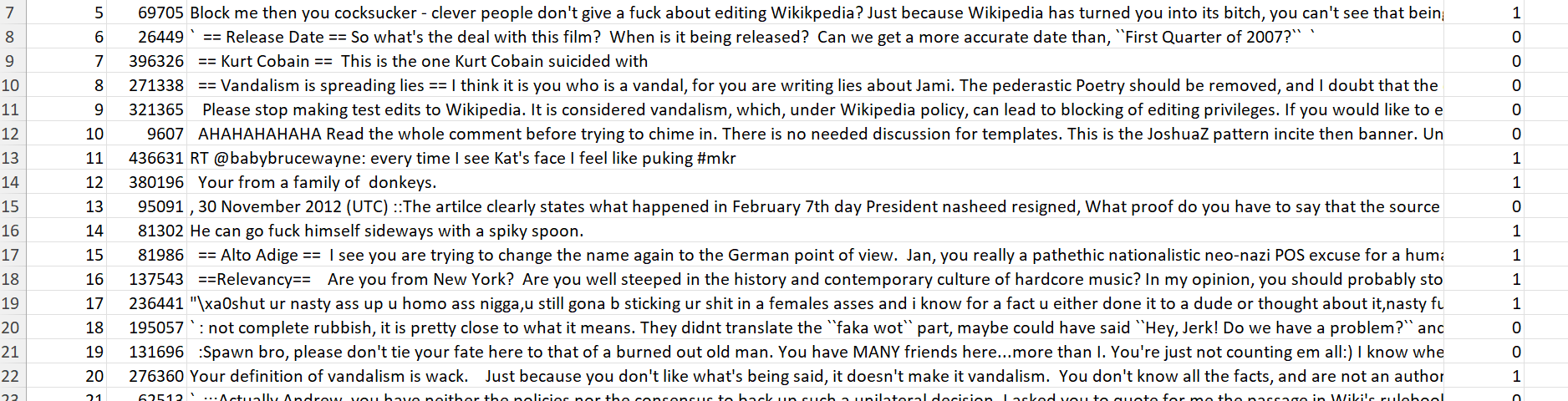
**Pre-prosses**

Out data consist of 100,000 samples of text and label (1 for bullying 0 for not).



The first step we did is removing any “weird” characters:

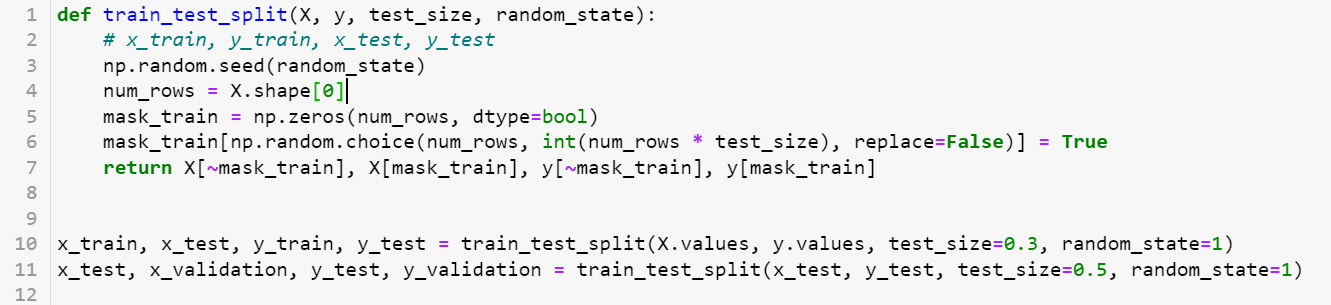
And then, in preparation for using “bag of words” features, we lemmatized each word.



Split to X and Y:



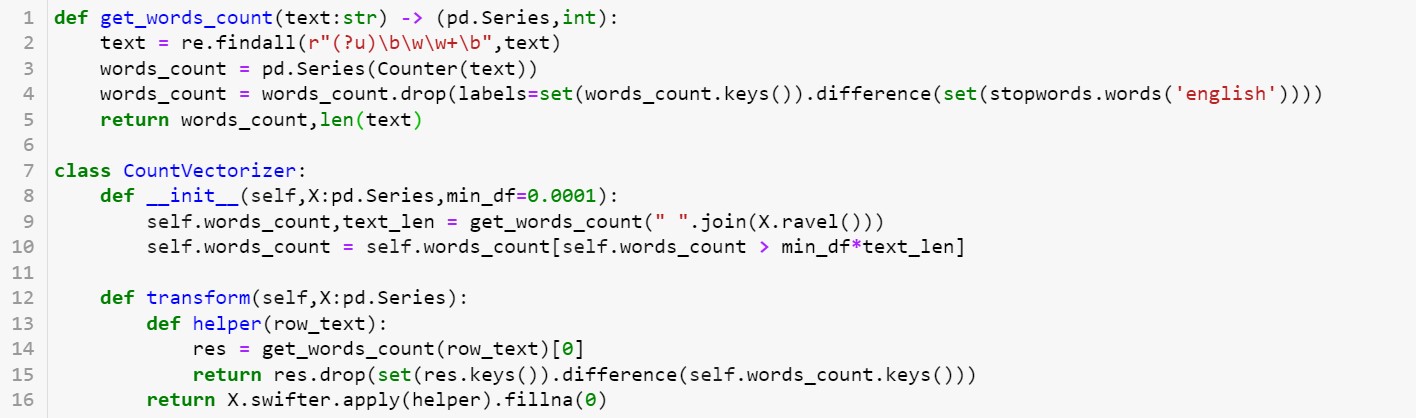
We split the data into train, validation, and test sets, setting 15% of the data for test, 15% for validation, and 70% for train.

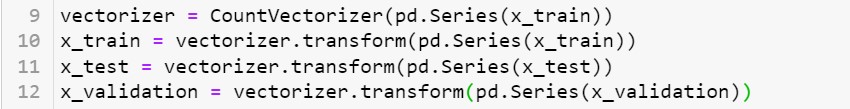


Chart, pie chart

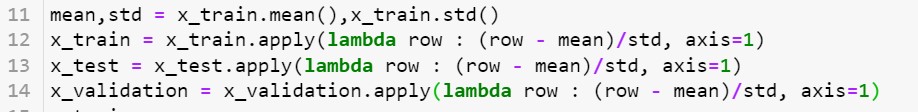
Description automatically generated

Then we generated features using “bag of words”:





And normalized:



To see if the data make sense we want to see the connection between the label and most frequent word in that label:

In the negative label:  
Chart, histogram

Description automatically generated

In the positive label:  
Chart, histogram

Description automatically generated

**Logistic regression**

First, we implemented the model as his simple version.

We train the model for 80 epochs with learning rate = 0.001, batch size = 100.

Text

Description automatically generated with low confidence

We plotted the loss value on the train and validation, in order to check we are not over fitting:

Shape

Description automatically generated

As can be seen from the plot, over fitting doesn’t seem to be a problem here.

This run gave an accuracy of 0.835 and the following results:

Chart, treemap chart

Description automatically generatedChart, treemap chart

Description automatically generated

We noticed the recall for 1 (bulling) is very low. We suspected is duo to the fact the data is imbalance: 73.2% of 0 and only 26.8% of 1, and that make the model tend to classify more zeros.

Chart, pie chart

Description automatically generated

We tried to over come this problem by over sampling – duplicating 1 samples in order to balance the data. We duplicated samples until we got a radio of #0/#1 = 0.9.

Chart, pie chart

Description automatically generated

we run the model again with 80 epochs and we got these results:

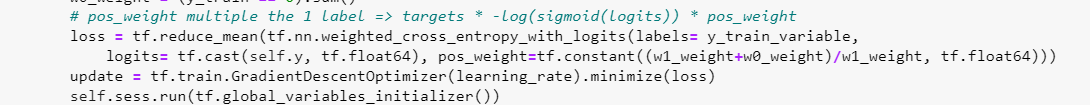
A screenshot of a cell phone

Description automatically generated with medium confidenceChart, treemap chart

Description automatically generated

The recall on 1 did increase dramatically, but the recall on 0 and precision on 1 decreased too mach. So we tried a different method to overcome the imbalance data problem – weighted loss. In the loss function, we gave a mistake on a 1 a higher weight then a mistake on a 0 sample. The weights are set to be 1 over the proportional part of the label.





This method gave less “extreme” results – the recall of 1 increased less, but the 0 recall only want down a little. As to be expected (we tackle with a trade-off between the recall and precision), the precision of 1 still decreased (but less than before). the accuracy stayed at 0.829.

Chart, treemap chart

Description automatically generatedChart, treemap chart

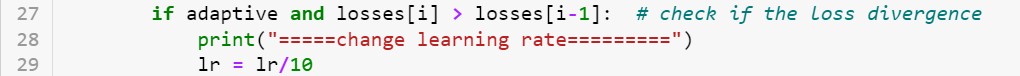
Description automatically generated

Shape

Description automatically generated

We decided to go through with this method, but tried to improve it by implementing a dynamic learning rate.

Each time the loss function was higher than on the previous iteration, we divided the learning rate by 10.



The results of this run were very close to the results with a static rate.

Chart, treemap chart

Description automatically generated

For the finale training, we concatenated the train and the validation sets, and trained the model on both.

Text

Description automatically generated with medium confidence

The results on the test set:

Chart, treemap chart

Description automatically generated Chart, treemap chart

Description automatically generated

Feature Importance – because we normalized the data, and this is a linear model we can see which feature got the largest weights for negative/ positive:

All the close to 1 weights:

Chart, bar chart

Description automatically generated

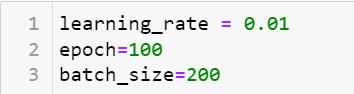
All the close to -1 weights:

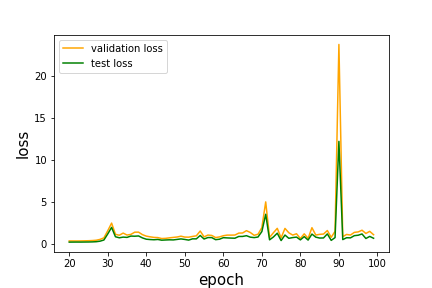
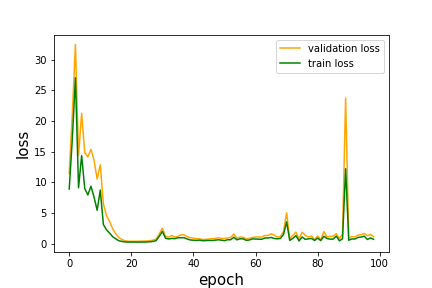
Chart

Description automatically generated

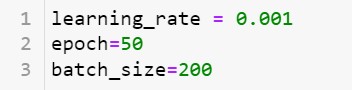
**MLP – multi layer perceptron**

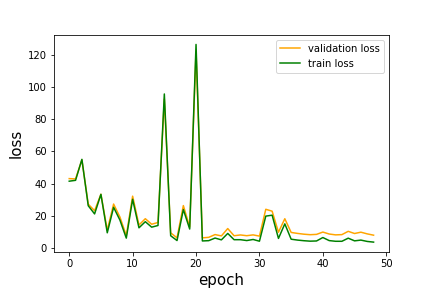
On our first try, we trained a simple MLP with hidden layers of sizes (100,60,20), learning rate of 0.01, and batch size of 200. We trained the network for 100 epochs. The graphs below show the change of the loss value of the train and validation sets over time.



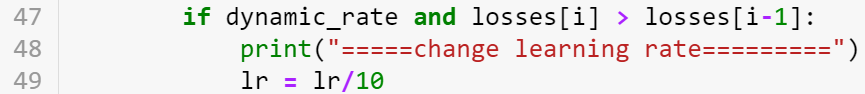


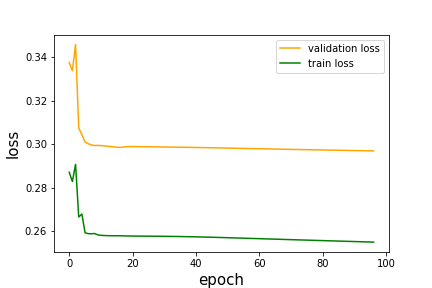
As can be seen, there is “noise” in the loss graph, and it as not going down after around 20 epochs. We suspect this is duo to too high learning rate. We try to re-train the network with learning rate of 0.001 (for 50 epochs this time).





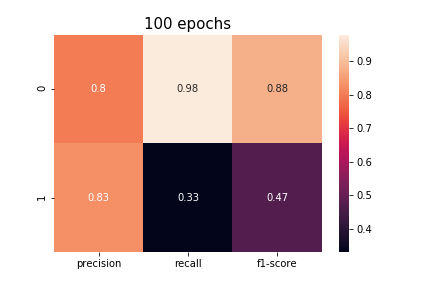
Lower learning rate did not fix the problem. The next thing we tried is implementing a dynamic learning rate – every time the loss is higher than the previous iteration, we divide the rate by 10.



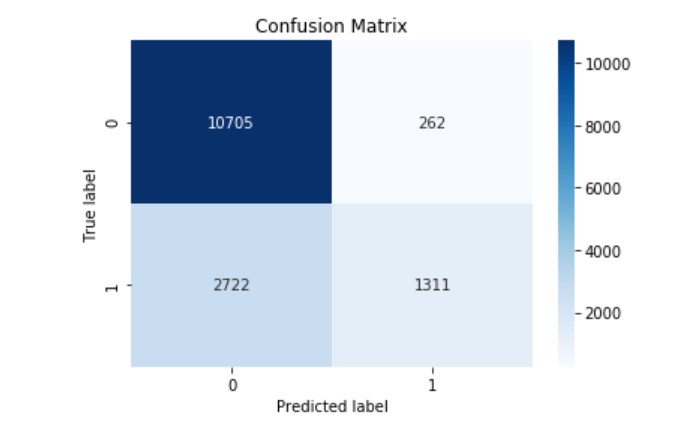


The dynamic learning rate did fix the problem – the loss is (almost) consistently going down.

Is immediately calls for more epochs of training. The results are:

 Chart, treemap chart

Description automatically generated

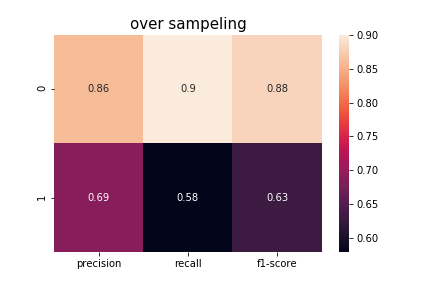
 Chart, treemap chart

Description automatically generated

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 duplicating 1 samples to get a ratio of #1/#0 = 0.9. this method yielded the following results:

Chart, pie chart

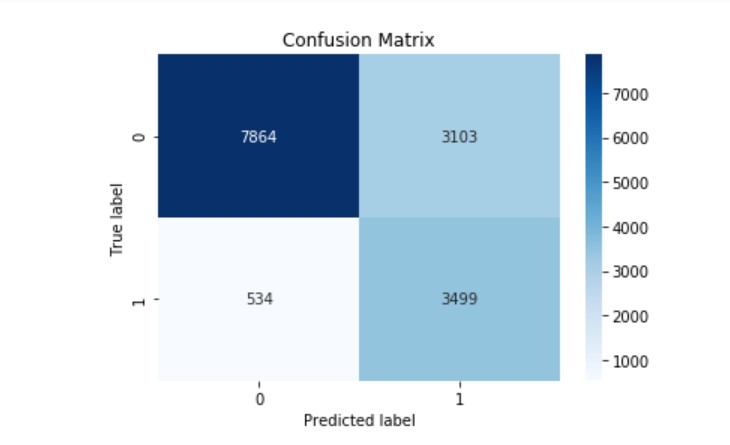
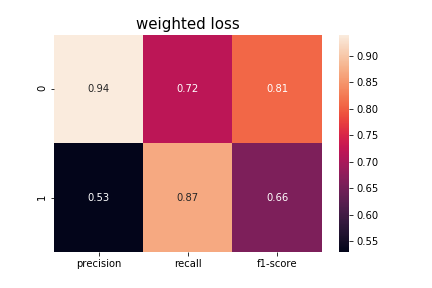
Description automatically generated



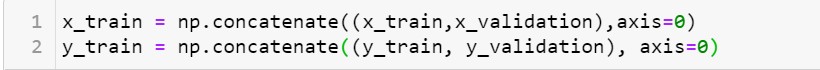
The recall for 1 did increase, at the cost of a big decrease in the precision.

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:



At the end, we trained out finale model on both the train and validation sets, and tested on test set:



The finale results on the test set are:

Accuracy = **0.7574**

