Project 3 – Fraud Detection

Soroush heidary - 96222031

Main Explanatino :

The biggest part of this exercise was the Data, the Data had missing values that had to be dealth with, varying ranges, huge ram allocations, and etc

What has been done on Data is as follow :

1 - the ‘dtype’s are changed so the data takes less space

2 - null values are dealt with in 2 fasions (only one is implemented)

3 - some features had been ‘semi-onehot vectored’ :)

4 - and normalized at last

The autoencoder is simple, the encoder and decoder are almost mirrored and it’s

Fit by only the Unfraudulent Data, So it learn how to reconstruct this type of data

And its patterns

After that we’ll predict all the test data to the autoencoder and record the error in which all the fraudulent features are reconstructed

As the autoencoder was built and trained to see the pattern in normal data, it will have an abnormal error when reconstructing the Fraud patterns

We’ll set a fixed threshold and divide the predicted outputs using the threshold as a deciding line to our classification

That’s the main method used to classify the Test data for the submission on Kaggle which had a not bad score (79 percent) though I tried some other techniques too

But for some reason they didn’t yield any better scores

They will be discussed later on though for the sake of just a peek, one other way is as follow :

We'll have 2 autoencoders :

One which is only trained on Fraudulent Data which learns to create usefull latent vectors of Fraudulent features that we'll use later on

Anotherone which is only trained on nonFraudulent Data which learns to create usefull latent vectors of nonFraudulent features that we'll use later on

Now that we have these 2, we'll use only the 'Encoder' part of the autoencoder

As a function to find really Dense and sophisticated demonstration of our fraud and nonfraud class of data

Then we'll have around 50 thousand latent vectors (half of fraud and half of nonfraud we picked only 25 thousand of non-frauds to have similar number of each class)

And we train a classifier (XGBoost) like a simple simple ANN as a classifier

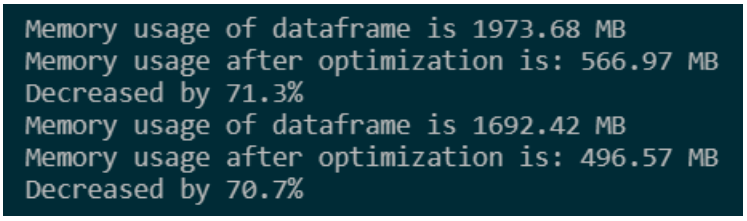
This too, had some interesting results but still not better than the first approach

The Work On Data :

First of all, there was a huuuuge memory allocation on each file, and we might want to deal with that first

Also we have to merge the identity and transaction files on each of test and train

The memory allocation before and after the reduction



How the reduction is done is simply by changing the columns dtypes to ones that require less space and there were many columns who could get converted to float16 and int16 as it was initially allocated float64 and int64

Lets see the range of each dtype :

```
Int   : -9223372036854775808 ~ 9223372036854775807
Int8  : -128 ~ 127
Int16 : -32768 ~ 32767
Int32 : -2147483648 ~ 2147483647
Int64 : -9223372036854775808 ~ 9223372036854775807
```

```
UInt   : 0 ~ 18446744073709551615
UInt8  : 0 ~ 255
UInt16 : 0 ~ 65535
UInt32 : 0 ~ 4294967295
UInt64 : 0 ~ 18446744073709551615
```

Here is an example list of which columns are converted to which types :

Same with the floats*

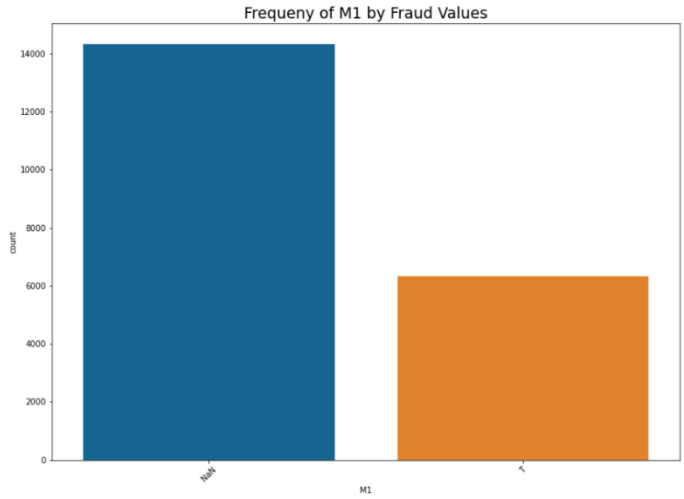
```
The Column 'C13_new' is converted from : int64        ---->       int8
The Column 'C14_new' is converted from : int64        ---->       int8
The Column 'D1_new' is converted from : int64        ---->       int8
The Column 'D2_new' is converted from : int64        ---->       int8
The Column 'D3_new' is converted from : int64        ---->       int8
The Column 'D4_new' is converted from : int64        ---->       int8
```

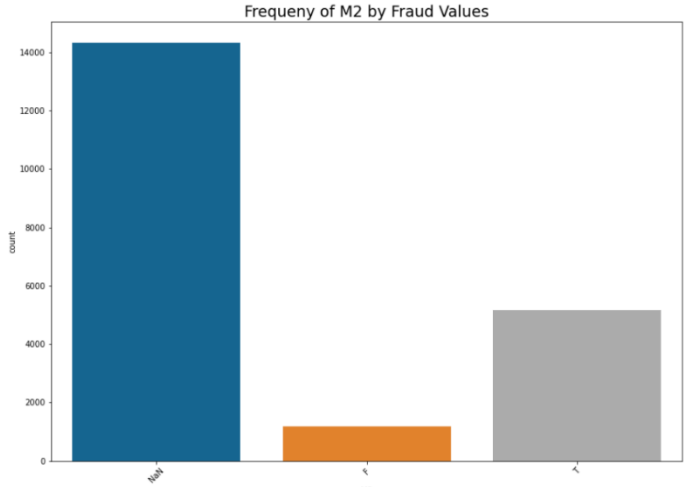
Now that we have reduced the size lets work on the null replacement (so called the imputation) :

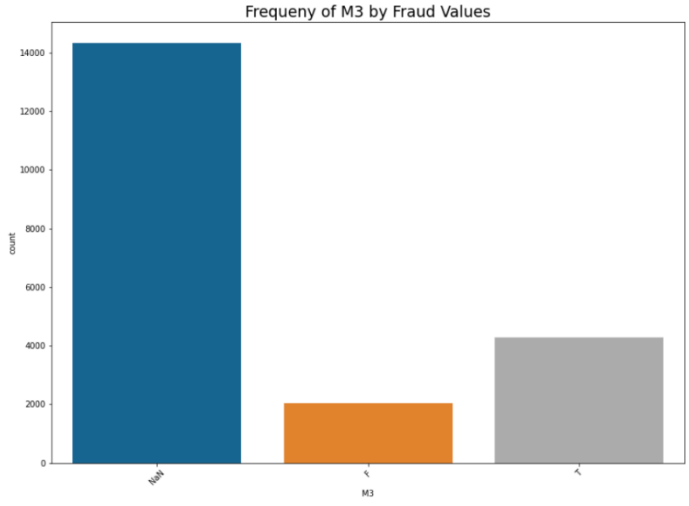
There were a couple of ways to deal with the Nan values, like removing the columns or rows, or filling the nulls with mean of mode of each column and …..

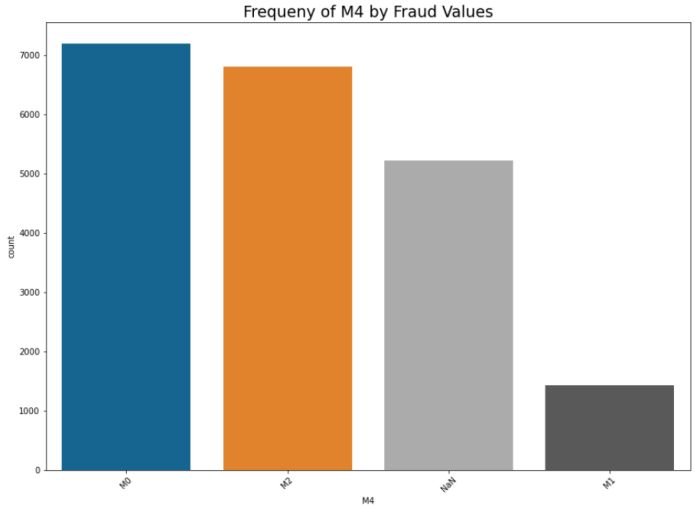
(worth to mention that I did try to fill the Nans of each column with the mode of that column and for that cause, we MUST divide the data to fraud and notFraud data to avoid the vanishing of fraudulent characteristics which is the dominated class of the two classes)

But, as I saw in a notebook from Kaggle which I have put its link at the end of the Report, it was shown that the values being null have a specific meaning, almost showing the lack of knowledge is, it self a feature that shows Suspicious characteristics, below are some charts (which was taken from the notebook that demonstrate what I claimed above)

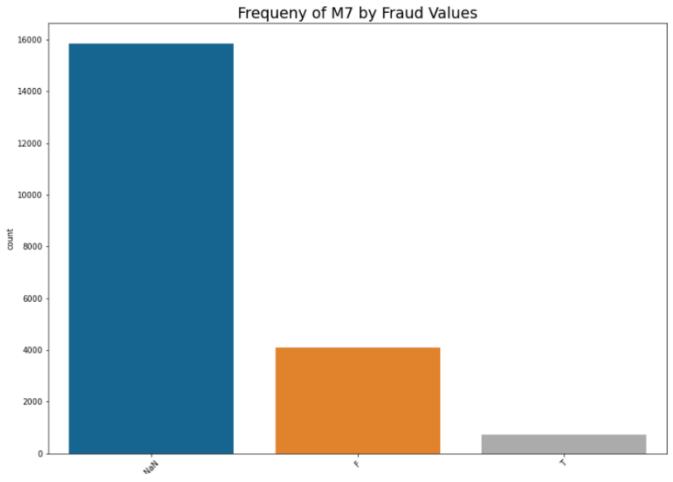


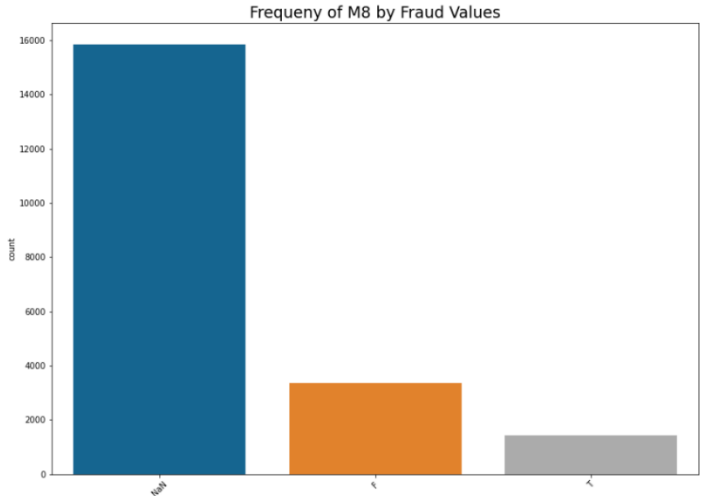


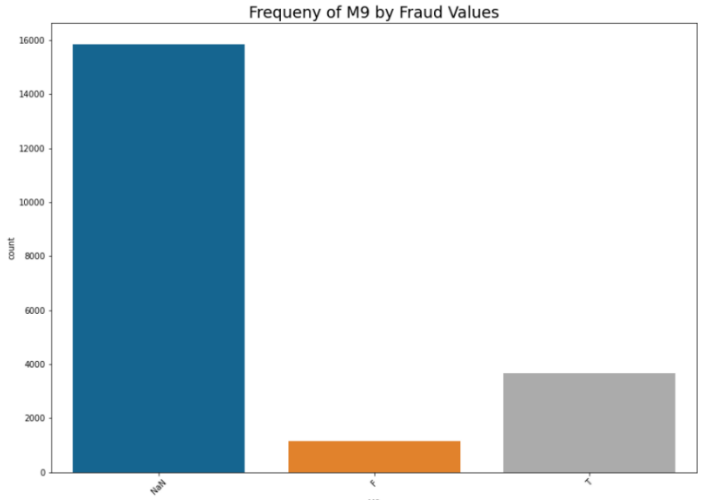


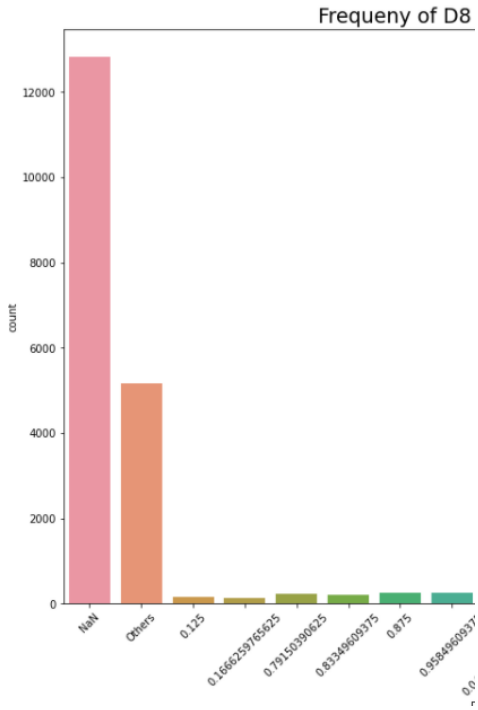


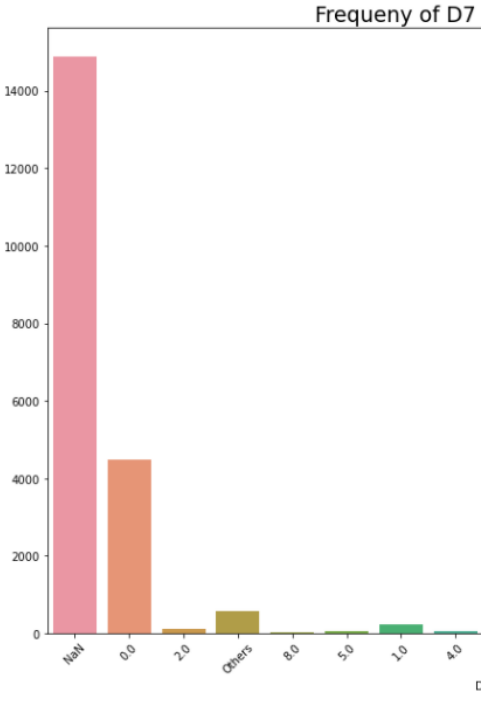

Frequeny of M6 by Fraud Values
count
T
NaN
F

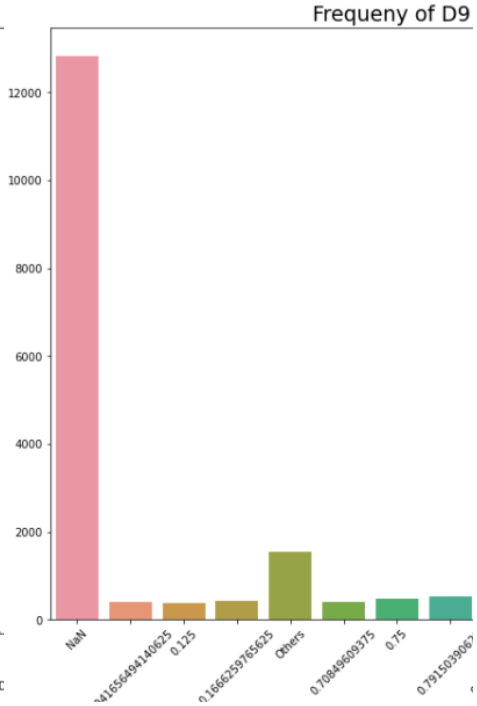

Frequeny of M7 by Fraud Values
count
NaN
F
T

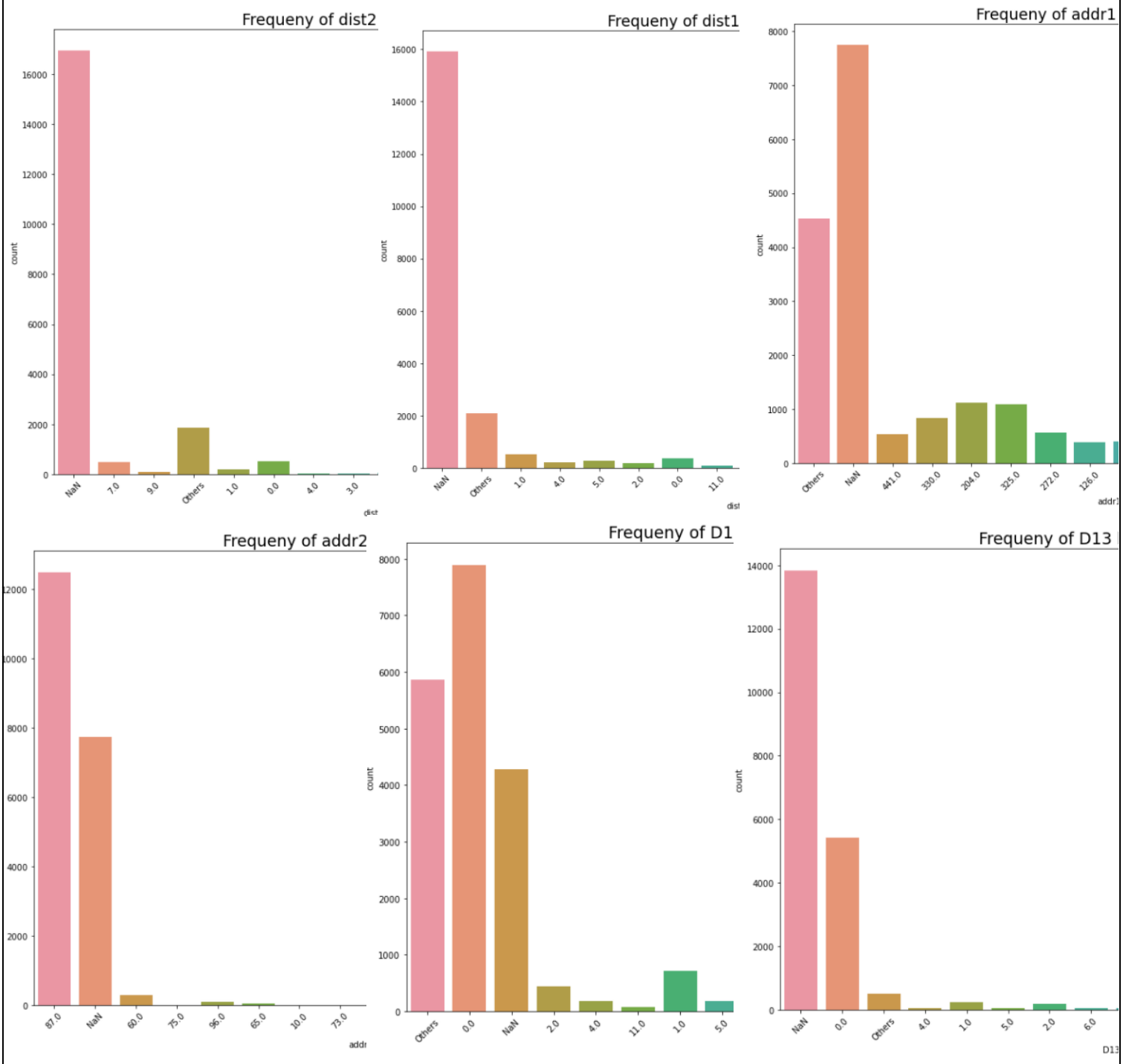

Frequeny of M8 by Fraud Values
count
NaN
F
T


Frequeny of M9 by Fraud Values
count
NaN
F
T


Frequeny of D8
count
NaN
Others
0.125
0.7915030625
0.8334960375
0.875


Frequeny of D7
NaN
0.0
2.0
Others
8.0
5.0
1.0
4.0


Frequeny of D9
NaN
0.125
Others
0.75



These are some of the features that showed a suspicious Nan amount on fraud data

Note that these plots are the amount of each value with respect to being fraudulent

As we saw in the plots, the Nans have a specific meaning so instead of filling them with mode or mean etc, we'll try to keep their identity while replacing them out of being Nan

For that cause we'll set an specific value like 9999 or something like that, and replace all the Nans with that value

Then we'll do some 'semi-one hot vectoring'

For example look at the plot for D13 (last plot) :

In this plot we'll see that a majority of fraud values lay in the 2 values of Nan and 0

For this example we can divide this feature into 2, one of them is related to those rows whom D13 is 0 or Nan and one those who have value of something other than 0 or D13, (I understand that doing this might be idiotic as we can simply assume one column, only to have value = 0 if that Row has D13 == 0 or Nan

And value = 1 if that Row has D13 != 0 or Nan

But I created 2 separate rows (which basically have the same meaning) but the idea was that by doing so, I'm emphasizing on the importance of this feature

As the majority of features (those labeled with 'V' were almost 70 percent of the features and didn't have richness of these features like 'D' or 'M' or 'C' or those mentioned above, but still I wanted to keep the 'V' labeled ones because of their minor effect and still we don't want the M or D to be denied comparing to 'V's

After this part the Nan amount have been finally reduced to zero and now we can start the training of the autoencoder with the non_fraudulent Data, here you can see the percentage of Nan values of some columns too :

```
Imputation of Fraudulent values in train :
amount of nan values are :  3003180
Nan amount after filling is:  0

Imputation of nonFraudulent values in train :
amount of nan values are :  103122168
Nan amount after filling is:  0

Imputation of values in test :
amount of nan values are :  83042804
Nan amount after filling is:  0
```

```
V258 0.7791343516103905
V259 0.7605310393876791
V260 0.7791343516103905
V261 0.7791343516103905
V262 0.7791343516103905
V263 0.7791343516103905
V264 0.7791343516103905
V265 0.7791343516103905
V266 0.7791343516103905
V267 0.7791343516103905
V268 0.7791343516103905
V269 0.7791343516103905
V270 0.7605310393876791
V271 0.7605310393876791
```

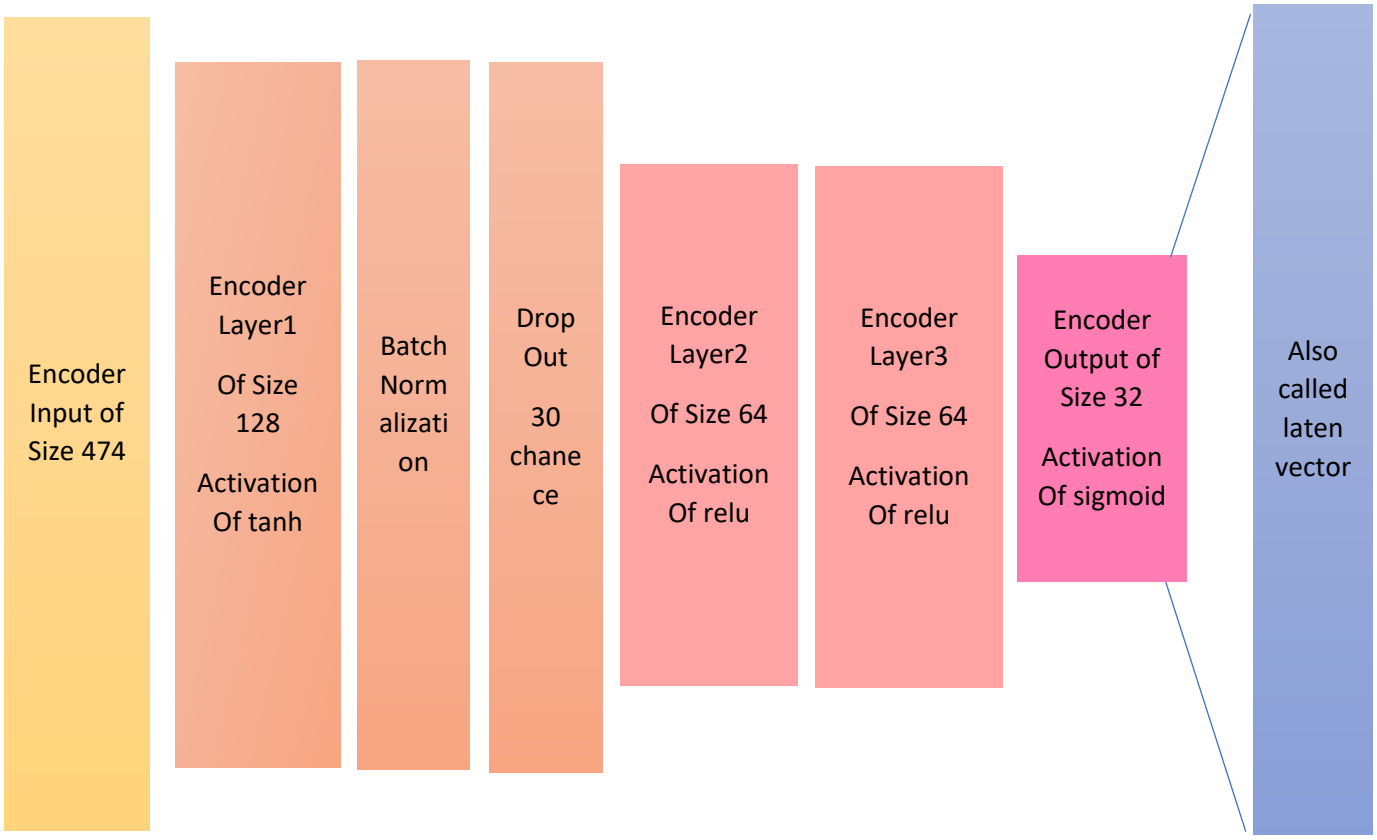
```
id_31 0.7624513157449114
id_32 0.8686185525112609
id_33 0.8758949436109323
id_34 0.8682477054898906
id_35 0.7612608798726589
id_36 0.7612608798726589
id_37 0.7612608798726589
id_38 0.7612608798726589
DeviceType 0.7615572188166763
DeviceInfo 0.7990551021099332
```

```
TransactionDT 0.0
TransactionAmt 0.0
card1 0.0
card2 0.015126833068039422
R_emaildomain 0.0
M1 0.0
V1 0.47293494090154775
V2 0.47293494090154775
V3 0.47293494090154775
V4 0.47293494090154775
V5 0.47293494090154775
V6 0.47293494090154775
V7 0.47293494090154775
```

Building the Auto Encoder:

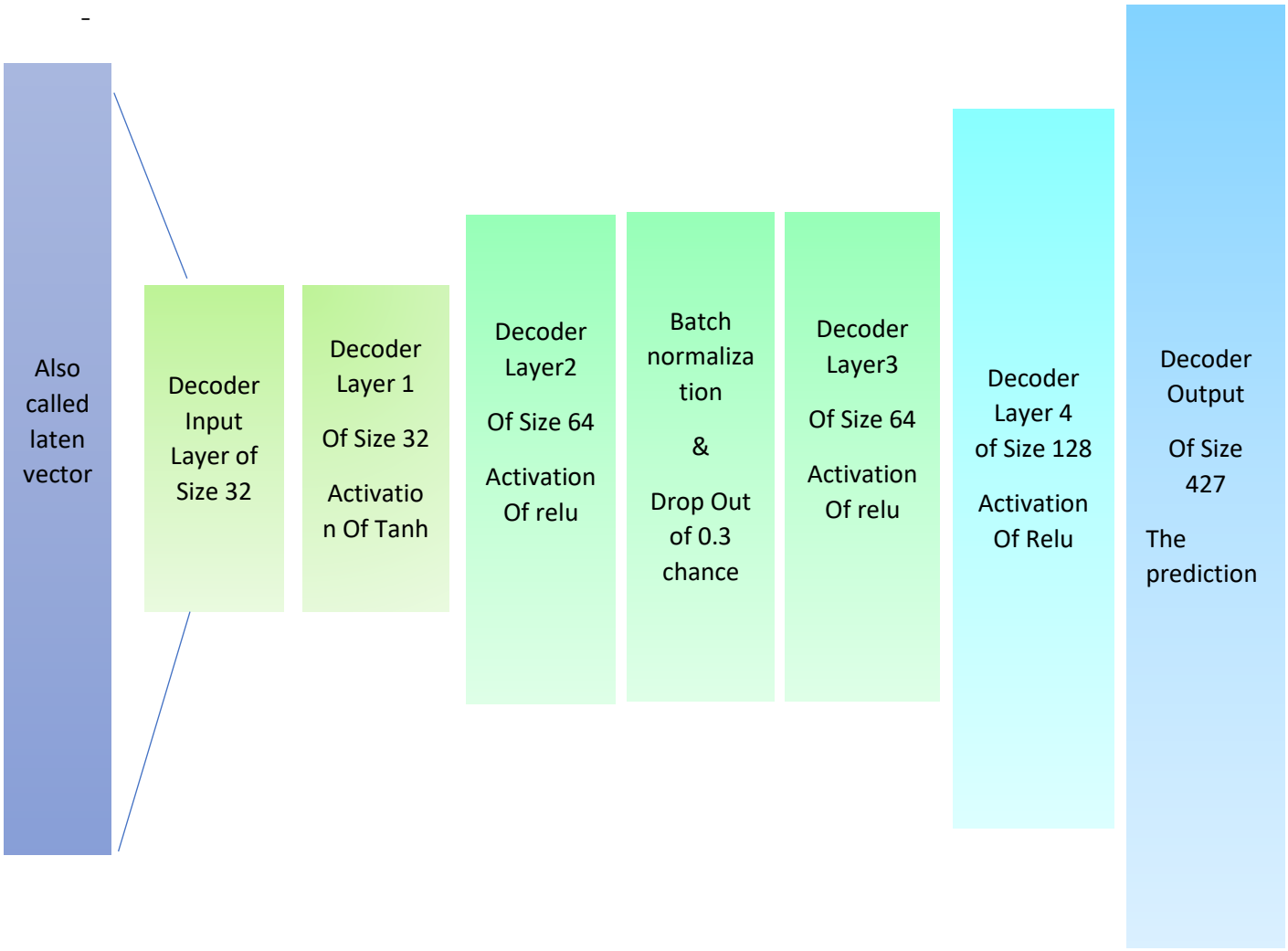
This autoencoder has almost mirrored decoder and encoder, though the structure is simple and doesn't need further explanations

Encoder :



The Decoder :

-



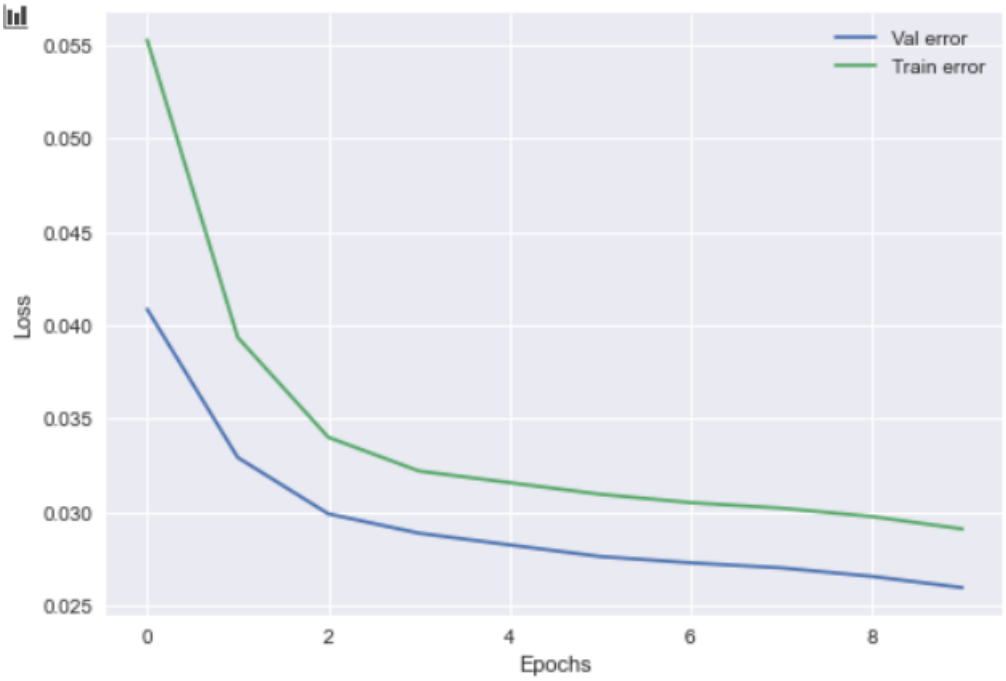
The procedure from now on is straightforward we'll just split the data into fraud and notfraud and we'll fit the notfraud ones

After that we'll try different thresholds for classifying the test data

The threshold used are : 0.032, 0.037, 0.057, 0.067, 0.087, 0.12

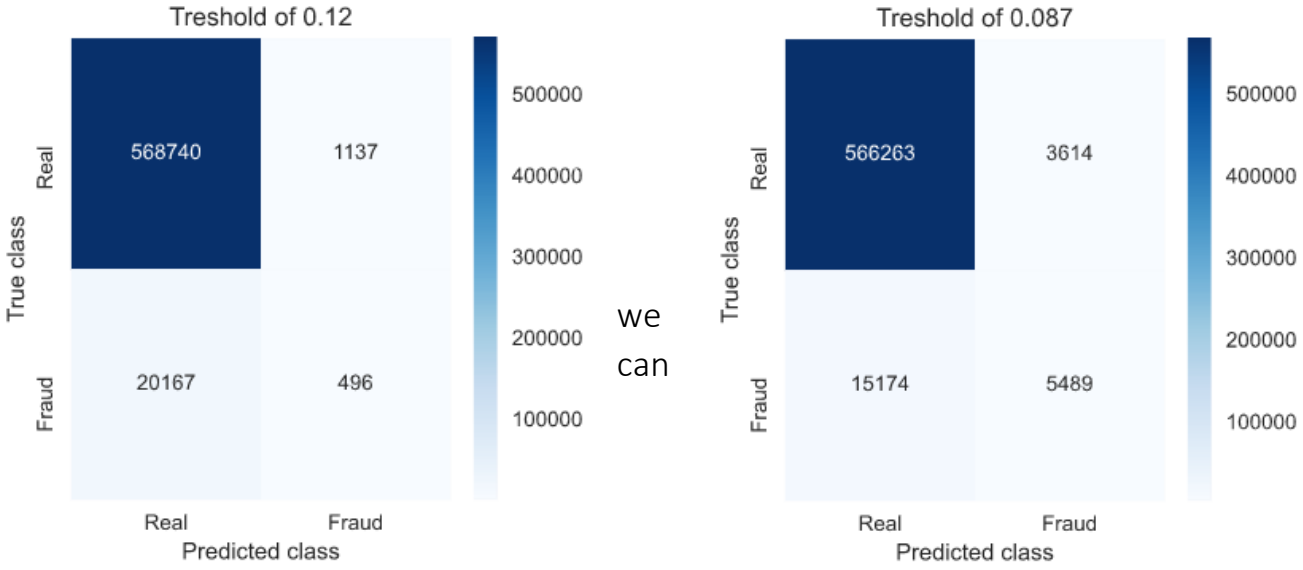
Why is used those are just a result of simple guessing and the fact that we have a val_loss of 0.026 at epoch 10

Here’s the loss(MAE) plot of each epoch : (of the best version of network)



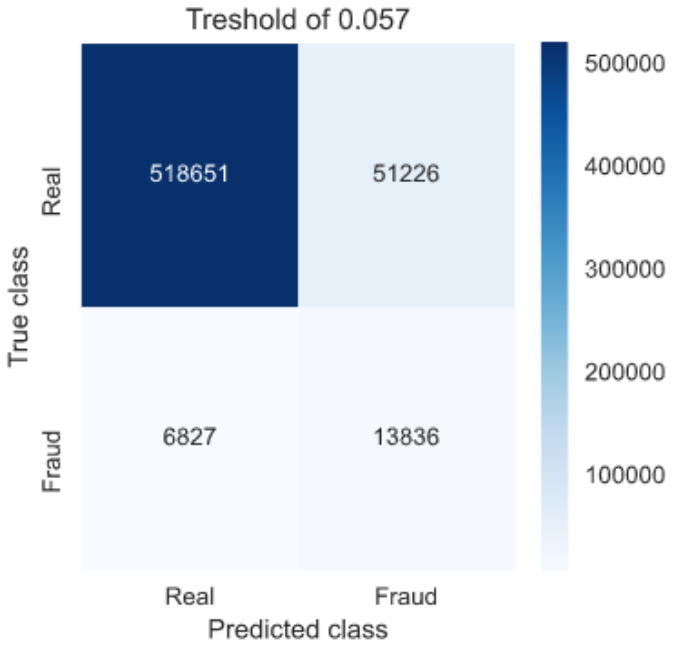
I could probably used more epoch for better result and score but I had limited time and hardware processing power so we’ll just leave it at that !

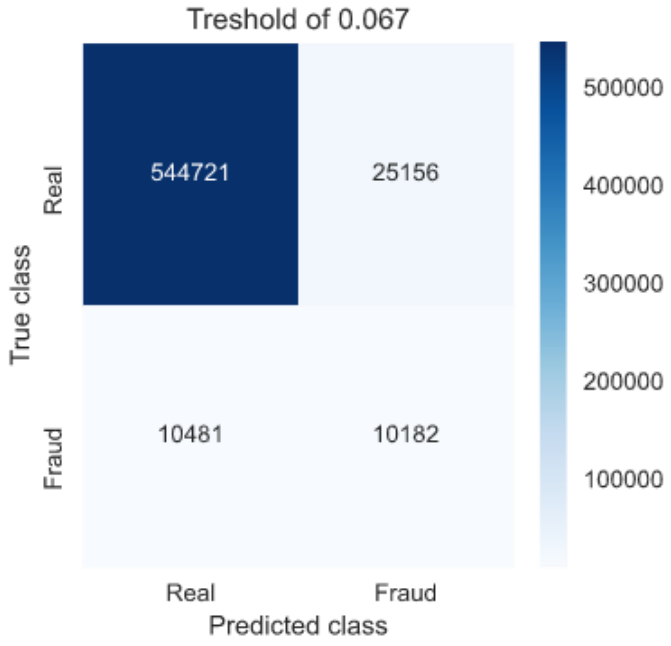
Now we’ll try the thresholds and create a confusion plot for sake of comparing the results of each threshold

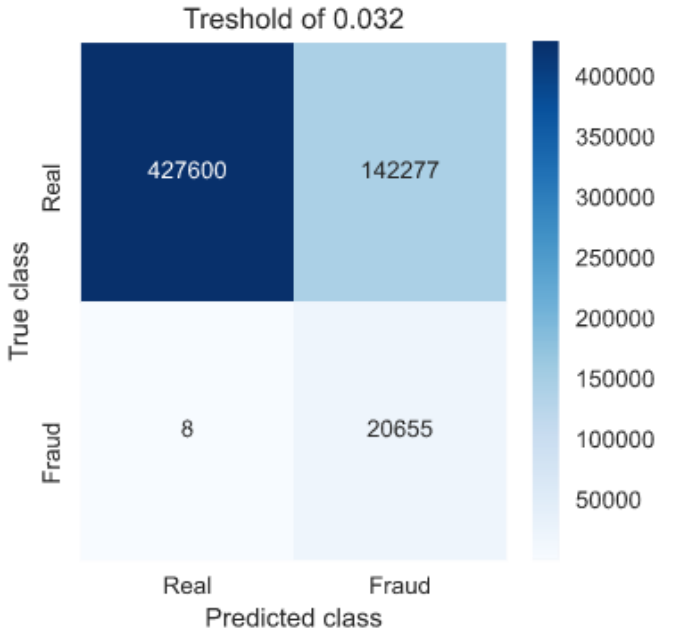


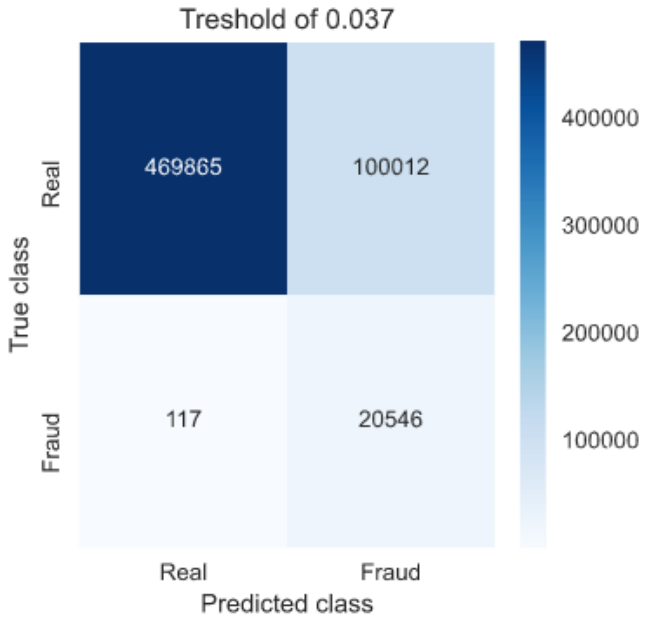
conclude that higher Tresholds (as expected) have really good prediction of Real data but a rather poor prediction for Fraud ones, in contrast lower tresholds like 0.032 have the exact inverted results

ofcourse we'll want something in between to predict each class on a good chance for that cause 0.037 and 0.042 were good decisions

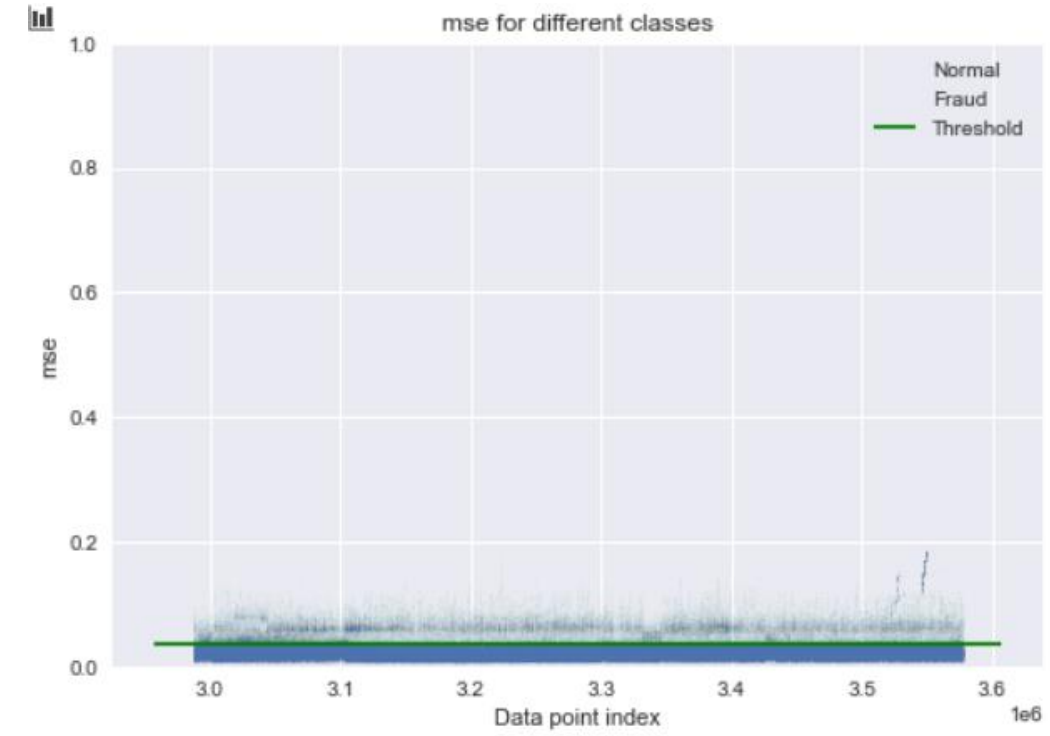


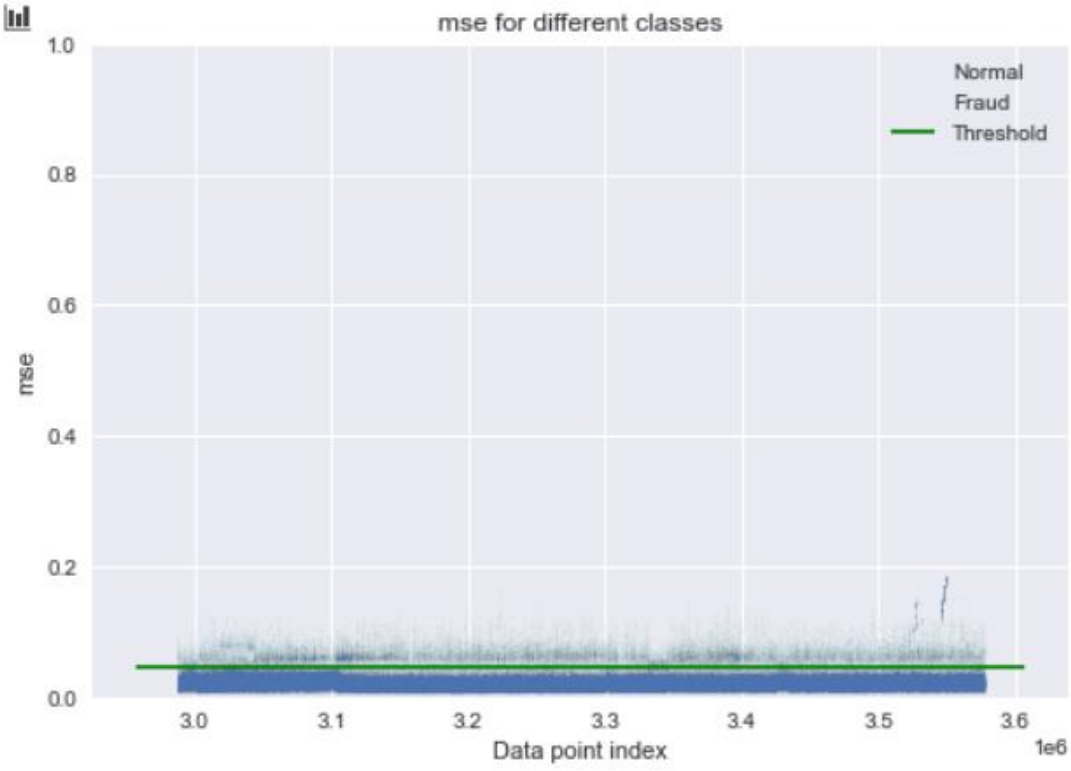






Also these two plots show the distribution of Data in respect to their error(first plot threshold = 0.037 and the second = 0.042) I've mistakably wrote mse but it's actually mae,



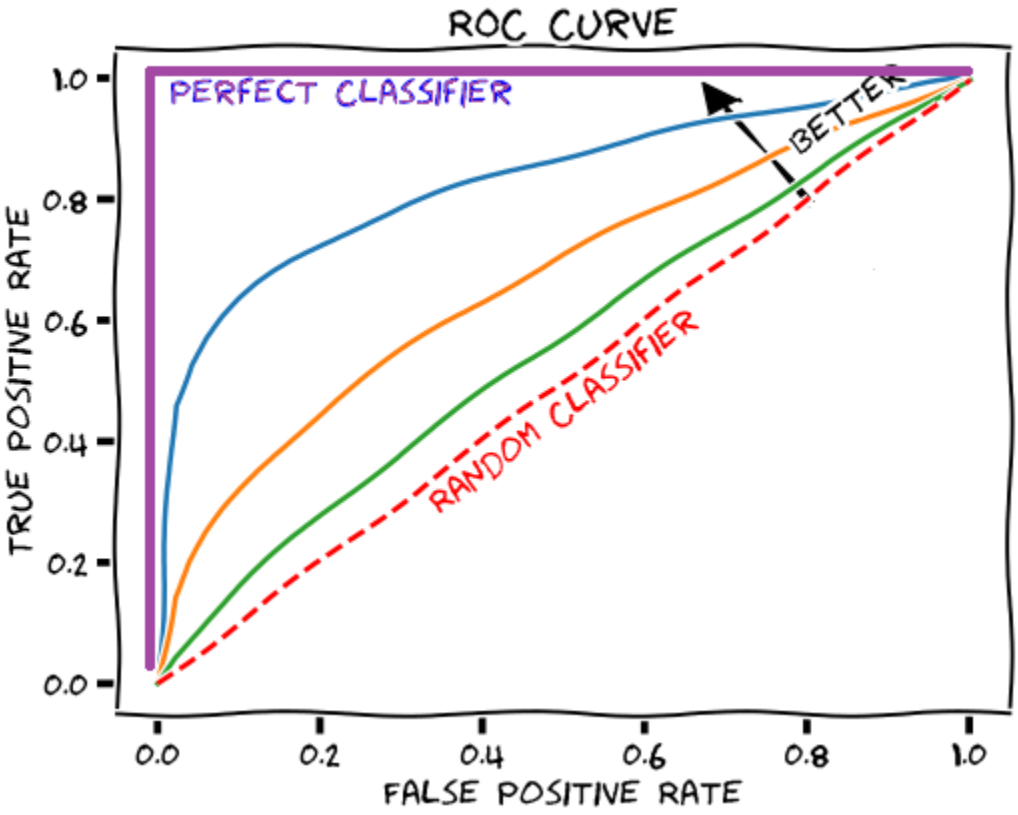


Just to not leaving The ROC Curve unmentioned here's the ROC Curve of the data

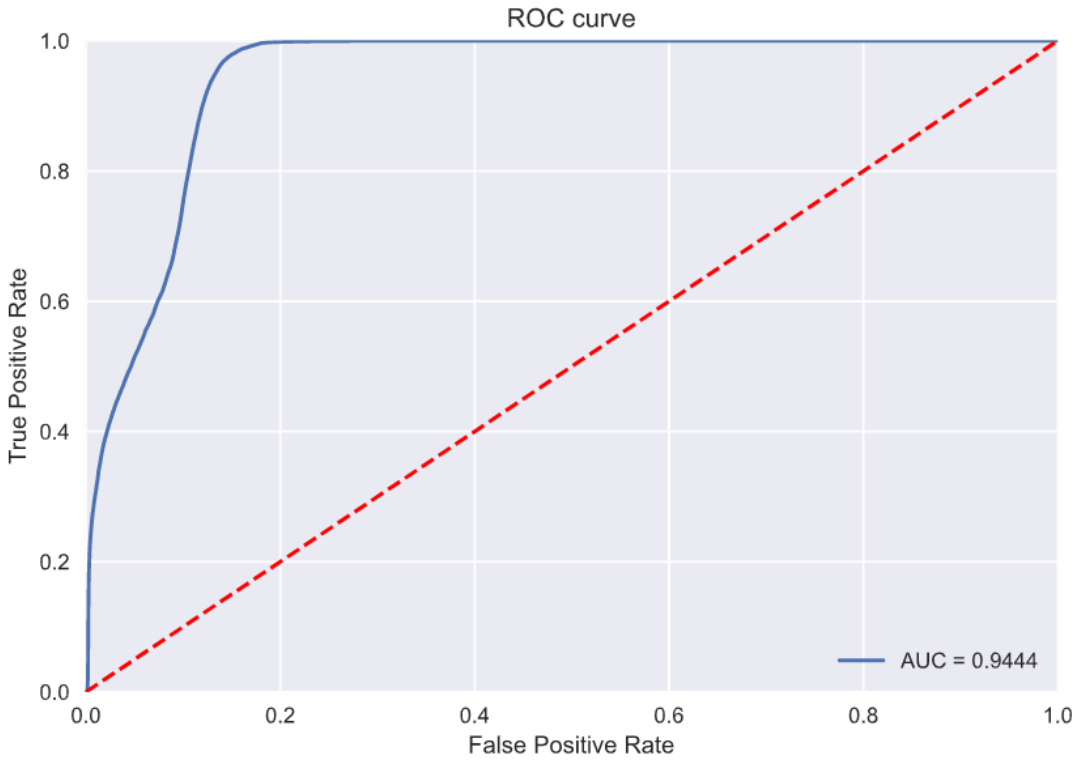
The ROC curve is an illustration of diagnostic ability of a binary classifier as it's discrimination threshold is varied (True positive rate with respect to different tresholds)

Basically, the more "humpy" it is, the better

There picture below is a good summary of what we said about a ROC Curve :



our ROC curve :



Other Techniques :

I said in the main explanation part that I tried other techniques like using an XGBoost classifier too but unfortunately my pc crashed during its training time and the code for this part was unsaved and I didn't have enough mentality to write it again :D so I'll just explain it

XGBoost is a recent kind of neural network that has become famous for it's good result in the recent times, What it does in detail is not the main point here

We'll have 2 autoEncoders one trained on frauds and one train on non faruds (we'll have to reduce the non frauds to match the dominated class we'll have around 20K of it's data picked randomly

Now we'll fit the Encoder part of each autoencoder with the respecting data

And use the output (A Dense and sophisticated version of the input) used as input of this XGboost network

The result didn't become much better than the Autoencoder (even a little bit more off) merely a good experience

I also tried to use a GAN for the project but for some reason I couldn't find the right hyperparameters for the network and thus couldn't get anything better than a random decider out of it !