Comment Abuse Tracker

Ho Yin Samuel Chan
hyc18@sfu.ca

Cole Stankov
cstankov@sfu.ca

Andrew Do
ada62@sfu.ca

Abstract

People are spending more time browsing the Internet, watching videos on streaming sites, and connecting with others through social networking sites. These online platforms allow people to share their thoughts freely and anonymously, which means that the online communities can become incredibly toxic if not well monitored. With the ever increasing amount of online users and social media platforms, it is becoming increasingly costly to monitor the online space. We will be looking at Comment Abuse Classification with Deep Learning by Chu, et al. (2) and apply its methods of identifying abusive behaviors and offensive language on Twitter by classifying the tweets. The data we used was classified by Crowdflower workers into three categories and was compared to our classifier's outputs. The models that we attempted to implement were long-short term memory cell (LSTM), Deep Neural Network with a rectified linear unit (DNN) and a linear classifier.

1 Introduction

1.1 Motivation

As more and more discussions move towards online platforms many public forums have been plagued with abusive behaviour such as cyber-bullying and personal attacks. However, determining whether or not a comment or post should be ”flagged” is difficult and time-consuming, and many platforms are still searching for more efficient and moderate solutions. This project will focus on automating the process of identifying toxicity by the classification of abusive comments on social media platforms such as Twitter. It will measure comments on social media to determine whether or not the comments are abusive or inappropriate, and if so, it classifies them. Furthermore, classifying these comments will increase user safety and improve online discussions.

The aspect that this project mainly focuses on is the evaluation metrics accuracy, recall(+), precision(+) and F-score. We will go more in depth with the evaluation metrics within the Experiments section. Being able to accurately determine whether or not a comment on social media should be flagged for the purposes of moderation is our first priority, and it represents the effectiveness of the product of this project. We will also examine the accuracy by whether or not the program can correctly classify comments with words or phrases that it has not encountered before in the training phase, or how close it can classify them.

Other aspects that we will be working on include interpretability and robustness. Interpretability discerns if a comment is an abusive one, and provides the reason why a comment is flagged as inappropriate. Without knowing what kind of abuse it is by interpreting the comment, we will not be able to classify it. Robustness tells us if all kinds of inputs are well handled. For example, if a comment contains a hashtag, which may or may not contextualize a it's contents, will the program still able to perform accurate classification, or will it have unexpected behaviors due to training data experienced with those specific hash tags? So robustness is a good measurement for the quality of the model.

1.2 Goal

Our goal for the project was to have the baseline models implemented as well as the more advanced models that were specified. The project infrastructure was the first to be finished. After that the first baseline model that was implemented which was the linear classification model, and we decided to change the least square loss function to a mean square error (MSE Loss) function instead. This was then followed by our implementation of a rectified linear unit deep neural network (DNN) using PyTorch. The third model that we attempted to implement was the long short term memory cells (LSTM) model, which contains a series if FORGET, LEARN, REMEMBER, and USE gates. We aimed to use this model to determine which words within a tweet are useful for abuse classification. However, after various failed attempts it was unable to function as intended.

2 Related Work

2.1 Comment Abuse Classification with Deep Learning(2)

The work of Theodora Chu, et al.'s paper explored the topic of social media and online forum platforms, and the best methods to identify and classify their increasingly common abusive behaviour that is associated with them. For our project we loosely based our ideas from their work, often drawing inspiration through their findings and the methods that they used.

Chu, et al. implemented two base models, one being a linear regression model with a least squared loss function and the second being a deep neural network with a logistic loss function. Originally we planned to implement both of these models as our baseline but as the project continued we decided against the linear regression model and implemented a linear classifier with a mean square lost function instead. As for their two primary models they implemented a recurrent neural network (RNN) with a long short-term memory (LSTM) cell and a convolutional neural network (CNN). We believed that implementing all three of these models would be too difficult within our given time frame and decided to choose only one which was the LSTM.

Within Theodora Chu, et al.'s paper they discussed their findings and what achieved their best results. As for embbedings they found that using word embeddings with their LSTM model achieved the best results comparatively to their attempt with character embeddings. However, the inverse was true with their CNN model. This is why we decided to first implement word embeddings for our own LSTM model and if time pertained then we would expand to try other options. What they found with their implementation was that both their LSTM and CNN model could achieve an accuracy of 94% on classifying their data correctly.

2.2 Automated Hate Speech Detection and the Problem of Offensive Language(1)

One of the main challenges that occurs when identifying toxic comments on social networking sites is how to differentiate between hate speech and offensive language, because the two categories have overlapping areas. Many of the lexical detection methods classify sentences just by identifying certain words or phrases in the sentences as hate speech or simply offensive language, and these methods often are not able to distinguish between the two categories. So this research conducted by Thomas Davidson, et al. used a data set which contain not only the sentences, but also labels which classify each comment, to train and test their models. The labels are put on by CrowdFlower users as humans can classify sentences by their contexts, so that the models does not classify sentences by depending on just part of the sentences, but can learn to take each sentence as a whole.

This is also the research from which we get the data set for our project. Since it is very difficult to distinguish between hate speech and offensive language, we decided to follow the definition from Davidson, et al.'s research. Hate speech is defined as "language that is used to expresses hatred towards a targeted group or is intended to be derogatory, to humiliate, or to insult the members of the group". (1) As we want the classification of our models to be accurate, we included the approach of this research, and aimed to remove ambiguity of the two categories while implementing our models.

2.3 Racial Bias in Hate Speech and Abusive Language Detection Data sets(4)

Although hate speech and offensive language detection has become popular in the natural language processing field, and there are many prior related works which explored this topic, these established systems can be biased because of cultural backgrounds and the use of languages. The bias can bring negative impacts to even the people who the systems want to protect, by penalizing victims who used abusive-like languages in context that is not offensive. These biases can also lead to a difficult choice while classifying these certain languages to be inappropriate and can possibly penalize people wrongly. On the other hand, not classifying them as inappropriate will block us from identifying potential hate speech and abusive words.

The research that was done by Thomas Davidson, et al. addressed this issue with biased hate speech and abusive language detection by examining racial bias in a few different data sets that are annotated for hate speech and abusive language. They found that the evaluation metrics can produce false positive cases that are caused by the model overgeneralizing the training data. We believe that this bias issue also happens to our implementation of the Comment Abuse Tracker where we might classify tweets as inappropriate by certain words, while the tweets actually are not intended to be abusive.

3 Model

3.1 Linear Classifier

For this project we decided to build two baseline algorithms. The first one is linear classifier involving the number of curse/hate words in a sentence which will map to an abuse classification once it reaches a specific threshold. The linear classifier was implemented with a PyTorch base neural network. We chose to use PyTorch for this implementation since we believed that it would be slightly simpler to implement rather than the TensorFlow counterpart. The loss function that was used for our linear classifier was a mean square error function. It was first carried out with a least square loss function, much like what was done within the workings of Chu, et al. (2), however, we decided to take a slightly different approach due to some issues that arose. With the change we hoped that it would provide a more general machine-learning baseline. The MSE function is as follows:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - y_i^p)^2}{n}$$

3.2 Rectified Linear Unit Deep Neural Network

The second baseline makes use of Deep Neural Network (DNN) implementation supplied by the PyTorch library which allows us to tune the number of hidden layers to optimize test accuracy. Our DNN implementation also uses the Rectified Linear Unit (ReLU) from the PyTorch library when feeding the data forward through the neural network. Much like the first baseline model, the linear classifier, we used a mean square error (MSE) Loss function. Originally we planned to use a logistic loss function which is the default for TensorFlows deep neural network. However, we found that the ”NLLLoss” function required integer valued labels which did not fit into our float valued labels which come from the Crowd Flower labelled data. Ultimately we decided it would be much easier to use the same MSE loss function as used in the linear classifier.

3.3 Long Short Term Memory Cells (LSTM)

At the beginning of this project we observed that projects with similar objective and scope had used LSTMs to produce results which improved greatly from baseline models, this meant that attempting to classify online comments using an LSTM was one of our first ideas. However, we quickly ran into incompatibility issues related to the data, it's labels, the general structure of the project and the structure of our chosen learning library (PyTorch). Due to time constraints we ultimately could not complete this section of the project, we did however explore multiple approaches which will be discussed in the next three sections of the paper.

3.3.1 LSTM Sentence Embedding Approach

Our first attempt at the LSTM model involved a similar approach to how the ”Linear Classifier” & ”Rectified Linear Unit Deep Neural Network” models were approached. These approaches involved gathering word embeddings from tweets and creating a sentence embedding either by applying a ”Word average” or ”Weighted word average” to create a single vector representing a sentence. The LSTM was run, however it produced undesirable results which included 0% precision and 0% recall across all classification categories. We immediately recognized that something was wrong here and eventually found the issue to be in incorrect assumption on how the LSTM used memory, because we had coalesced words in a sentence into a single embedding, the LSTM was learning the order in which the training data was being fed into it. The order of training data is definitely not something we want an NLP application to be learning and thus scrapped this approach and used what we learned to guide our next approach.

3.3.2 LSTM Sequence of Word Embeddings Approach

Our second attempt at the LSTM model consisted of creating a sequence of word embeddings. The process of turning a tweet into a sequence of glove embeddings involved getting rid of words that don't contribute to the message, these include twitter handles and hashtags which can be identified by a leading ”@” or ”#” character respectively. Then the words are matched with their glove vectors, this information is found and saved. Since spelling is expected to be imperfect in online environments, in order to better handle out of vocabulary words (OOV) a robust word recognition was required. Robust word recognition can be achieved by using a character embedding scheme found in the paper (5) by Keisuke Sakaguchi et al., this character embedding involves creating one-hot vectors for the leading character, middle characters and trailing characters in a word. The robust character embedding allowed the model to take into account slang words which would otherwise be discarded. The tweets were then ran against the LSTM model, this is where we discovered the problem with this approach. The problem experienced involved the shape of the output from the forward function, for a tweet with N word embeddings and 3 output classifications, a Pytorch Tensor with shape:

$$1 \cdot N \cdot 3$$

This tensor shape proved to be problematic as the the expected output had label dimensions:

$$1 \cdot 3$$

This meant that a classification was produced for every word in the tweet, but what we wanted was a classification for the entire phrase. Attempts were made to collect data from the series of output tensors, these included: extracting the last answer; taking the average answer; and taking the most severe classification of the outputs in this order:

$$hate\_speech > bad\_words > safe$$

However all of these approaches triggered errors in the LSTM model when the output was used to calculate the loss, this error occurred for a variety of built in loss functions including: MSELoss, NLLLoss, CTCLoss and CrossEntropyLoss. We deduced from this failure that we were likely not structuring the output labels correctly which prompted our final approach.

3.3.3 LSTM Restructured Labelling Approach

For our final attempt at the LSTM model we wanted to try restructuring the tweet labels, this was in part due to the failures of getting LSTM to work with the structure of the project in previous attempts. For the baseline models labelling involved 3 classifications and hence a label vector of length 3, with each value representing the share of Crowd Flower votes for the respective categories: hate_speech; bad_words; safe. For each tweet label the following was true:

$$\sum_{c=hate\_speech}^{safe} c = 1.0$$

We restructured the labels such that a single integer value represented the Crowd Flower classification (max of the vote categories) and each category was assigned an integer label:

$$hate\_speech = \mathit{0}, bad\_word = \mathit{1}, safe = \mathit{2}.$$

When this new labelling strategy was used in the LSTM model it produced out of range predictions where the output landed at values 2.

3.3.4 LSTM Summary

Ultimately we were unable to implement the LSTM due to issues related to the incompatibility between the way our project was structured and the way LSTMs are used in PyTorch. In the future we may explore the use of Keras' LSTM model as it's a much easier to use and popular machine learning framework.

4 Data

For our data we are using a Twitter comments data set found on a GitHub project (3), the data contains the comment itself and labels of whether it is an offensive comment or not attached to each comment. There are six attributes for each set of data, the first attribute is the index, the second is the number of people who reviewed the comment, the third is the number of people who judged the comment to be hate speech, the fourth is the number of people who judged the comment to be offensive, the fifth is the number of people who judged the comment to be neutral, and the last is a class label for the comment according to the majority of the vote. An example of each classification input are as follows:

An input classified as *hate speech*:

*354,3,2,1,0,0,”””@jayswaggkillah: Jackies a ****** #blondeproblems”” At least I can make a grilled cheese!”*

An input classified as *bad words*:

*59,3,0,3,0,1,""""..All I wanna do is get money and **** model *******!"" - Russell Simmons"*

An input classified as *neither*:
369,3,0,0,3,2,"""@metroadlib: colored contacts in your eyes? blinders on mine. cause i can't see you at all."" Lmao"

The data set has three categories including hate speech and offensive language, which are being separated as two different categories but they often overlap with each other. This can create ambiguity for classification amongst the tweets. Therefore, having CrowdFlower users labeling the tweets removes the ambiguity and makes the classification more accurate. The following chart breaks down our statistics of our data that we used:

Category	# of tweets
Total tweets	23190
Hate speech	1349
Bad Words	17892
Neither	3949

Table 1: Dataset Statistics

The reason why we chose Twitter comments as our data set is that Twitter comments are usually replies, so they tend to be short coherent statements rather than random comments. Another reason is that Twitter is one of the most popular social media that people from around the world is using nowadays, so the data we obtain has sufficient representativeness and quantity.

We split the data set into two in our experiment, 20% of it is used for the training phase, and the rest is used during for the testing phase. Our prediction on the data during the testing phase, is the argmax of a 3-long array where each value represents the probability of a specific classification:

$$hate\_speech = 0, bad\_word = 1, safe = 2.$$

We implemented functions that load the data file and get tweets from the file. The tweets are then stripped of any unnecessary punctuation and all letters are replaced by lower case letters. We then create features from tweets by getting word embeddings from pre-trained glove file which is outsourced. After obtaining the embeddings, we average them over a tweet, in order to create labels of whether the tweet is a hate speech, contains bad words, or is safe. The labeling is done by comparing the number of votes the tweet gets in these three categories.

An issue we discovered in our experiment is that the results we obtained for hate speech had very low precision(+), recall(+) and F-score which means our models are having difficulties spotting any tweet that should be categorized as hate speech. This issue was caused by testing/training partition not being randomized since we are currently using the first 20% of the data for training and the rest for testing. After a some attempts of partitioning the data set in different ways, we eventually used 20% of each category as the input for training, instead using the first 20% of the file, and we saw improvements in the accuracy of the hate speech and save tweets categories. Another reason that caused the issue was that the glove file did not contain certain common offensive words which can be used to determine whether the tweets are hate speech or offensive.

5 Experiments

5.1 Embeddings

Since the beginning of the project we knew that we should use word embeddings instead of character or sentence embeddings due to the work that was found in Comment Abuse Classification with Deep Learning (2). They found that the best results can be achieved when word embeddings are used with their models, specifically the LSTM. Therefore, We also planned on attempting different word embedding techniques such as the weighted average embedding and shift embedding. When we attempted to use these embeddings with punctuation we found our results to be worse in comparison to when punctuation is removed. We realized that this could only be achieved if unnecessary punctuation was fully removed, so we built a function to perform this task. In order to compare the difference we made sure to implement our infrastructure to have the functionality to easily switch back and forth between the two embedding techniques.

5.2 Epoch, Learning Rate, Hidden Layers and Training Partition

We altered our baseline models to see what produces better outcomes. Our ideas included changing the number of hidden layers that are in both our deep neural network and linear classifier. We predicted that the ideal number of hidden layers would be around eight, so we used that as our baseline for our first implementation, and then we slowly increased the numbers progressively in order to obtain a better result. As we increased the number of hidden layers, we did observe improvements in the performance until reaching an optimal value. We then noticed a decline in performance extending the number of hidden layers past that.

We also decided to change the learning rate of our models, which was originally set to 0.01, since we observed from previous work learning rate was set to this value. Similar to hidden layers we planned to increase the values comparing our evaluation metrics to decide what is optimal for each model. Similarly, from other related projects, online resources and previous work, we determined to set epoch to 10 because we noticed reasonable performance and output from them. And we increase it progressively to observe changes in behaviour and result as well.

Lastly we did experiments with different sizes of training partition. Since we had a relatively low data set size we wanted to keep the training partition as low as possible to leave enough data for actual testing, while having enough data to achieve better learning for the models. However, it was difficult to find a suitable ratio to balance the two. So we tried different ratios that ranged between 10% to 30% of the data set in order to get the best output, and at the end we settled on 20% for training the models.

5.3 Loss Functions

A variety of loss functions were experimented with when developing the Rectified Linear Unit Deep Neural Network and LSTM models. For The RLUDNN we experimented with MSELoss and NLLLoss functions. The MSELoss function proved to be a simple yet reliable loss function producing respectable results. The NLLLoss function although used by other torch project examples was incompatible with the outputs from our model and labelling strategy. The LSTM model used the MSELoss and NLLLoss functions as well as the CTCLoss and CrossEntropyLoss which were suited for classification, however despite making modifications to the labelling to better suit the loss functions, they all proved to be incompatible with the original labelling strategy as well as permutations of the labelling strategy involving changing from float & integer data types as well as permutations on the structure of the tensor outputs.

6 Results

6.1 Evaluation Metrics

We decided on four evaluation metrics to use for the comparison of our models. Originally our project only contained accuracy as our only evaluation metric, however, we noticed that when we calculated other metrics such as recall or precision their score was a substantially lower than accuracy's. The evaluation metrics that we used are as follows:

$$Accuracy = \frac{TP + TN}{Total}$$

$$Precision(+) = \frac{TP}{TP + FP}$$

$$Recall(+) = \frac{TP}{TP + FN}$$

$$F - Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

6.2 Linear Classifier Results

With our linear classifier model we noticed that we achieved the best results when learning rate was increased substantially to the 0.1-0.12 range compared to the starting value of 0.01. When this was increased we noticed an increase in all of hate speech's evaluation metrics as well as safe tweets with only a minuscule drop in bad words precision(+) and recall(+). Without the increase in the learning rate hate speech's f-score and recall were at 0%. Increase hidden layers did not affect the linear classifier model so it was left at its base value. Epoch was found to increase safe words and bad words metrics as it increased from the base value of 10 until reaching 20. Setting epoch to a value greater than 20 did not increase any of our evaluation metrics for any of the three categories and would slightly decrease them the higher they would go. Lastly training partition was always set to 20%. If the partition was altered by increasing or decreasing it, we would receive a substantial negative impact on our results in all three categories.

Hate Speech	Score
Accuracy	94.168%
Precision(+)	37.500%
Recall(+)	0.278%
F-Score	0.551%

Table 2: Linear Classification Score on Hate Speech

Bad Words	Score
Accuracy	80.786%
Precision(+)	82.614%
Recall(+)	95.110%
F-Score	88.423%

Table 3: Linear Classification Score on Bad Words

Safe Tweet	Score
Accuracy	85.399%
Precision(+)	60.910%
Recall(+)	39.842%
F-Score	48.173%

Table 4: Linear Classification Score on Safe Tweet

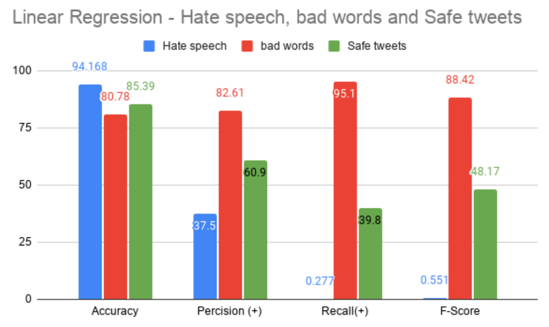

Figure 1: Linear Classifier Metric Graph

The above tables 2-4 and figure 1 show our raw output from our optimal performance of linear classification with 0.12 learning rate and 20 epochs, and training partition 20%.

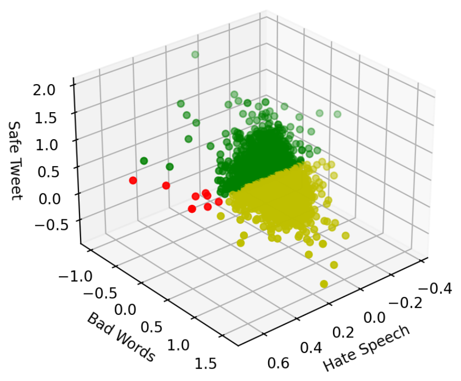

Figure 2: Linear Classification Plot (Predicted)

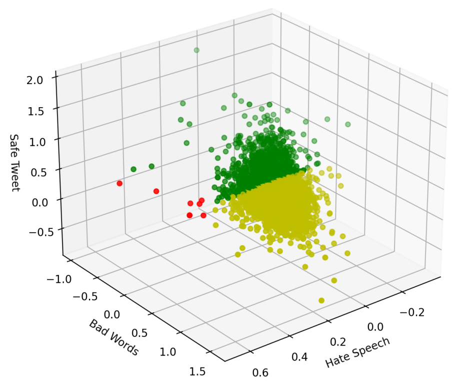

Figure 3: Linear Classification Plot (Actual)

Figure 2 & 3 show the same plots where position indicates the strength of each classification but figure 2 uses colours based on predicted classification of tweets while figure 3 shows colours based on the actual classification of tweets.

6.3 Rectified Linear Deep Neural Network Results

As for our DNN model we noticed that the best results were achieved when learning rate was around the base value 0.01-0.03, unlike our linear classifier. If learning rate was increased we noticed a overall drop for bad words and safe tweets metrics. Hidden layers had a large impact on our DNN, where increasing them to roughly 20-30 shown the best results with in all evaluation metrics. Changing the hidden layers from 20-30 would have slight impact on each evaluation metric but nothing considerable. As for epoch our DDN reacted very similarly to our linear classifier where the optimal value was 20 and increasing it or decreasing it would slightly affect the output. The training partition was set to 20% as well for similar reason as the linear classifier.

Hate Speech	Score
Accuracy	94.179%
Precision(+)	0.000%
Recall(+)	0.000%
F-Score	0.000%

Table 5: DNN Score on Hate Speech

Bad Words	Score
Accuracy	80.166%
Precision(+)	83.918%
Recall(+)	91.903%
F-Score	87.729%

Table 6: DNN Score on Bad Words

Safe	Score
Accuracy	84.186%
Precision(+)	53.926%
Recall(+)	49.113%
F-Score	51.407%

Table 7: DNN Score on Safe Tweet

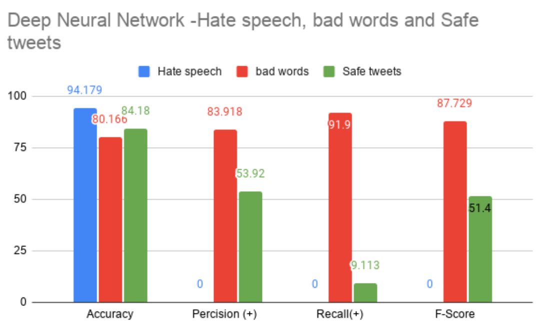

Figure 4: DNN Metric Graph

The above tables 5-7 and figure 4 show our raw output from our optimal performance of Deep Neural Network with 0.03 learning rate, 20 hidden layers, 20 epochs, and training partition 20%.

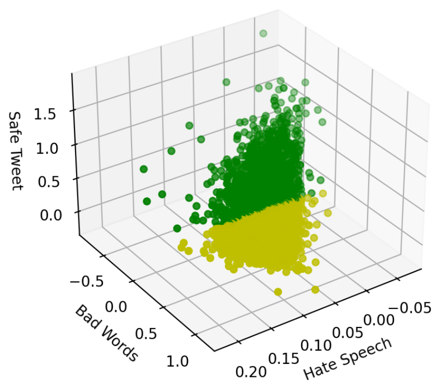

Figure 5: DNN Classification Plot (Predicted)

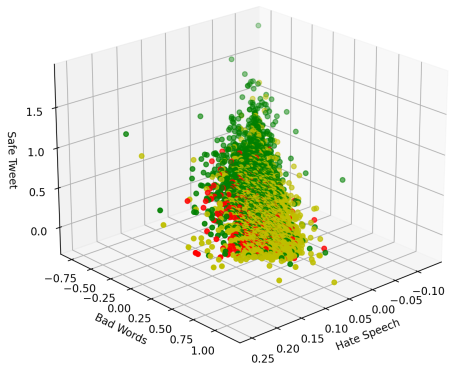

Figure 6: DNN Classification Plot (Actual)

Figure 5 & 6 show the same plots where position indicates the strength of each classification but figure 5 uses colours based on predicted classification of tweets while figure 6 shows colours based on the actual classification of tweets.

7 Analysis

7.1 Accuracy

Accuracy is relatively high when compared against all other metrics, however it should be noted that accuracy is not an important metric since we're dealing with a use case where the class imbalance is quite high since hate_speech tweets are much less likely to occur in testing and training data sets compared to the other 2 classifications.

7.2 Precision

The precision for hate_speech was 37% in linear classification and 0% in DNN which indicates that the linear classifier has identified some features related to hate speech. However it lacks context which is important when considering hate speech. As an example consider these tweets which have similar words but different meanings:

"I love gay people so much!"
"Gay people show too much love!"

The precision for bad_words classification is high across all models, this suggest that it's easy to find out if a tweet has bad words, this makes sense as certain words should trigger a tweet to be classified as containing bad words regardless of how those bad words are phrased.

The precision for safe tweets classification was lower than bad_words. An explanation for this phenomena is that in the other 2 classifications we're looking features that identify their membership. By contrast a tweet which is considered "safe" can be described as all other tweets. As such we should consider starting as predictions as "safe" initially and elevate their classification to bad_words or hate_speech if they contain certain features.

7.3 Recall

Recall was generally quite similar to the values observed in precision, one notable exception however was seen in the hate_speech classification for the linear classifier model which saw 37% precision and 0.3% recall. These values indicate that very few predictions were made for this classification but when they were made they were likely to be correct, this means that for future experiments we should consider lowering the barrier of selection for hate_speech.

7.4 F-Score

F-Score is a more suitable measurement than accuracy for our results, we noticed that bad_words and safe had good F-Scores indicating the models used were sufficient in those categories, however hate_speech was lagging behind by a considerable margin, this means that more complicated models should be used to classify this classification of tweets.

8 Conclusion

After testing the Linear Classification model and Rectified Linear Deep Neural Network model on Tweeter comments, we were able to reach up to 88% F-Score in classifying the tweets. Although our models are not perfect and still have rooms for improvement, by conducting the experiments we came to the conclusion that deep learning is an useful technique to identify toxic online comments. Thus deep learning can create a healthier online environment for people to have discussion and express their views. We tried classifying tweets by picking up words in the tweets. But it is hard to determine whether the tweet is hate speech or just contain inappropriate words just by using words, from the analysis of results we found that context was important for hate speech classification. One thing that could've been fixed in our approach was to consider the structure of our data and labels before diving into determining suitable models such that we run into less issues with implementation.
Some things we did not have time to finish but would of been useful: The PyTorch LSTM proved to be too problematic to implement given the structure of our project, however we noticed similar projects successfully use Keras LSTM, it would've been great to see how context and word ordering affect hate_speech precision but this would however require overhauling the flow of data within most of the code base. Considering how the data was labelled (1 tweet = 1 classification) a tf-idf model may provide a more accurate baseline, however this too may involve restructuring the code base.

References

[1] Automated hate speech detection and the problem of offensive language. https://arxiv.org/pdf/1703.04009.pdf. Accessed: 2020-10-23.

[2] Comment abuse classification with deep learning. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/reports/2762092.pdf. Accessed: 2020-10-22.

[3] Hate speech and offensive language. https://github.com/t-davidson/hate-speech-and-offensive-language/blob/master/data/labeled_data.csv. Accessed: 2020-10-23.

[4] Racial bias in hate speech and abusive language detection datasets. https://arxiv.org/pdf/1905.12516.pdf. Accessed: 2020-10-31.

[5] Robsut wrod reocginiton via semi-character recurrent neural network. https://arxiv.org/abs/1608.02214. Accessed: 2020-11-30.