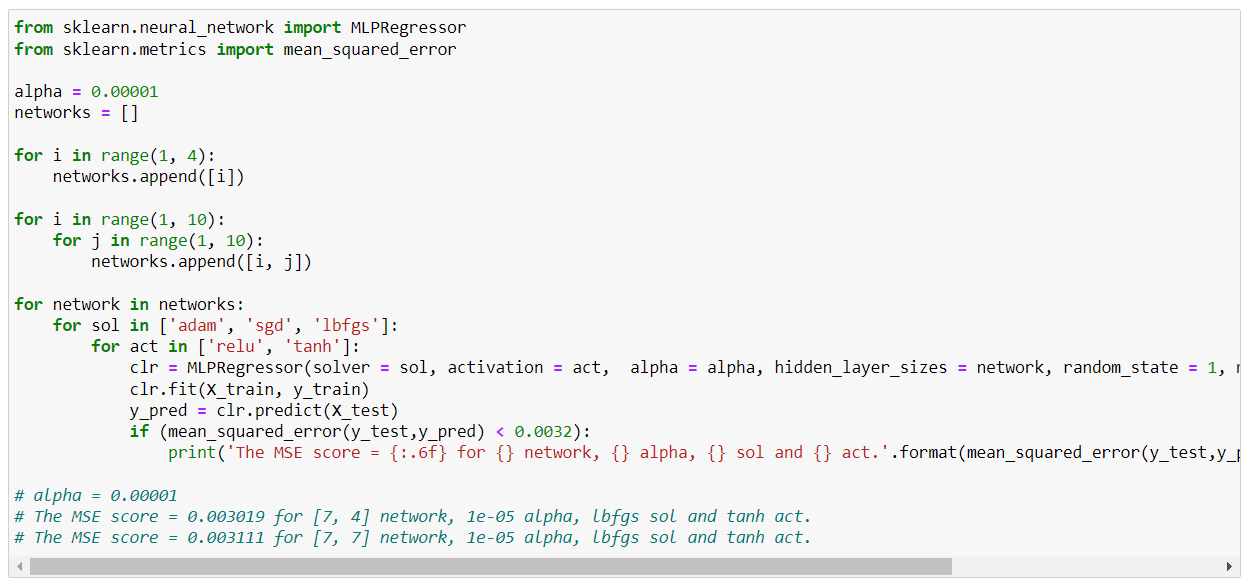
### Splitting the Dataset



### Tuning parameters for the model

The goal of this part of the process is to find the best parameters for the multi-layer perceptron regressor so it can predict the best possible continuous output data based on input data (number of layers and neurons, alpha value etc.)

To do that, we used for a couple first tests a set of loops that go through some values of different parameters that the model can use.

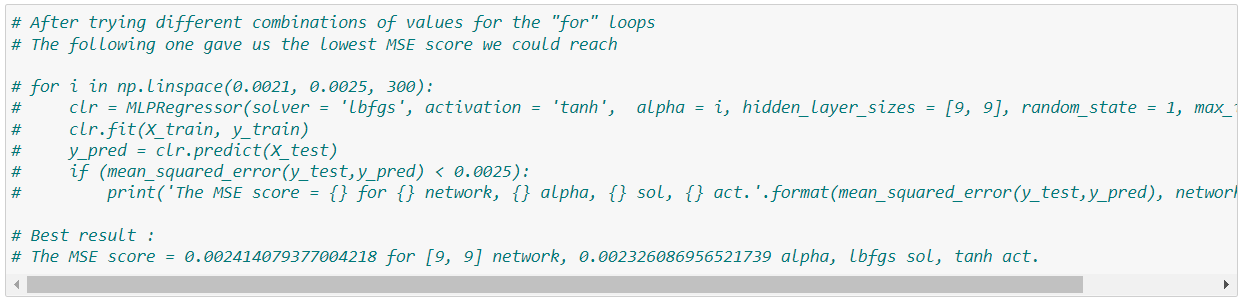


After several tests with this method, we noticed after that :

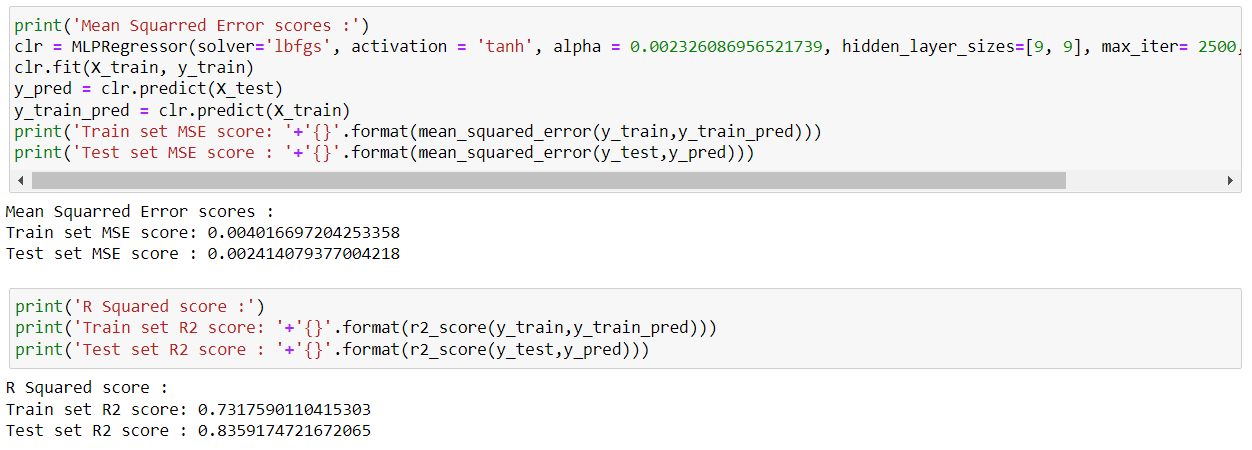
* Logistic activation function does not produce any interesting results at all no matter the chosen alpha values (0.00001 etc.), we had to remove it for the remaining tests.
* All three solver can produce relatively similar results, so we must try testing all three solvers with more larger sets of values for the rest of the parameters.

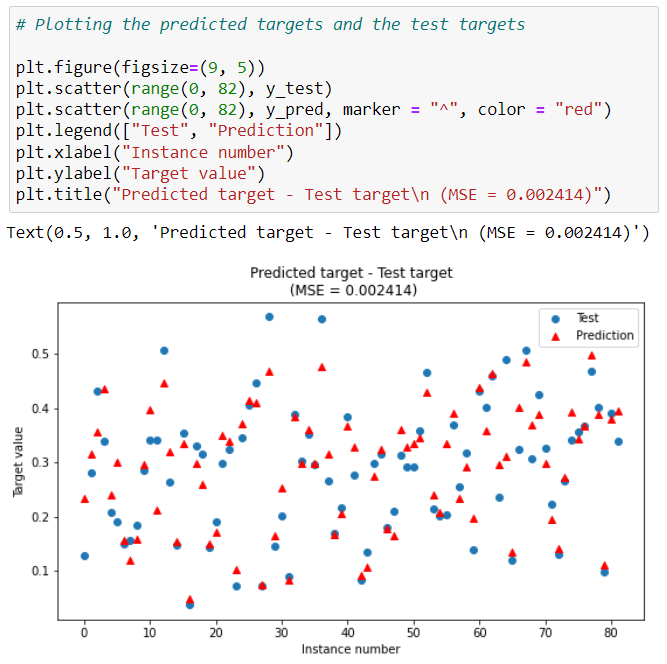
We had to keep testing all solvers with more precise parameters (alpha, network, with “relu” and “tanh” activation function).

After several tests, we managed to find the lowest mean square error value (0.002414) we could reach, which is an acceptable value, and a relatively good r-squared (0.08359), for a [9, 9] neural network structure, with the "lbfgs" solver and the "tanh" activation function. We also notice that the model



### Final Results





In conclusion, our Neural Network model gives us a Mean Squared Error value of 0.002413, and a R-Squared value of 0.835917.

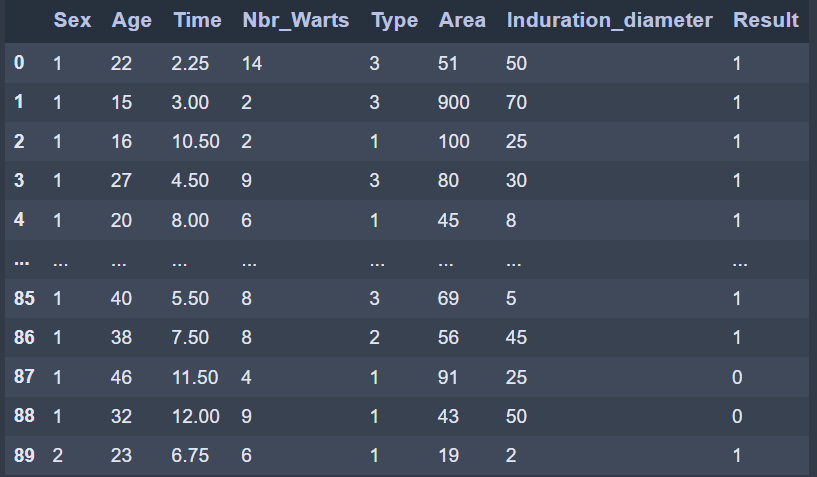
# Immunotherapy Treatment Result

## Dataset Description

This dataset contains information about wart treatment results of 90 patients using immunotherapy.

## Preprocessing

The imported dataset df:



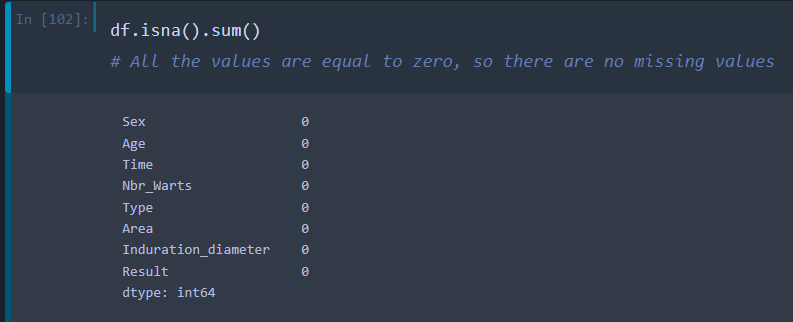
### 4.2.1 Graphic Representation

We represent all the 7 features in comparison to ‘Result’ in 7 subplots:



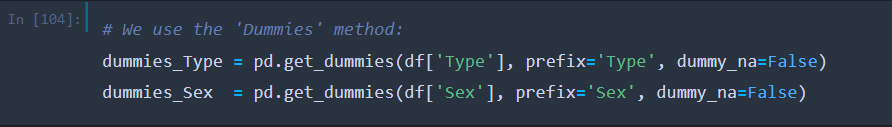
### 4.2.2 Missing values

We use the command **isna().sum()** to check if there are any missing values

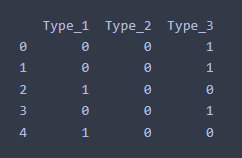
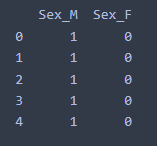


### 4.2.3 Removing Categorical values

The features 'Sex' and 'Type' are of Categorical type, we have to transform them to a numerical type (int) using the 'Dummies' method:

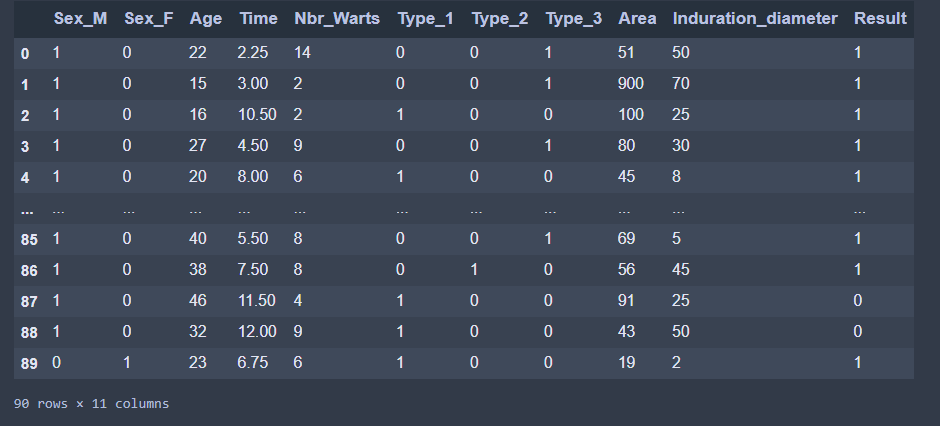


‘Type’ dummies: ‘Sex’ dummies:

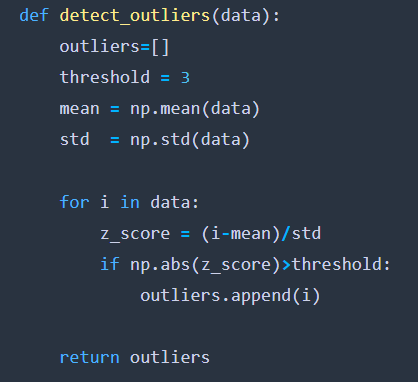
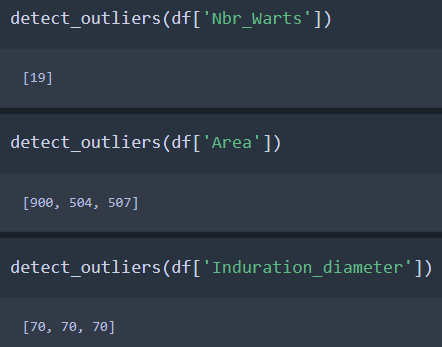
 

-We remove the features ‘Type’ and ‘Sex’ from the initial dataset df.

-We merge all the features to get our new and processed dataset:

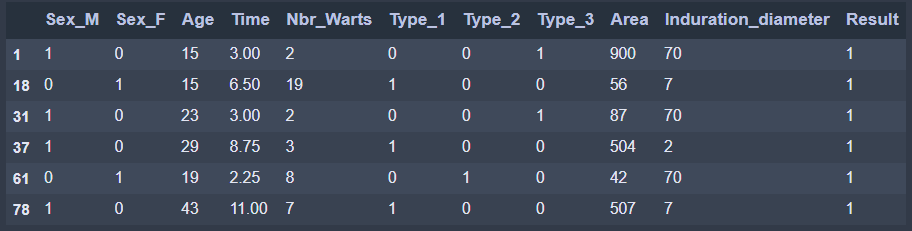


### 4.2.4 Detecting Outliers

We use the **‘z\_score’** method to detect the outliers:

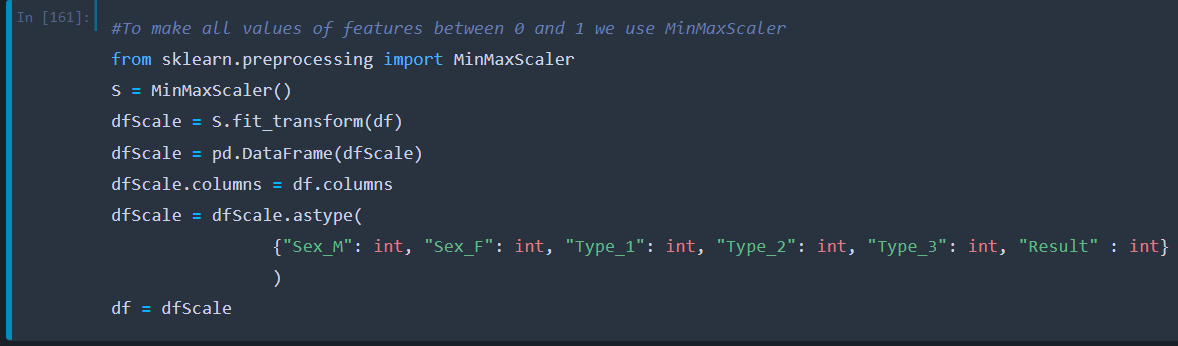
**Remark:** we checked all the other features and they don’t contain any outliers.

Table of all the outliers of ‘Nbr\_Warts’, ‘Area’ and ‘Induration\_diameter’:

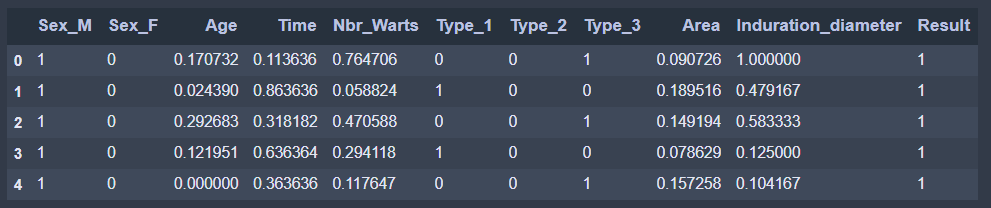


### 4.2.5 Feature scaling

To make all values of features between 0 and 1 we use MinMaxScaler from sklearn.preprocessing



Df after the feature scaling: (and this is also the final result after all of the pre-processing methods)



## Logistic Regression

### 4.3.1 Splitting the dataset

After splitting the dataset, we get the following shapes:

Une image contenant texte

Description générée automatiquement

### 4.3.2 definition of the functions

To be able to perform the Gradient Descent, we need to implement the following functions first.

Une image contenant texte

Description générée automatiquement

### 4.3.3 Implementation of the Gradient Descent

Now we use the functions we defined previously to perform the Gradient Descent and train our model using the train part of the dataset. The Gradient will perform a set amount of times “iterations” the calculations shown below in order to reduce the cost and give the best values of theta.

Une image contenant texte

Description générée automatiquement

### 4.3.4 Testing different values of alpha

The goal here is to minimize the cost, so we tried different values of “alpha” and “iterations” in order to get the best result possible and found that alpha = 0.6 is the best value for our dataset. If we go further than 0.6, we will no longer see a big decrease in the final value of the cost.

Une image contenant texte

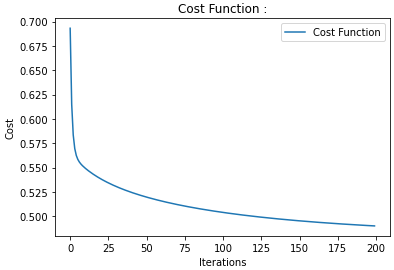
Description générée automatiquement

We can see below the decrease of the cost value while training the model.

Une image contenant texte

Description générée automatiquement

And here is the plot of the cost function.



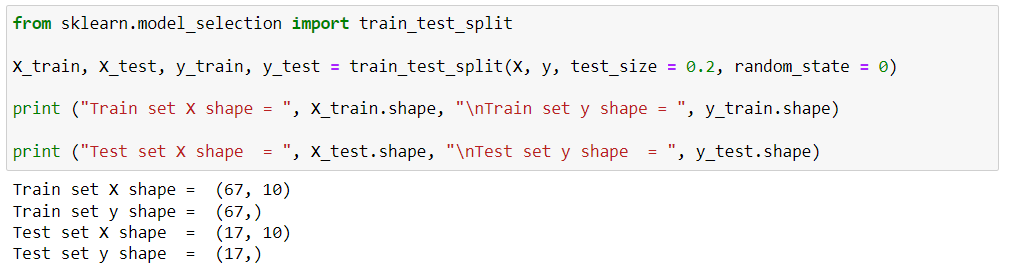
At the end, we get an accuracy of 77.77% when predicting the test set, which is a good value. (Picture of the accuracy function in the appendix)

We can now use the predefined function in “sklearn” to compare our results, and we can see that we got an identical accuracy of 77.77%. (Picture of the code function in the appendix)

## Neural Network

### Splitting the dataset

Train sets and tests sets are split as follows:



### Tuning parameters for the model

The goal of this part of the process is to find the best parameters for the multi-layer perceptron classifier so it can predict the best possible output data based on input data (number of layers and neurons, alpha value etc.).

To do that, we used for a few first tests a set of loops that go through some values of different parameters that the model can use.

* A more detailed version of this section of code is to be found in the appendix A.1.

****

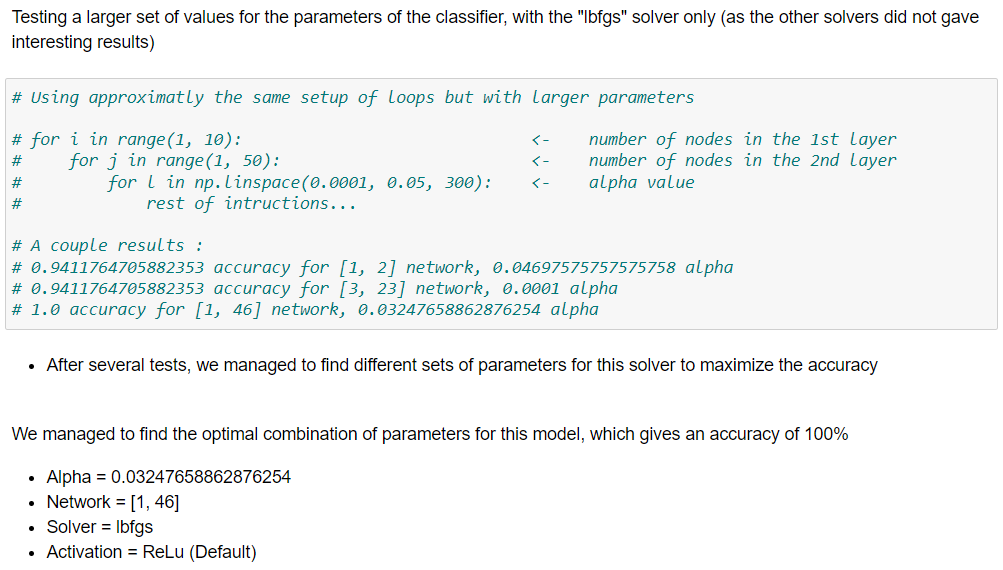
After a few tests with this method, we noticed that:

* We can use “relu” activation function as a parameter for the classifier for the remaining tests.
* No matter the alpha value chosen manually, we can reach a maximum accuracy value of 94.11%, we must add a loop to test a larger set of alphas to try to increase the accuracy.

We must keep testing all solvers with more precise parameters (alpha, network, but the same “relu” activation function).

After several tests, we noticed that the “adam” and “sgd” solvers could not give good results no matter the range of parameters we fitted into the “for” loops.

**Conclusion:** In the final executions, we managed to find an optimal set of parameters for the classifier so it can give an accuracy of 100%, as follows:



**GridSearchCV:**

We tried to optimize the model parameters using the GridSerachCV algorithm.

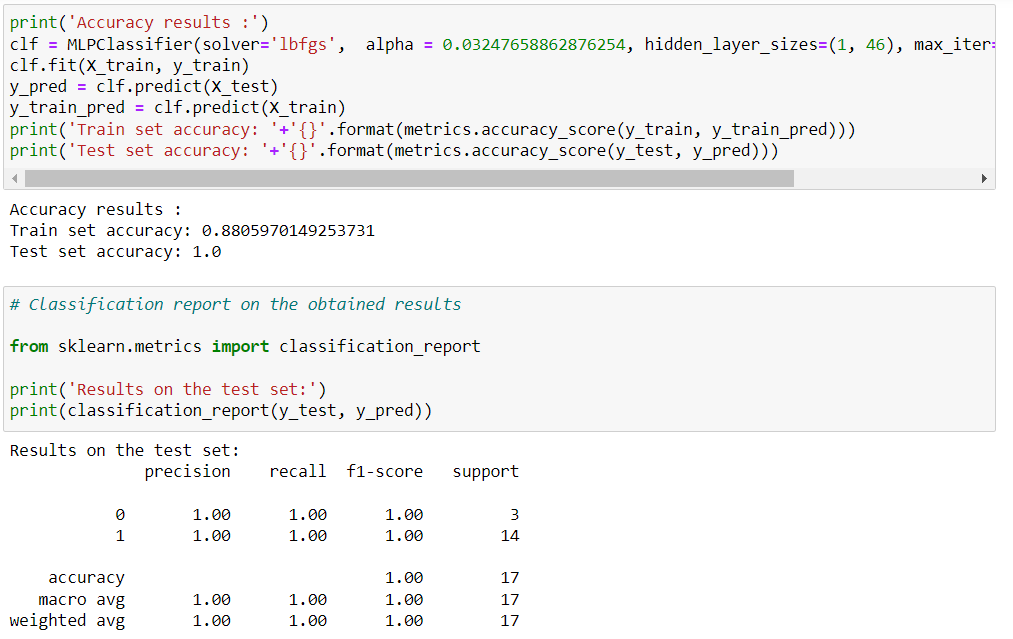
In brief, it does the process of performing parameter tuning in order to determine the optimal values for a given model, as we pass predefined values for parameters to the GridSearchCV function by defining a dictionary in which we mention a particular parameter along with the values it can take.

We initialized the searching algorithm by giving it the model we’re trying to optimize, a parameter space that contains different values for the different parameters that the model can have, and other parameters. After the search, the algorithm returned the best parameters it could find.

We concluded that using this algorithm was not an interesting way to find the optimal parameters for our model, as the accuracy produced with the best parameters the algorithm could find fitted in the mlp classifier is lower as the accuracy we found with the method above.

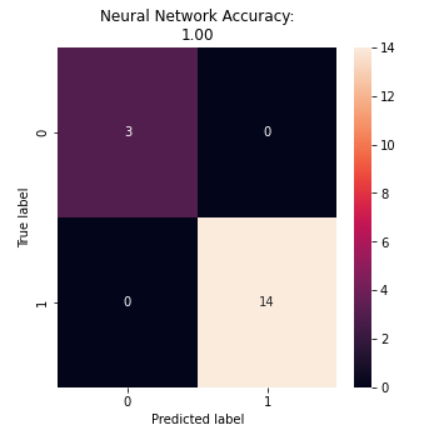
* All the details of this part are to be found in the appendix A.2.

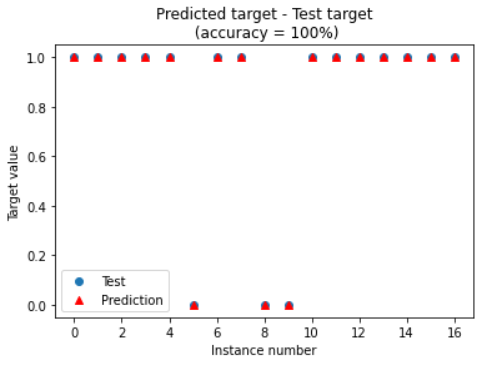
### Final Results



### Confusion Matrix - Plotting the results

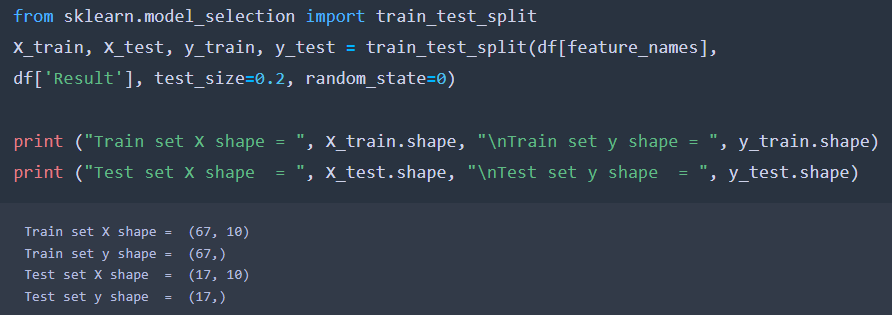
* Code details in the appendix A.3.





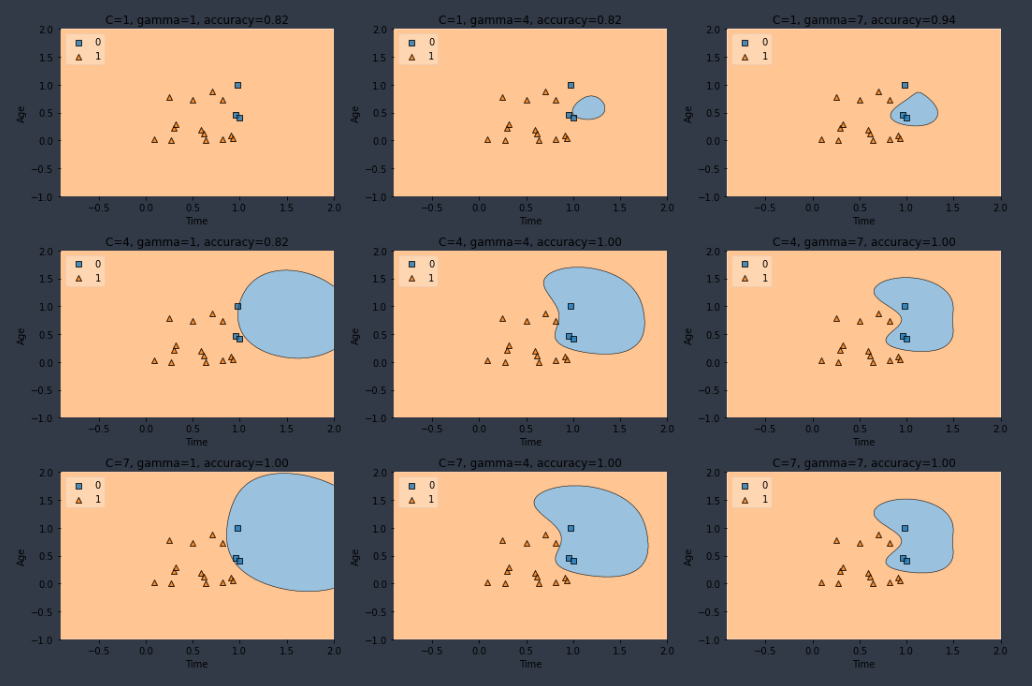
## 4.5 SVM

### 4.5.1 Train and Test Datasets

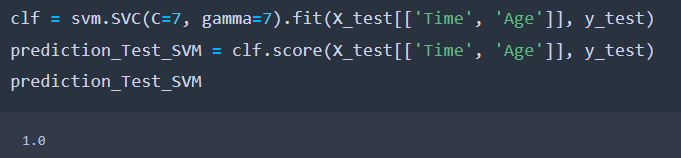


### 4.5.2 Graphical representation

We plot the following graph using the kernel ‘rbf” and ‘plot\_decision\_regions’:

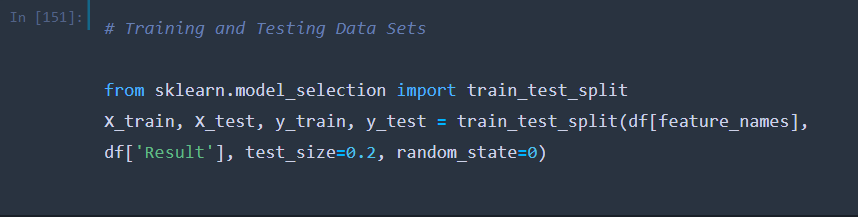


We found multiple combinations of C and gamma that resulted in 100% train prediction accuracy, we chose C=7 and gamma=7:



## 4.6 Decision Trees

### 4.6.1 Train and Test Datasets

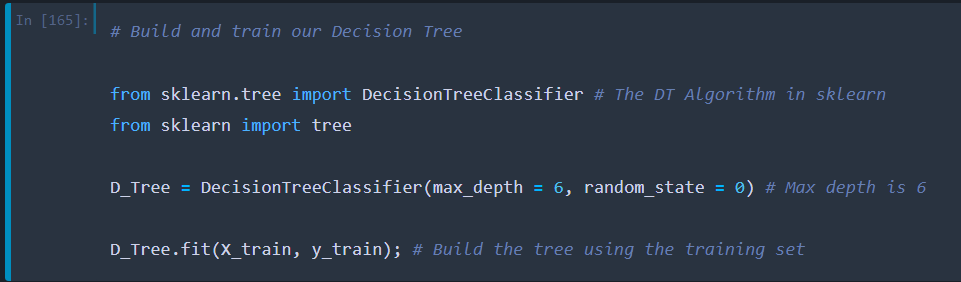


After trying many test-sizes, 0.2 gave the best test accuracy:

|  |  |  |
| --- | --- | --- |
| **Test Size** | **Train Accuracy** | **Test Accuracy** |
| 0.1 | 98.67 % | 88.89 % |
| 0.15 | 98.59 % | 92.31 % |
| 0.2 | 100 % | 94.12 % |
| 0.25 | 100 % | 80.95 % |
| 0.3 | 98.28 % | 92.31 % |
| 0.35 | 98.15 % | 93.33 % |
| 0.4 | 100 % | 91.18 % |

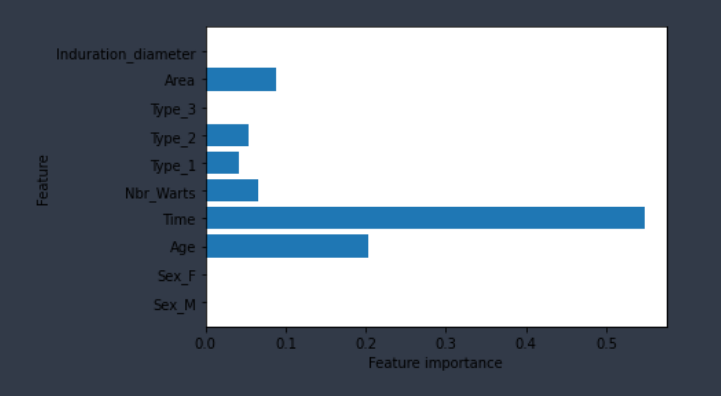
### 4.6.2 Building the Decision Tree

We found that the best max\_depth is 6.



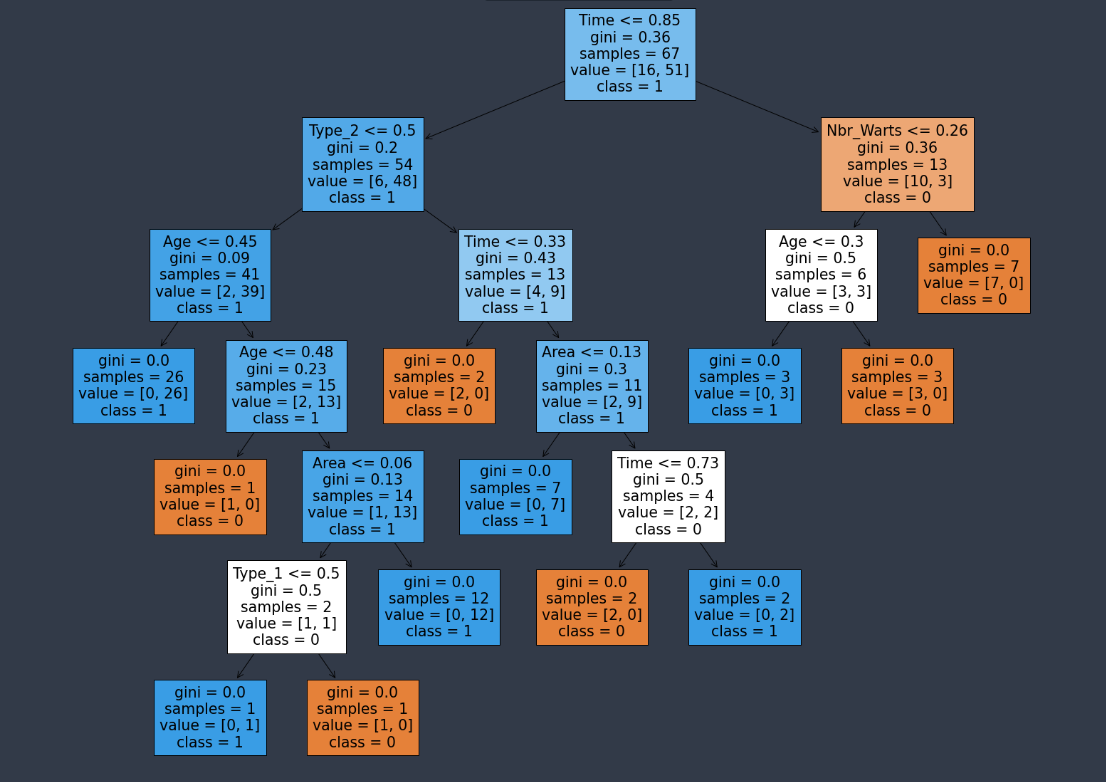
### 4.6.3 Feature importance in the Decision Tree

We used D\_Tree.feature\_importances and plt.barh() to create this graph :

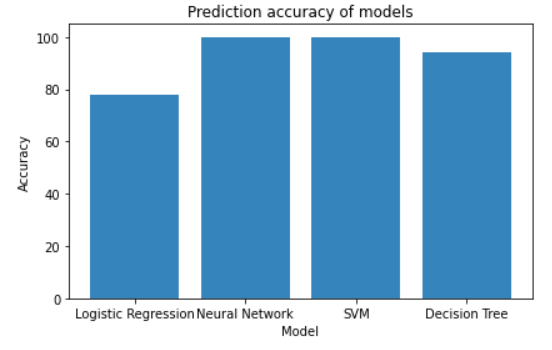


We notice that ‘Time’, ’Age’ and ‘Area’ are the most important features, while ‘Sex\_F’, ‘Sex\_M’ and ‘Type\_3’ are not used.

### 4.6.4 Graphical representation



## 4.7 Comparing models



We conclude that SVM and Neural Network models gave the best test prediction accuracy (100%).

# Appendix

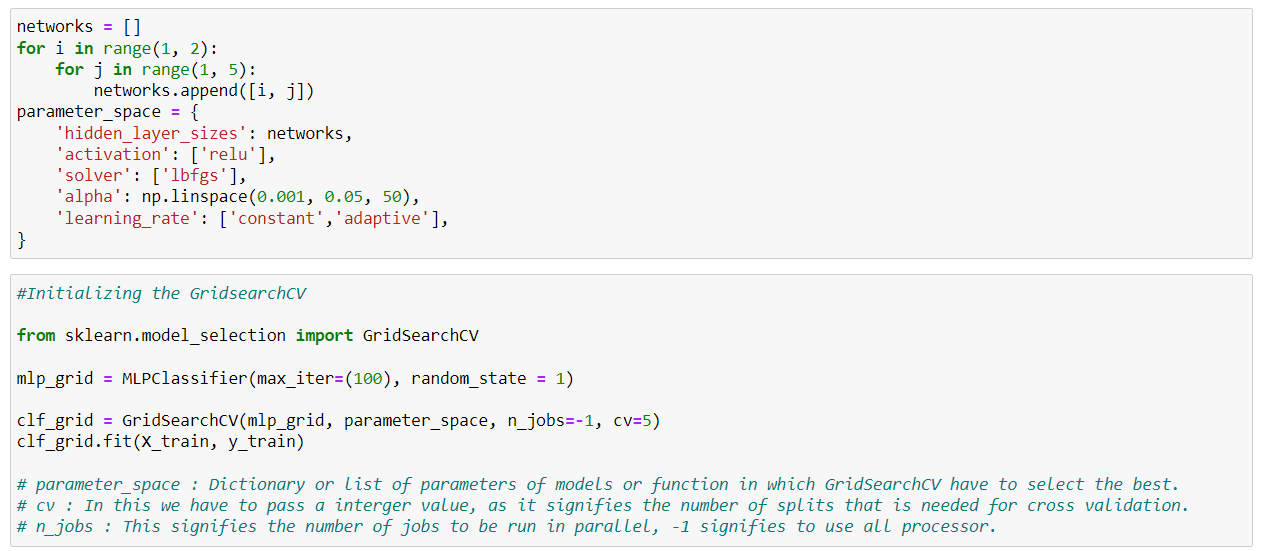
## A.1 Detailed set of loops example (Neural Network for Classification)



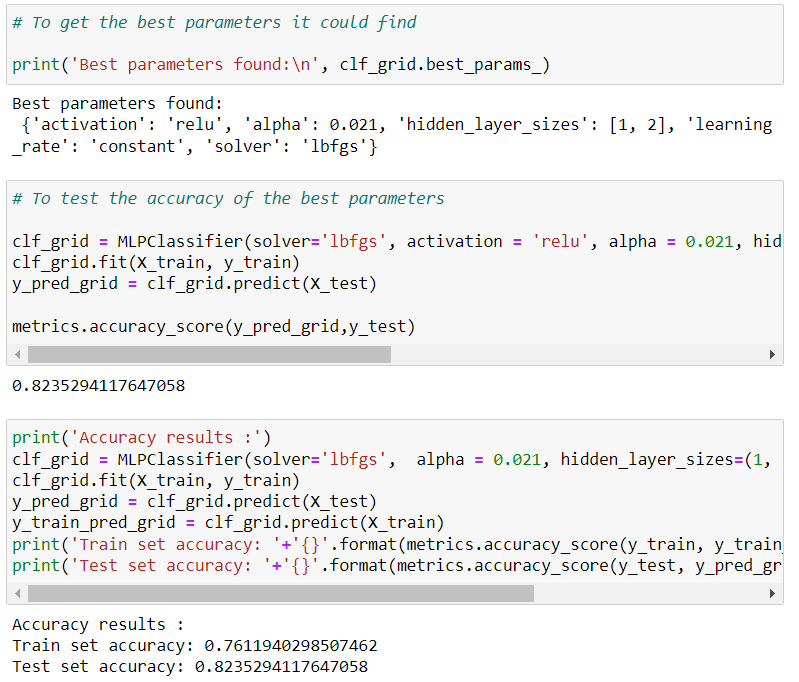
## A.2 GridSearchCV implementation (Neural Network for Classification)

### A.2.1 Initializing the algorithm

As mentioned in GridSearchCV part of the report (4.4.2. Tuning parameters for the model), we have to pass predefined values for parameters to the GridSearchCV function by defining a dictionary in which we mention a particular parameter along with the values it can take.



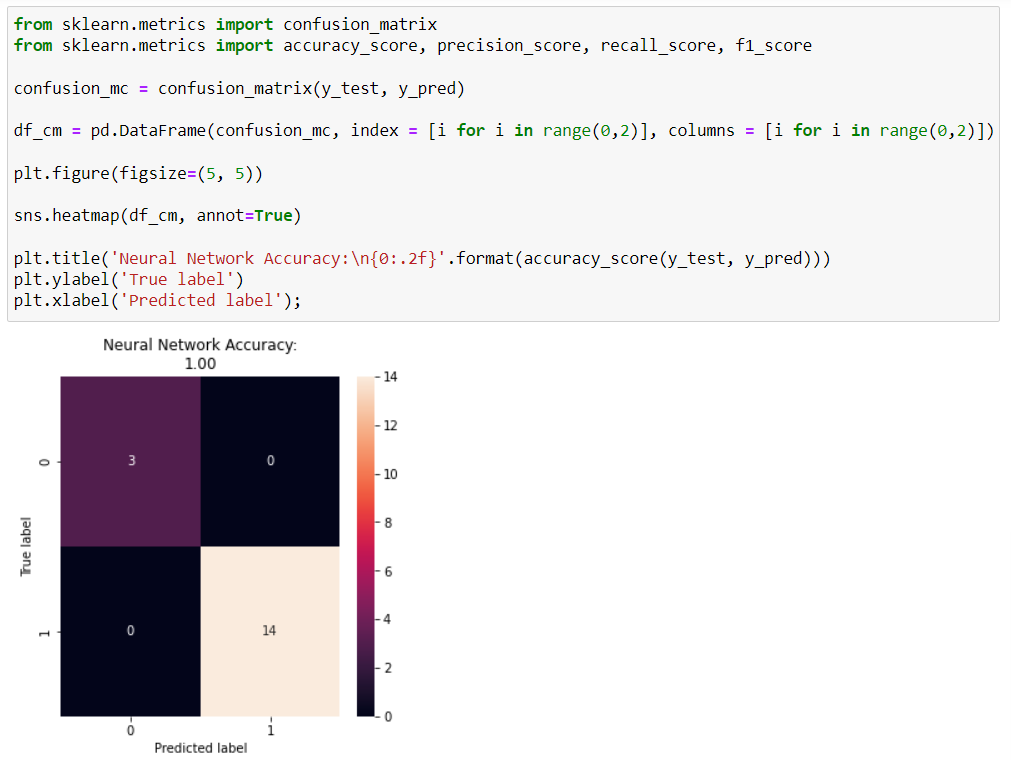
### A.2.2 The final results

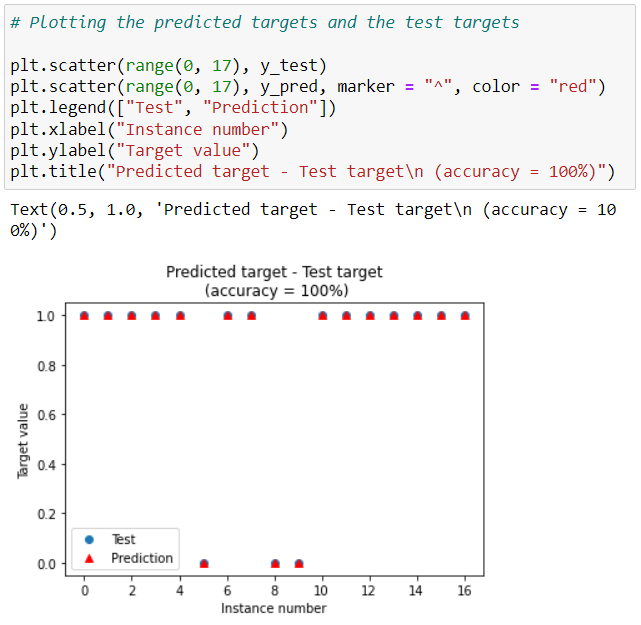


Finally, we notice that using GridSearchCV to find the best parameters for the classifier is not the optimal way to do it, doing it with for loops seems to be better, in our case at least.

This limitation is due to the fact that the search can only test the parameters that we fed into, and giving a large set of values in each parameter can produce a really high time-consuming search.

## A.3 Confusion matrix and results plotting code (Neural Network for Classification)



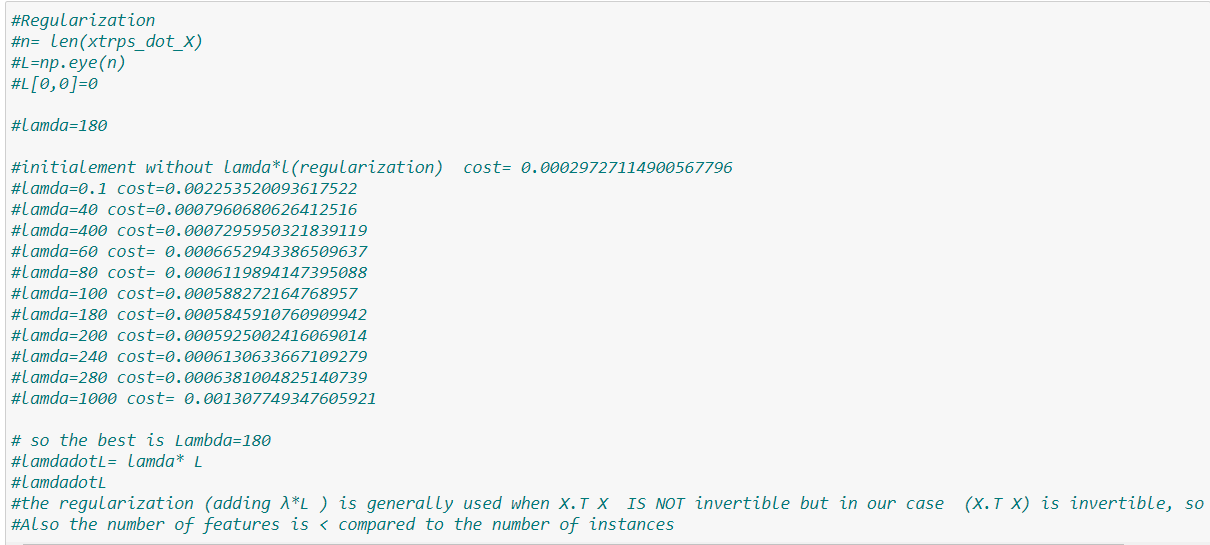


## B.1 Regularization in Normal Equation

In order to apply regularization in Normal equation we just need to add λ\*L in our formula like this:

reverse(X.T X + λ\*L)

Now we test many values of λ and evaluate the cost



After that, we replace the formula: **temp\_1=np.linalg.inv(xtrps\_dot\_X) # inverse(X.T X)**

**By: temp\_1=np.linalg.inv(xtrps\_dot\_X+ lamdadotL) # inverse(X.T X + λ\*L)**

And leave the rest of the code as it is.

**Remark:** we prefer put regularization as a comment is our code for many reasons:

1. The regularization in normal equation is to add lambda\*L in the formula X.T\*X in order to apply the reverse, however, in our case the formula inv(X.T\*X) without adding λ\*L is possible .
2. The number of features n is < to the number of instances

## C.1 Accuracy function

Here is the function we defined to get the accuracy of our model at predicting the test set after training it with the train set.

Une image contenant texte

Description générée automatiquement

## C.2 Accuracy test with predefined function

After finishing our train and test with the functions we defined, we use already existing functions to compare the results.

Une image contenant texte

Description générée automatiquement

# References

**Logistic Regression:**

<https://www.youtube.com/watch?v=nzNp05AyBM8&ab_channel=CodingLane>

**Neural Network for Classification - GridSerachCV:**

<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html>

**Pre-Modeling: Data Preprocessing and Feature Exploration in Python:**

<https://www.youtube.com/watch?v=V0u6bxQOUJ8>

**Machine Learning Tutorial Python - 10 Support Vector Machine (SVM):**

<https://www.youtube.com/watch?v=FB5EdxAGxQg>