CS 6350 Project

Big Data Management & Analytics

Toxic Comment Classification

Names of students in your group:

Ch Muhammad Talal Muneer cxm180004

Chirag Shahi cxs180005

Number of free late days used: 2

Note: You are allowed a **total** of 4 free late days for the **entire semester**. You can use at most 2 for each assignment. After that, there will be a penalty of 10% for each late day.

Please list clearly all the sources/references that you have used in this assignment.

Toxic Comment Classification

Muneer, Ch Muhammad Talal
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, USA
cxm180004@utdallas.edu

Shahi, Chirag
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, USA
cxs180005@utdallas.edu

***Abstract*—There has been an exponential increase social media platforms and data in the 21st century. Due to media censorship and everyone not having access to traditional media outlets, people turn towards applications like Twitter, Facebook, Instagram etc. to voice their opinions. These platforms allow anyone to sign up and start pouring out their ideas. What these platforms lack is the filtering mechanism to check what the person is posting. Mostly people share their thoughts on politics, sports, current events etc. but some users tend to do more than that when they start abusing other people or make terrible comments. This effectively marginalizes a part of the community and shuts them down. These applications are trying as best as they can to counter these threats of abuse and harassment and create a safe space for people.**

Keywords—filtering mechanism, social media, abuses, outbursts

I. INTRODUCTION

In order to improve online conversations and protect against misuse people around the world are developing tools to filter natural language. In this project we study online toxic behavior of people and try to see if we can classify this data into multiple categories.

II. BACKGROUND WORK

A. Data Exploration

Twitter released its own dataset a couple of years that included more than 150,000 tweets. We use this data and to try to train a model that helps us filter these abusive comments. This is a multi label problem. There are a total of 6 categories:

- Toxic
- Severe Toxic
- Obscene
- Threat
- Insult
- Identity Hate

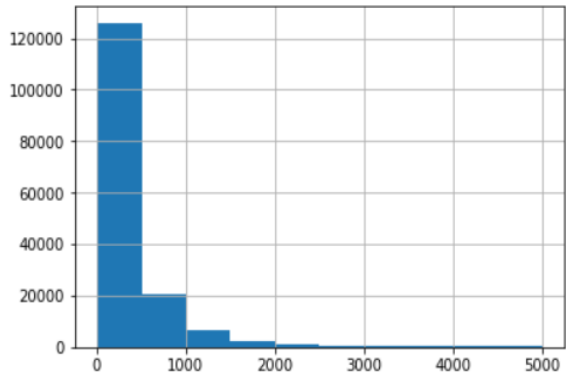
A lot of people have taken up this challenge and produced good models. These models achieve good accuracy (above 90% on test data).

B. Maintaining the Integrity of the Specifications

The template is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin in this template measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations.

III. PREPROCESSING AND FEATURE ENGINEERING

After looking at the data we saw that some most of the comments were of length from 0 to 1000 characters. The histogram on the right shows that. y-axis represent the number of tweets and x-axis shows the length of tweet.



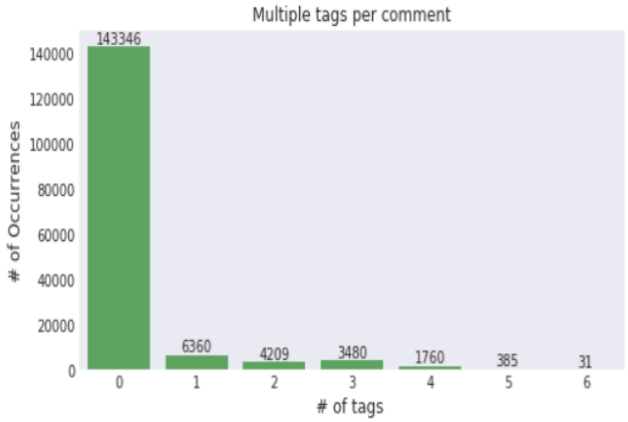
# of tweets vs length of tweets

Next we take a look at how the classes are distributed. The number of clean comments come out to be 89.83%. The number of comments of each class is listed below. We can clearly see that there is a class imbalance as some of the classes have too many comments while others have close to none.

```
Distribution of Comments
Toxic :  9.584448302009765 %
Severe Toxic :  0.9995550569965721 %
Obscene :  5.2948217407925 %
Threat :  0.2995531769557125 %
Insult :  4.936360616904074 %
Identity Hate :  0.8804858025581089 %
```

It is also observed that comments that are not clean have weird user IDs, IP addresses, slang and symbols that are not part of the natural language. This can be a problem as some words or symbols might mean the same thing but our model would consider them to be different. This would possibly cause overfitting as well. We also check for null values and they there are none.

Another problem that can be seen from the graph on the right is that there is a multi label class imbalance as well. Most of the comments have no tags. This will become a problem when we train our model as the higher number of clean comments will dominate the weights in the model.



In order to reduce the features we first remove the stopwords from all the comments. This however did not remove all the weird symbols and slang words hence we had to use regular expressions to remove them individually. We also had to trim spaces from each side and remove tab and end of line characters.

Then we tokenize the words and feed it to a Neural Network however the system exceeds the memory probably because we still have a lot of excess words that do not have any significance. Any model cannot take words we need to convert them into numerical format. We used HashingTF from the Spark machine learning library to give every word a specific value. A problem that arose was that some words come more often in all of the comments and hence they will dominate when the model is training, hence we used the TF-IDF method. This method gave significance to words that do not appear very often in the text and reduced the weight of words that appear a lot more times.

IV. Theoretical Study of Model Used

A. *Definition*

In order to solve the problem we build a neural network. This is fully connected network. This algorithm works in the same way as a human brain does. Every neuron receives some data processes it using some function to process. It then sends the data to the neurons that it is connected to.

B. *Working*

- Each edge between any two neurons has a weight that is initialized to a random value at first.

- Our numerical data is passed to the input layer of the network. It travels through and at the end produces a real number
- Then we determine how it is classified. This process is known as forward propagation.
- When the answer is misclassified the difference between the actual output and our output is calculated and the error is propagated back into the network to adjust the weights of the edges. This is known back propagation.
- For every data point one forward and back propagation is done and finally we get a model.

C. *Hyper Parameters*

There are number of hyper parameters that we need to adjust to get an even better model. They are as follows:

-
- Activation function: sigmoid, tanh, ReLu
- Number of layers in the network
- Number of nodes in each layer

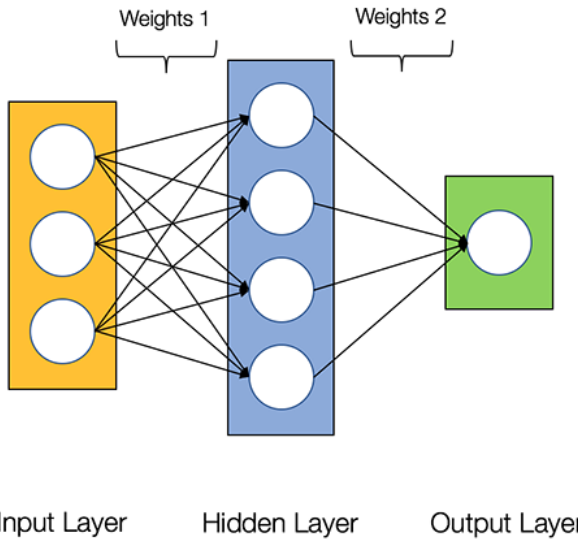
We try to make an educated guess on what possibly works and then change them if the accuracy does not increase. A drop out method can also be used where nodes are randomly dropped from the network forcing more load on the other nodes in the same layer. This technique has been proven to produce good results. However, we have not implemented this method.

D. *Accuracy*

The accuracy of the model can be measured using Root Mean Square or by calculating the percent of correctly classified data points.

V. EQUATIONS

A. *Neural Net*



Architecture of a 2-layer Neural Network[4]

The output $\hat{y}$ of a simple 2-layer Neural Network is:

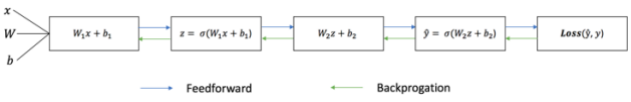
$$\hat{y} = \sigma(W_2\sigma(W_1x + b_1) + b_2)$$

[4]

The weights W and the biases b are the only variables that affects the output ŷ.
Naturally, the right values for the weights and biases determines the strength of the predictions. The process of fine-tuning the weights and biases from the input data is known as training the Neural Network.
Each iteration of the training process consists of the following steps:

- Calculating the predicted output ŷ, known as **feedforward**
- Updating the weights and biases, known as **backpropagation**
-

The sequential graph below illustrates the process.



[4]

B. *Feedforward*

The output of a 2-layer Neural Network is:

$$\hat{y} = \sigma(W_2\sigma(W_1x + b_1) + b_2)$$

[4]

Now we need a loss function to estimate how well our network performs.

C. *Loss function*

There are a variety of loss functions to choose from. In our model, we chose the root mean square error (RMSE) as our loss estimator.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Stock image

Our goal is to minimize this loss estimator by choosing a set of weights that minimize this.
Now we find a way to propagate the error back and update the weights and bias accordingly.

D. *Backpropogation*

Derivative of the loss function with respect to weights and biases determine the amount of change in our weights and biases.

The equations of derivatives of some activation functions used are as follows:

- Sigmoid: x * (1 - x)
- ReLu: 1 if x > 0, else 0
- Tanh: 1 – (x * x)

VI. Results

The matrix that was generated after the TF-IDF transformation was a sparse matrix. This was because we had chosen a large number of features and every feature was not present in every sentence. The feature matrix looked like this:

$$\begin{pmatrix} 1.0 & 0 & 5.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3.0 & 0 & 0 & 0 & 0 & 11.0 & 0 \\ 0 & 0 & 0 & 0 & 9.0 & 0 & 0 & 0 \\ 0 & 0 & 6.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7.0 & 0 & 0 & 0 & 0 \\ 2.0 & 0 & 0 & 0 & 0 & 10.0 & 0 & 0 \\ 0 & 0 & 0 & 8.0 & 0 & 0 & 0 & 0 \\ 0 & 4.0 & 0 & 0 & 0 & 0 & 0 & 12.0 \end{pmatrix}$$

It had a lot of zero values and when we trained on these values the results were not favorable. Hence we decided to reduce the number of features to 100. The matrix still was sparse but not as much as we had gotten earlier. The weights adjustment was still not very good because there we a lot of zeros in the matrix and the ones which we not zero were small because they had been normalized by the TF-IDF transformation. Whenever the neural net was trained on the feature set the zeros influenced the weight changes. The output at the end of model training was coming out to be very low. Another possible problem that could have happened was due to the sigmoid function. When there are multiple layers of neurons and many weights. After a certain number of iterations the change in weights get very low due to the sigmoid function. Hence, we tried using another activation function called ReLu which gave slightly better results. The next stop in parameter tuning was increasing and decreasing the layers and neurons. Increasing the number of layers was not helping that much as most of the input numbers had zero value. The ones that did not were not huge numbers. As these number moved along the network their values decreased because of the weights and by the time they reached the output they would be close to zero. By increasing the number of neurons in each layer we saw that the training phase started to take a lot of time and in the end the cluster ran out of memory. This was because it was a fully connected network with over 10,000 weights to tune.

VII. Conclusion

The class imbalance problem was difficult to handle. As seen by the results above the number of clean comments were very high (around 90%). Since there were more clean comments even during training the weights favored more towards them. The number of toxic comments did not have as

much of an effect on weight training. We can see this hypothesis pan out when we finally test out model. The number of misclassified instances of one (comment is toxic) are more than that of zero. The network mainly identified these instances as zero because of the imbalance in the dataset. One more reason could be that the words in toxic comments that differentiated them from clean comments were only a handful of words. Given that we already had a very sparse matrix. These handful of words only present in some instances and hence they did not influence the model as much as we had hoped. The overwhelming majority of clean words skewed the model towards them.

REFERENCES

[1] https://stackabuse.com/creating-a-neural-network-from-scratch-in-python-multi-class-classification/

[2] https://www.freecodecamp.org/news/building-a-3-layer-neural-network-from-scratch-99239c4af5d3/

[3] https://github.com/robmarkcole/Useful-python/blob/master/Numpy/Build%20a%20neural%20network.ipynb

[4] https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6

[5] UTD CS 6375 Fall 2018 Course Content (Assigment 3)

[6] https://docs.databricks.com

[7] https://spark.apache.org/docs/latest/api/python/index.html