**Logistic regression**

First, we implemented the model as his simple version, we divide the data to three parts , train, test, validation,

Chart, pie chart

Description automatically generated

We too put attention to the fact of the imbalance data of the train set,

Chart, pie chart

Description automatically generated

Then, we run the model at his simple version, 80 epochs, with learning rate of 0.01, we also divide the data train to batch of 100 samples for any iteration,

Text

Description automatically generated with low confidence

Then for every iteration on all the data we keep the loss value for the train, test and validation in order to recognize over fitting if exist and find the optimal early stopping from test lost function,

Graphical user interface, application

Description automatically generated with medium confidence

At the first and simple running we got this result,

Shape

Description automatically generated

As we can see the loss function decrease very fast, and we limit to 0 for only 80 epoch, we can show from the graph that we not tackle with overfitting.

And the classification report that we got,

Chart, treemap chart

Description automatically generated

As we can see the recall for 1 class his equal to random, so, we see that have very high tendency to the model to say not bulling (0 class) because almost all the data, complicated from not bulling sentence, so the accuracy that we got was **0.8345333333333333**.

we can see it too from the result of the confusion matrix,

Chart, treemap chart

Description automatically generated

First we want to check the option that the problem dependent on the number of epochs,

So we run it 150 epochs, and we got this results,

Chart, treemap chart

Description automatically generated

As we can see the recall stay very low, and the model didn't improve as we can thought, so we try to using methods of imbalance data,

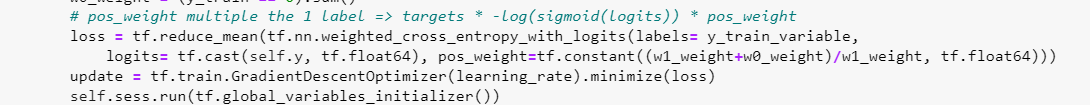
We try too to change the size of batch under assumption that the number of samples with 1 label very low in every batch, so we put 50 samples in every batch and we got,

Chart, treemap chart

Description automatically generated

The recall decline to change, the model didn't succeed to understand 1 label.

First we added weights to the loss function,



We define weights as below,



And we run it with 80 epoch,

Shape

Description automatically generated

First, we can see that the loss function decrease like before, we can see too that we not deal with overfitting, but now in the classification report,

Chart, treemap chart

Description automatically generated

The recall of 1 class highly increase, and the precision decrease (we tackle with the trade-off between the recall and precision) because that from now the model give more weight for 1 class, the accuracy still stay 0.8285333333333333,

We can see it too from confusion matrix,

Chart, treemap chart

Description automatically generated

If we compare the result that we got with the previous model we got this,

In the previous model from 3998 samples with label 1 we got only 2017/3998 = 0.52 of the data with label 1, in this model we got 3123/3998 = 0.78 of samples label as 1.

We try another method for imbalance data oversampling, so we use sickit-learn library for that,

Text

Description automatically generated

In the sampling\_strategy we brought the model that the relative between the labels will be 0.9, so the data look like this,

Chart, pie chart

Description automatically generated

we run the model again with 80 epochs and we got this results,

A screenshot of a cell phone

Description automatically generated with medium confidence

The recall increase very high, but the decrease too much, the accuracy decrease very much,

0.6973333333333334,we can see it too from the confusion matrix,

Chart, treemap chart

Description automatically generated

As we can see we have 4251 samples that the model label them as 1, so the incorrect increase very high.

Then we try to improve the best model that we got until here by using dynamic learning rate, so for this we create tensor,

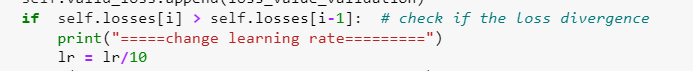


And we insert him to the gradient descent optimizer,

we too keep the loss that we got every iteration,



And every iteration on all the data we checked if the model loss decrease or increase, and we correct the learning rate,



We run it, and we got,

Chart, treemap chart

Description automatically generated

As we can see in compare to simple weight model, we can't see impact change in the performance of the model, so we can understand that the loss converge very good without the changeable of the learning rate.

Finally, we concat the train set and the validation set,

Text

Description automatically generated with medium confidence

And we check the model on the test set, and we got this result,

Chart, treemap chart

Description automatically generated

As we can see, the result of recall of 1 improved, as we increase the weights in loss function,

We can see it too from the confusion matrix,

Chart, treemap chart

Description automatically generated

**MLP – multi layer perceptron**

with learning\_rate =0.01, batch\_size=200, hidden\_layer\_sizes= (1000,500,100) and Adam optimizer we got:

precision recall f1-score support

0.0 0.74 1.00 0.85 11002

1.0 0.93 0.05 0.10 3998

accuracy 0.75 15000

macro avg 0.84 0.53 0.48 15000

weighted avg 0.79 0.75 0.65 15000

Chart, treemap chart

Description automatically generated

Chart, line chart

Description automatically generated

We notice that we have issue with the learning rate so use adaptive learning instead of constant meaning we reduce the learning rate when the loss increasing. Also we use more simple NN with (500,100) hidden layers

And we got:

precision recall f1-score support

0.0 0.88 0.94 0.91 11002

1.0 0.80 0.65 0.72 3998

accuracy 0.86 15000

macro avg 0.84 0.80 0.82 15000

weighted avg 0.86 0.86 0.86 15000

Chart, treemap chart

Description automatically generated

After these results we used oversampling method to getting better recall score:

precision recall f1-score support

0.0 0.91 0.89 0.90 11002

1.0 0.72 0.77 0.74 3998

accuracy 0.86 15000

macro avg 0.82 0.83 0.82 15000

weighted avg 0.86 0.86 0.86 15000

Chart, treemap chart

Description automatically generated

Chart, histogram

Description automatically generated

And with more complex network of hidden layers (1000,500,100,50) we got:

precision recall f1-score support

0.0 0.91 0.91 0.91 11002

1.0 0.75 0.76 0.76 3998

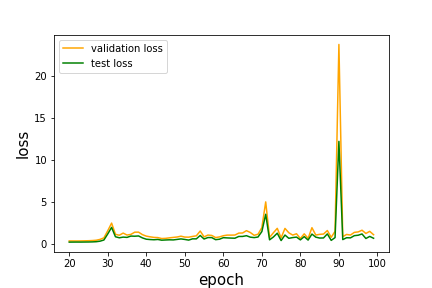
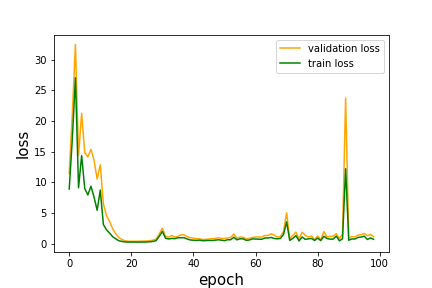
accuracy 0.87 15000

macro avg 0.83 0.83 0.83 15000

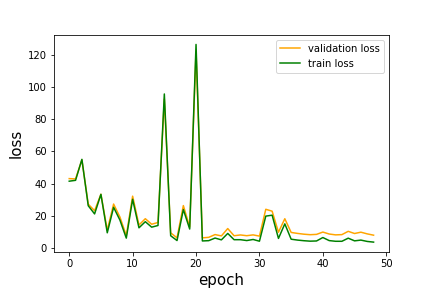
weighted avg 0.87 0.87 0.87 15000

**MLP – multi layer perceptron**

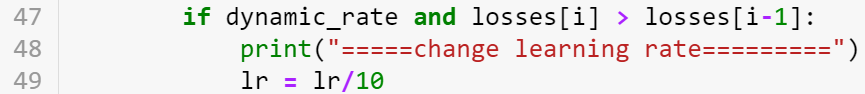
On our first try, we trained a simple MLP with hidden layers of sizes (100,60,20), learning rate of 0.01, and batch size of 200. We trained the network for 100 epochs. The graphs below show the change of the loss value of the train and validation sets over time.

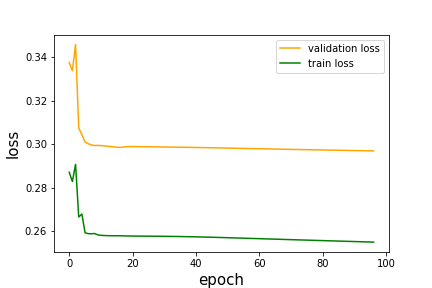


As can be seen, there is “noise” in the loss graph, and it as not going down after around 20 epochs. We suspect this is duo to too high learning rate. We try to re-train the network with learning rate of 0.001 (for 50 epochs this time).



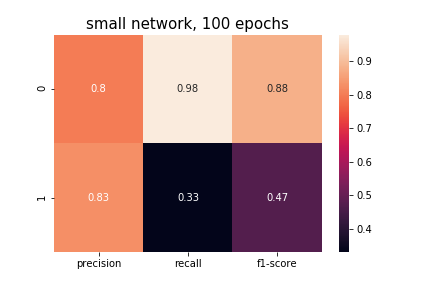
Lower learning rate did not fix the problem. The next thing we tried is implementing a dynamic learning rate – every time the loss is higher than the previous iteration, we divide the rate by 10.



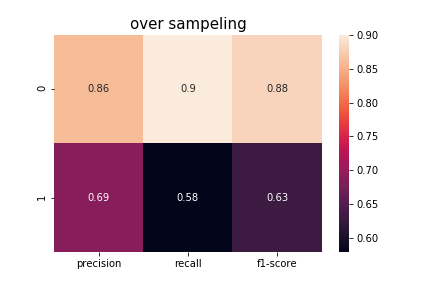


The dynamic learning rate did fix the problem – the loss is (almost) consistently going down.

Is immediately calls for more epochs of training. We also got a bigger network: (1000-600-200). The results are:



In both cases, the recall for 1 (bulling) is very low. This can be because the data imbalanced – 73.2% is 0, while only 26.9% is 1. We tried to overcome this by over sampling and replicating 1 samples to get a ratio of #1/#0 = 0.9. this method yielded the following results:



The recall for 1 did increase, at the cost of a big decrease in the precision.

Next, we tried dealing with the issue of the imbalance data by adding weights to the loss function. For each class (0 or 1 in our case), we gave a weight of 1 over its part in the data (that is, if for example 0 is 0.25 of the data, it weight will be 1/0.25 = 4).

This method was better for the recall of 1, but lowered the recall for 0 and the precision for 1:

