Topic :

Analysis and Pretreatment of DataSet in Classification : Logistic Regression

Module :

Artificial Intelligence - Machine Learning

Professor :

Dr.Boutorh

Student:

- *El-Moubarek Fayçal*

2020/2021

Summary

Classification : Bitcoin Ransomware Transaction

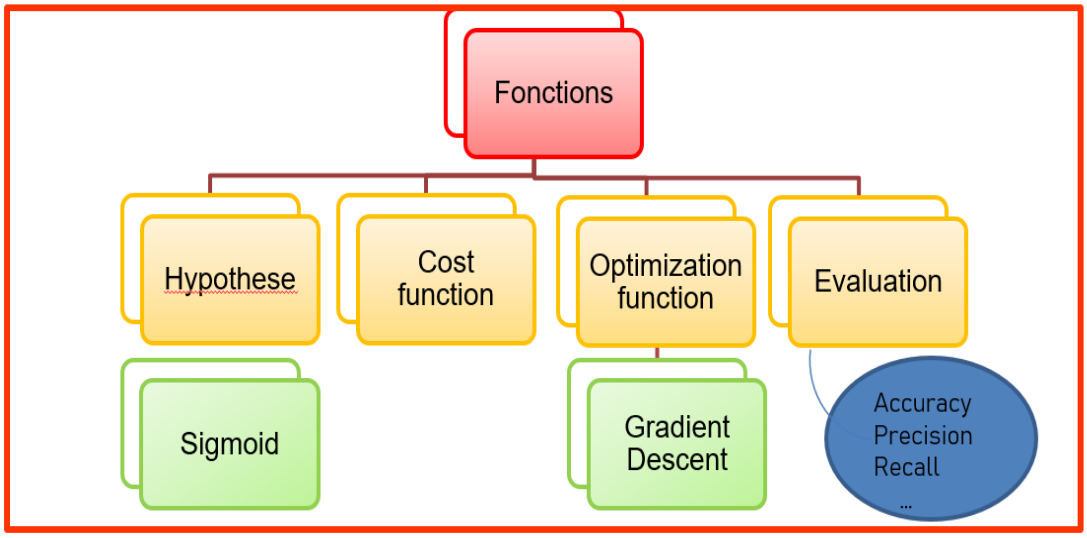
In this part we will browse our database, study our attributes and come up with approaches through Logistic regression

Introduction

Logistic regression:
the purpose of the model during this topic is to classify our different tansaction in two classes (a normal or malicious transaction) and in order to come out with a good prediction, this model will need 4 functions which will provide the necessary tools For this approach

Logistic Regression

Fonctions used



Preparation data :

In part A of our project, we made a first preparation of the data (removing outliers, noisy and duplicate data), and in order to avoid any anomaly that could arise during our approach from the data we found that it's judicious to review it and modify it again in order to removing useless attributes, reduce the features and convert our labels to integer '0' and '1'

Note: You will find the code of this part in the annex corner

Step 1: remove attribute « address »

The transaction addresses have no role in detecting the outputs (white or dark transaction)

Step 2: merge attributes

We have two features which represent the date of the transaction year and (1 = <day = <365), we can merge the two columns by converting the day into year and adding the result to year of the same row

Step 3: Convert labels to integer '0' or '1'

We want to detect malicious transactions so each time we find a dark transaction we put a 1 in the output: 1 represent a dark transactions / 0 represent a white transaction.

Step 4: Normalization of data

Transformation of dataset:

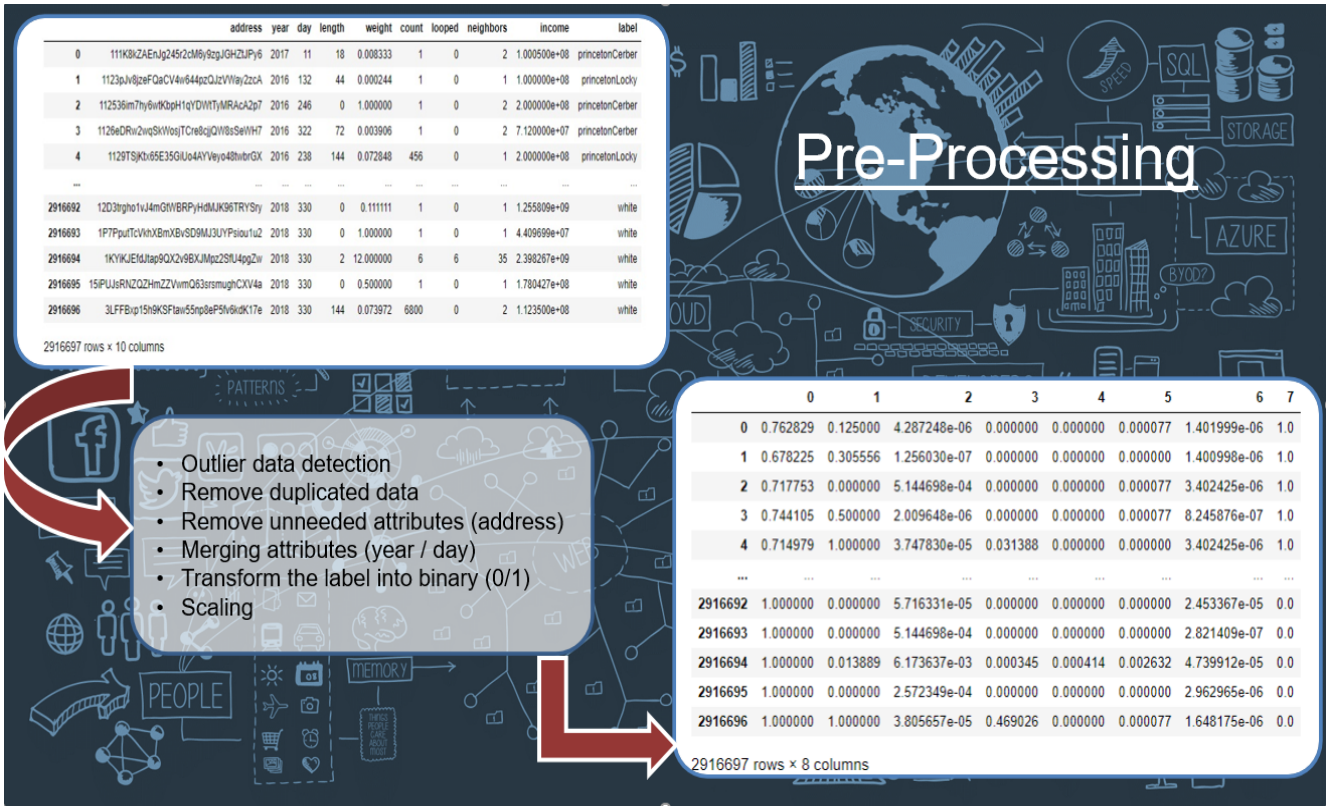


Figure 8 : Original Data-Set

The used version of the data:

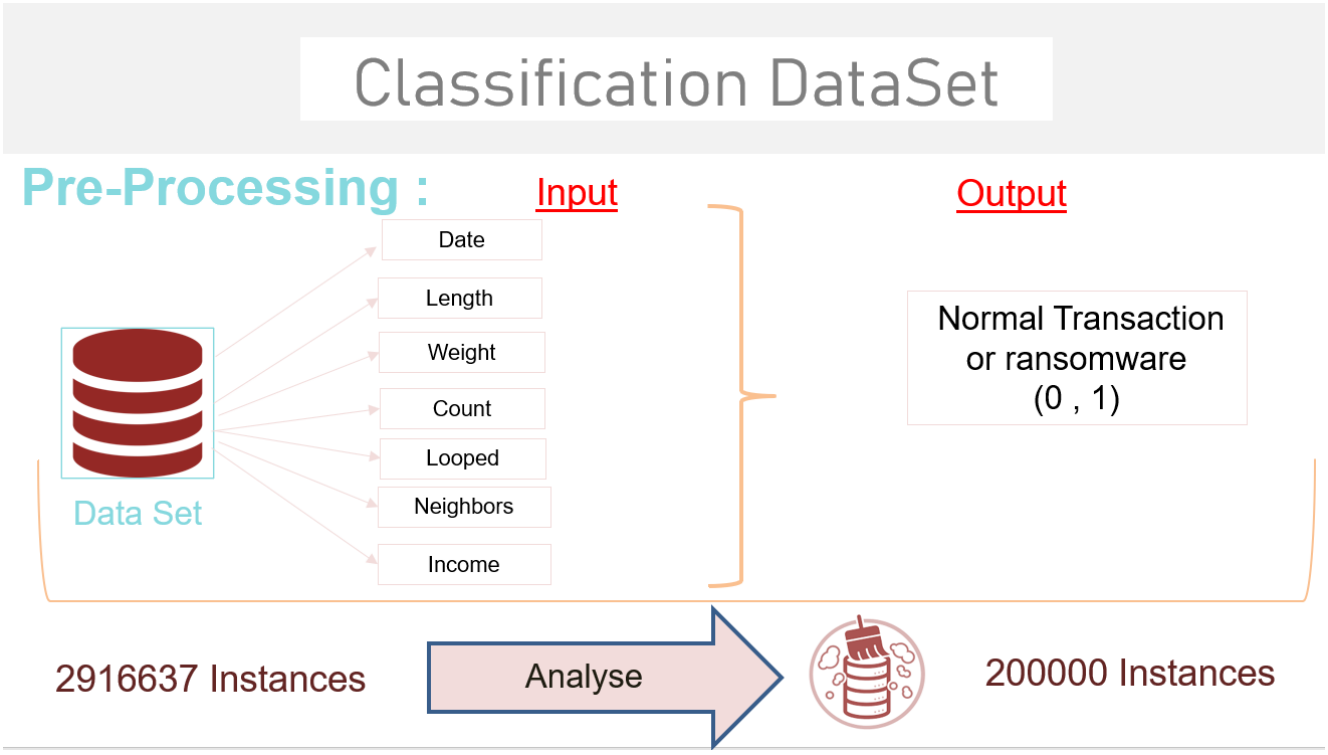
	0	1	2	3	4	5	6	7
0	0.762829	0.125000	4.287248e-06	0.000000	0.000000	0.000077	1.401999e-06	1.0
1	0.678225	0.305556	1.256030e-07	0.000000	0.000000	0.000000	1.400998e-06	1.0
2	0.717753	0.000000	5.144698e-04	0.000000	0.000000	0.000077	3.402425e-06	1.0
3	0.744105	0.500000	2.009648e-06	0.000000	0.000000	0.000077	8.245876e-07	1.0
4	0.714979	1.000000	3.747830e-05	0.031388	0.000000	0.000000	3.402425e-06	1.0
...	...	...	...	...	...	...	...	...
2916692	1.000000	0.000000	5.716331e-05	0.000000	0.000000	0.000000	2.453367e-05	0.0
2916693	1.000000	0.000000	5.144698e-04	0.000000	0.000000	0.000000	2.821409e-07	0.0
2916694	1.000000	0.013889	6.173637e-03	0.000345	0.000414	0.002632	4.739912e-05	0.0
2916695	1.000000	0.000000	2.572349e-04	0.000000	0.000000	0.000000	2.962965e-06	0.0
2916696	1.000000	1.000000	3.805657e-05	0.469026	0.000000	0.000077	1.648175e-06	0.0

2916697 rows × 8 columns

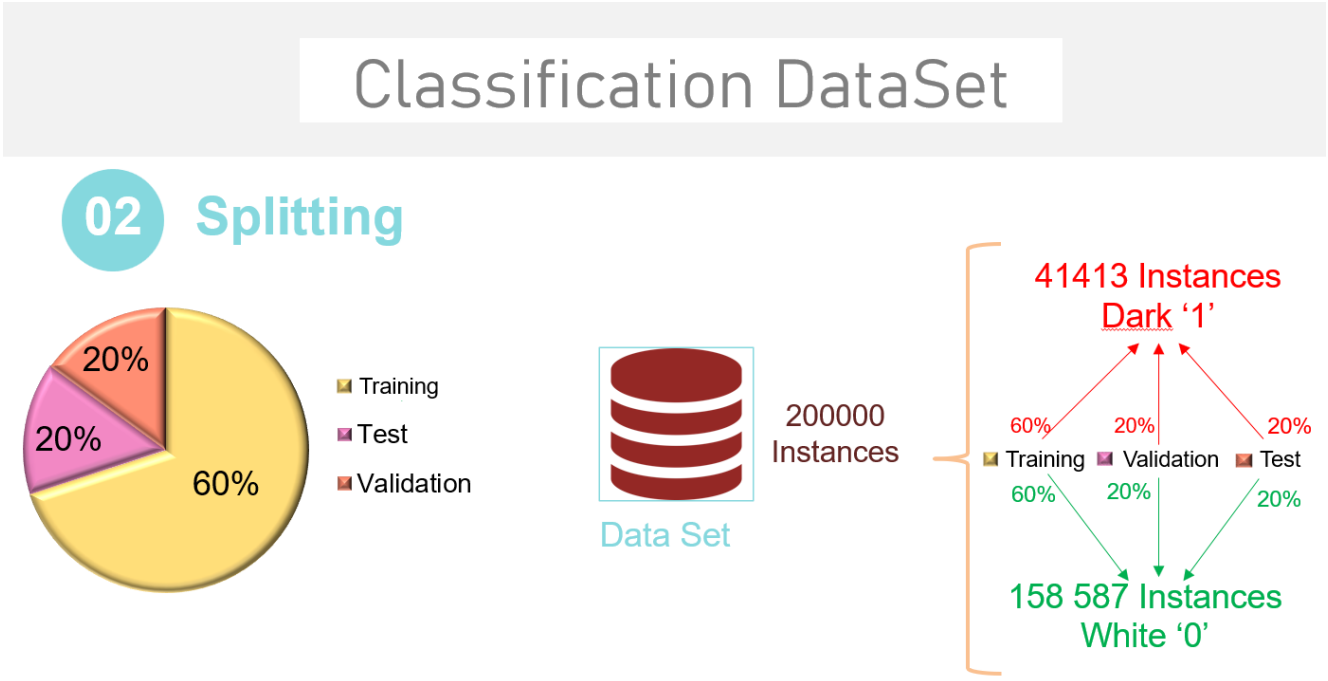
Figure 9 : Data-Set After Normalization

Splitting Data

Before starting the data splitting, we want to show you that number 'm' of instances is almost 3 million and we only took a '200 thousand' in order to avoid the problems of overfitting and underfitting and also take into account the performance of our machines which will calculate and execute our approaches.



60% for Training, 20% for Cross-validation, 20% test



we noticed in the data-set that all the transactions having an output '1' are aligned at the beginning and in order to distribute the data well, we shared the part with an output '1' for the 3 sections "training, cross-validation, test) more clearly: we have 41413 lines with an outputs '1' so: 60% of the lines with an outputs '1' for training, 20% for validation and 20% for test. and the rest is added according to the number of lines which correspond to the percentage of each part

- **The reason for this repartition of data with output 1 :**

In order to evaluate our approach in a correct way, it is very important to have the TP / FP and TN / FN in all parts so that we can compare the scores of the Cross validation and training part properly.

Logistic regression :

Before starting to show the different steps of the approach I would like to remind you that we are treating the case of binary classification of logistic regression, so the objective is to predict for each data the appropriate class [0,1]

In order to arrive at good predictions, we will need functions that will provide us with different tools (Best-Theta, Minimal-Cost, Accuracy ...) that will help us to classify the data and evaluate the different possible results.

I want to clarify that before starting to build my functions I added to my Inputs a column x0 containing '1', also I created a vector containing the starting thetas with a same size with number of columns (n) + column x0

size of the vector theta = n + 1

Hypothese (sigmoid function):

X : Matrix of Inputs of training data

Θ : Vector contain the weights of thetas

$$h_\theta(x) = \sigma(\theta^T x)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

(Z = X0*Θ0+ X1*Θ1+ X2*Θ2+ X3*Θ3+ X4*Θ4+ X5*Θ5+ X6*Θ6+ X7*Θ7) => Polynom °1

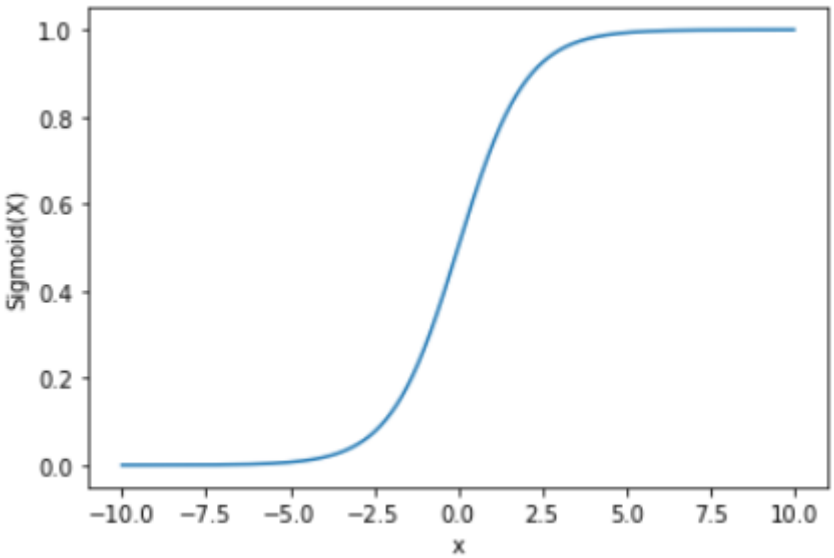


Figure 10 : Sigmoid Function

If h(x) > = 0.5 outputs will be classed in 1

If h(x) < 0.5 outputs will be classed in 0

Also the value of H(x) is used to calcul the costfunction and the gradient descent

Cost function.

CostFunction (With regularization):

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$
$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m} \sum_{j-1}^{n} 0_j^2$$

In binary classification the value of Y can be 0 or 1 so:
when Y = 1

$$(1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})$$

And the CostFunction = $-\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + \frac{\lambda}{2m} \sum_{j-1}^{n} 0_j^2$

When Y=0

$$y^{(i)} \log h_\theta(x^{(i)})$$

And the CostFunction = $-\frac{1}{m} \sum_{i=1}^{m} (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) + \frac{\lambda}{2m} \sum_{j-1}^{n} 0_j^2$

When I implement the cost function in my code

I separate the two parts of this formula, I run the data and I sum them at the end

you can find the code for each function in the annexe part

Lambda λ : it's a variable that controls and helps our approach to avoid overfitting and underfitting

Gradient Descent (with regularization):

In order to find the best thetas, the descent gradients each time generate new theta values through its formula.

The gradient descends in general needs the sigmoid function as well as the cost function to provide us with the minimum costs and the best thetas.

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$
$$\theta_j := \theta_j - \alpha \left[\left(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right] \qquad j \in \{1, 2 \ldots n\}$$

Alpha α : is the learning rate parameter set in a gradient descent to get the desired outcome

Implementation :

With First polynomial degree :

Try 1 :

Alpha α = 0.08, Lambda λ = 0.01, iteration : 50

```
Best thetas : [-0.3207344,  0.89080486, 0.80005749, 0.99972073,0.9980365
1,  0.99936585, 0.99989884, 0.99970288]
Optimal cost : 0.08508797394339432
```

Evaluation 1:

Accuracy on Cross Validation = 75.3925%

before setting up the test-data part we see that our accuracy is far from being accepted then we try to change our alpha and lambda parameters and if necessary we add the iteration number of the learning

Try 2 :

Alpha α = 0.08, Lambda λ = 0.05 , iteration : 50

```
Best thetas : [-0.3207344,  0.89080486, 0.80005749, 0.99972073,0.9980365
1,  0.99936585, 0.99989884, 0.99970288]
Optimal cost : 0.052438230548465435
```

Evaluation 2:

Accuracy on Cross Validation = 75.42%

I tried to modify the lambda and alpha in different way, the cost function is changing a little bit but the accuracy it's stable at around 75%.

I decided to increase the number of iterations to 150 :

Try 3:

Alpha α = 0.08, Lambda λ = 0.01, iteration : 150

best thetas = [-0.48499347 1.83218166 1.01885592 0.99983389 1.06599888 1.01049505

0.99992192 0.99932409]

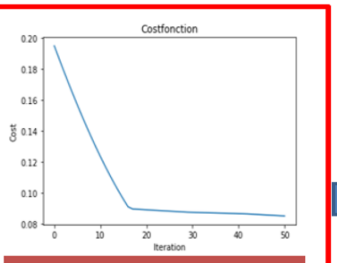
Optimal cost : 0.012438230548465435

Eavluation 3:

Accuracy = 92.5475% / Classification Error = 7.4525% / Recale = 73.69926694081889%

tests

TEST 1

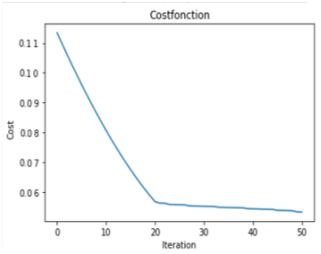


Degré polynomial = 1
Lambda=0.01
Alpha=0.08
Nbr Iterations=50

Cost minimum
0,08508

Accuracy in
Cross Validation
75,3925%

TEST 2



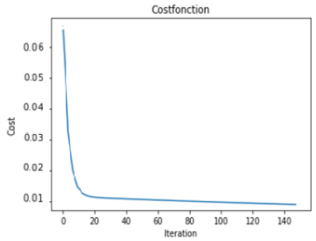
Degré polynomial = 1
Lambda=0.05
Alpha=0.08
Nbr Itérations=50

Cost minimum
0,05243

Accuracy in
Cross Validation
75,42%

tests

TEST 3



Degré polynomial = 1
Lambda=0.01
Alpha=0.08
Nbr Itérations=150

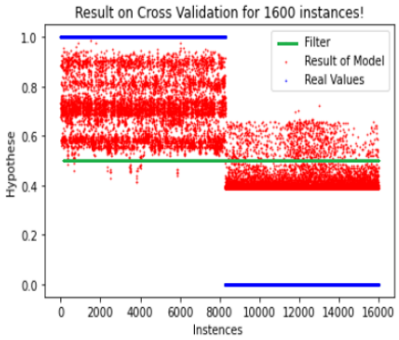
Cross validation

Cost minimum
0,012438

Accuracy in
Cross Validation
92,5475%

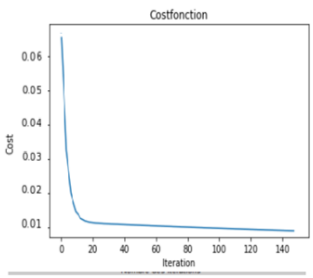
Classification
Error in
Cross Validation
7,4525%

Recall in
Cross Validation
73,69%



tests

TEST 3

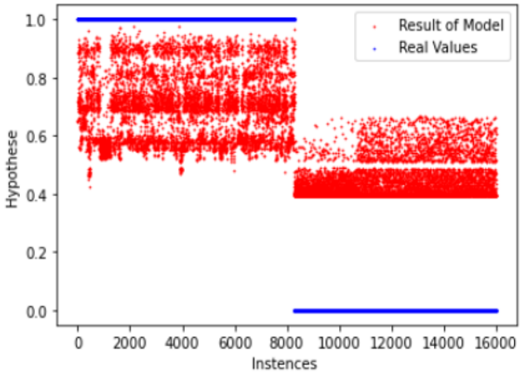


Degré polynomial = 1
Lambda=0.01
Alpha=0.08
Nbr Itérations=150

Test

Accuracy in
Test
80,251%

Classification
Error in
Test
20,849%



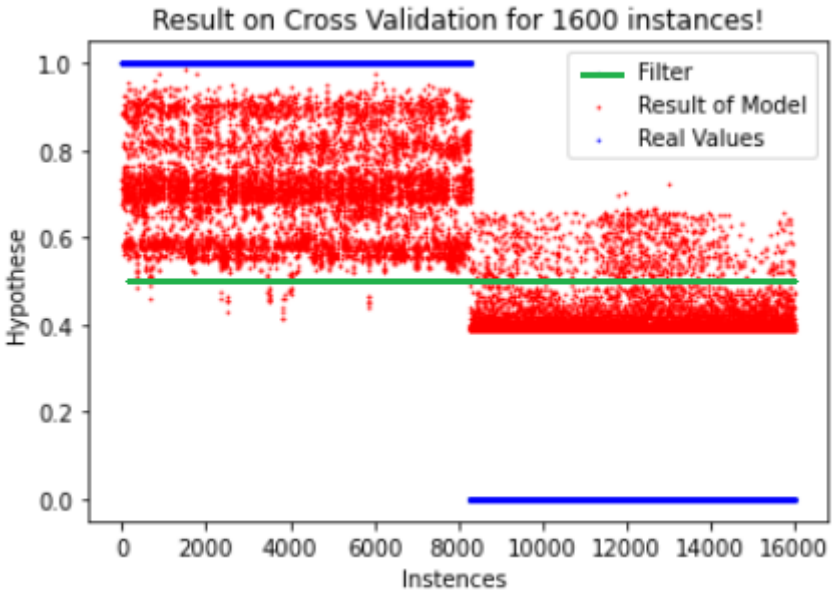


Figure 11 : Prediction on cross validation

Evaluation in test data :

Accuracy = 82.251%
Classification Error = 17.849%

Try 4:

in order to find if the evaluation will be better by adding the iterations, I tested with 1500 iterations : alpha α = 0.08, Lambda λ = 0.01, Accuracy = 93 %

for iterations x 10 of the 3rd test we obtained a slightly better accuracy, which leads to staying on an iteration number 150 with alpha = 0.08 and lambda 0.008

With 2nd polynomial degree :

I tried it With differents alpha and lambda and the probleme is the cost function diverge (fate of a local optimum)
an exampl of costfunction with alpha = 0.08 , lambda = 0.001 , iterations = 50

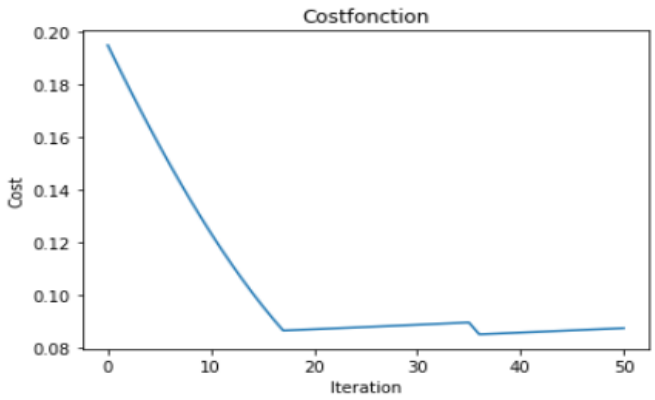


Figure 12 : Cost Function in 2nd polynomial degree