Toxic Comments Classification Challenge

Authors: LIM WEI LIANG BARRY (SCI2) , MUHAMMAD AFIF B MOHD ALI(SCI2), LEE YI QUAN (SCI3,COM3), ISABEL ANG YUET TING (SCI2) , LEE JUN HAN (BRYAN) (CEG3), TAN CHEE WEE (COM3)

Abstract

This project aims to come up with a model to identify toxic comments on Wikipedia from clean ones. On top of identifying toxic comments, we also wish to develop a model that distinguishes and classifies toxic comments into different categories. By investigating the effects of both traditional supervised models (Logistic Regression, Naive Bayes) as well as modern approaches (unsupervised learning, word embedding, recurrent neural network) on toxic comment classification, we manage to find some success in correctly classifying toxic comments. We also looked at how spam in training data affected our model's performance.

1. Introduction

Maintaining a certain level of decorum online is part of our responsibility as a user of the Internet. However, the anonymity that comes with it has empowered many to speak out and confront others, sometimes rather aggressively. It is thus up to moderators of websites to sieve out content that serves to cause harm to others. That is easier said than done when information flow is on a scale of 151 exabytes per month in 2018 and is expected to double in 2021[1]. It is hence essential to come up with an accurate and automated way to flag out comments that are deemed to be undesirable.

In Kaggle's Toxic Comment Classification Challenge, we are provided a dataset which consists of user comments and discussion from Wikipedia's editorial section. Using this, we attempted to deal with the scenario as described above. Each comment also came with corresponding labels consisting of "*toxic*", "*severely toxic*", "*obscene*", "*insult*", "*threat*" and "*identity hate*". For instance, if a comment is deemed to be "*toxic*" and an "*insult*", it will be labelled as such; correspondingly, if a comment does not fall under any label, then it would not be labelled at all.

The task at hand is a multiclass labelling problem; we have to decide which categories does each comment fall under.

2. Data Exploration/ Preprocessing

Data Cleaning

When using supervised methods with TFIDF and word2vec, we first cleaned the data by removing IP addresses, converting all words the lowercase, removing punctuation (except !), numbers. Even though there may be more capitalized words in toxic comments, converting all words to lowercase simplifies the analysis on categorizing the comments. White space is trimmed because it is often unnecessary.

When modelling with LSTM network, the keras tokenizer takes care of the cleaning.

Preprocessing

The area of text/topics classification which is frequently studied in the area of Natural Language Processing (NLP). Unlike most other Machine Learning classification tasks where raw data is already in the form of numerical data, data in the form of text is not useful to computers. So one of the underlying questions was how to transform textual information into numerical data. For different classification algorithms, we have explored several popular NLP methods to transform text into useful data.

Bag of Words (BoW)

The Bag of Words model is a feature extraction method that describes the occurrence of words within a document.

An example will be scikit-learn's Term Frequency-Inverse Document Frequency (TFIDF) vectorizer. Here, document refers to a single text file and corpus refers to the entire collection of documents.

For all unique words in the corpus , TFIDF calculates the product of Term Frequency (TF) and Inverse Document Frequency (IDF) , where

$$\text{TF} = \frac{Number\ of\ times\ word\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ document}$$

and

$$\text{IDF} = \log_2\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ that\ contains\ word}\right) + 1$$

Note that Add-1 smoothing is applied on the calculation of IDF and L2 normalisation is applied on the resultant product vector. We can see that the addition of IDF to the calculation will assign low scores to words that occurs frequently across documents regardless of context.
This is hence a significantly more useful method compared to scikit-learn CountVectorizer[2] , as this will aid us in removing common words such as "the" , "a" , "is" etc. , so that we could better sieve out important words that contributes to the toxicity of comments.

[1] Figures from https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html

[2] Scikit-learn 's CountVectorizer only calculates the Term Frequencies of words.

Embeddings
Another method of transforming text into numerical data is through the use of word embeddings. Word embedding, as its name suggests, “embeds” words from the vocabulary into real vector space, transforming each word into a single vector. Having vector representations of words is desirable since we can retain and preserve meaning of some of the words represented by their respective vectors in relation to other word vectors. For instance, we can perform the following vector operation for words “king” - “man” + “woman”. The resulting vector is close to the word vector representing “queen”. Also, unlike bag of words implementation, there is a significant dimension reduction from being vocabulary-sized to a much smaller one, typically in the range of 100-300.

One such algorithm to efficiently derive a word embedding is word2Vec. The vectors are first randomly initialised. For each word pair, present in the corpus, the algorithm samples k (a tuneable hyperparameter) other word pairs that are not present in the corpus. The dot-product of each of the word pair is then taken into a sigmoid function, signifying the probability of the word context pair appearing. The objective function then tries to maximise the probability of the pairs present while minimising probabilities of the pairs not present. This thus explains why words with high similarity contextually tend to have a highly positive inner product value.

Each comment can then be transformed into a vector by averaging the sum of the word vectors of each word present in the vector. However, if the comment is long, this might cause some problem because the overall effect of summing words might dilute the effect of more important words in the phrase.

Keras Tokenizer (used with LSTM model)
The tokenizer is initialised to turn all words to lowercase and to filter all special characters out, as we are only interested in the words in each document (1 document = 1 training example)

Calling *Tokenizer.fit_on_texts(train_comments)* where *train_comments* is the training examples in the training set, creates a dictionary of words consisting of several meta-data:

1. **Tokenizer.word_counts**: A dictionary of words and their counts.
2. **Tokenizer.word_docs**: A dictionary of words and how many documents each appeared in.
3. **Tokenizer.word_index**: A dictionary of words and their uniquely assigned integers.
4. **Tokenizer.document_count**: An integer count of the total number of documents that were used to fit the Tokenizer.

Each training example is then turned into a sequence of integers, where each integer is unique to each word in the training example, via the function *Tokenizer.texts_to_sequences*. The mapping between words and their uniquely assigned integers are found in **Tokenizer.word_index**

We then padded all the training examples to the same length to ensure that all the training examples have equal dimensions (padding is the process of assigning ‘0’ to other dimensions which has no words, after defining the max length of a training example vector). This facilitates training the LSTM model where every input vector must be of the same length.

3. Metric

The evaluation metric used in this task is the Area Under Receiver Operating Curve (AUROC). AUROC calculates the true positive rates (TPR) against the false positive rates (FPR) and hence is a common measure of accuracy for binary classification problems, such as this one[3]. TPR is calculated as $\frac{True\ Positive}{True\ Positive + False\ Negative}$, while FPR is calculated as $\frac{False\ Positive}{True\ Negative + False\ Positive}$, using values from the confusion matrix. The TPR and FPR are computed against many different thresholds and combined into a single metric, which is then used to plot the ROC curve. The area under the curve is finally calculated. The resulting values range from 0 to 1 - the closer it is to 1, the better the model is at the classification problem.

4. Problem Approach

Since this is a Multi-label classification problem, there are several notable methods[4] such as Problem Transformation, Adapted Algorithm and Ensemble approaches. For this task , when using supervised methods, we will transform it into multiple single-labelled classification problems - that is to assume that the labels are independent of each other and do classification on all the labels separately.

5. Model Implementation

Since this is a supervised classification problem, we started with supervised approaches with various feature extraction methods.

1. Naive Bayes with TFIDF
2. Logistic Regression with TFIDF
3. Logistic Regression with Word2Vec

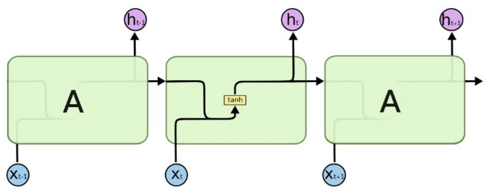
We then explored neural networks employing the LSTM layer. From the results, we chose to go with a neural network with a LSTM layer as our final model.

Our model consists of an input layer, an embedding layer, a LSTM layer, dropout and pooling layers, and an output layer.

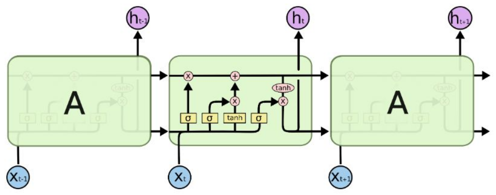
LSTM
Long-Short-Term Memory (LSTM) is a variant of the Recurrent Neural Network (RNN), composed of LSTM units/layers, as opposed to a single tanh layer.

[3] What does AUC stand for and what is it?
https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it
[4] Multi label classification methods
https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/

They are a special kind of RNN capable of learning long-term dependencies, as they avoid the long-term dependency problem faced by traditional RNNs.



The repeating module in a standard RNN contains a single layer. [5]



The repeating module in an LSTM contains four interacting layers.

The core of LSTM is the cell state, which is the horizontal line running through the entire chain. Each gate comprises of a sigmoid neural net layer and a pointwise multiplication operation. The gates have the ability to remove or add information to the cell state, optionally letting information through depending on the output of the sigmoid layer (0 = let nothing through, 1 = let everything through)

The repeating module in an LSTM consists of four neural network layers:

1) *Forget gate layer*: decides the information to throw away from the cell state. (1 = remember, 0 = forget)
2) Input gate layer: decides which values to update.
3) *Tanh layer*: creates a vector of new candidate values that could be added to the state
4) *Output gate layer*: outputs the computed result.

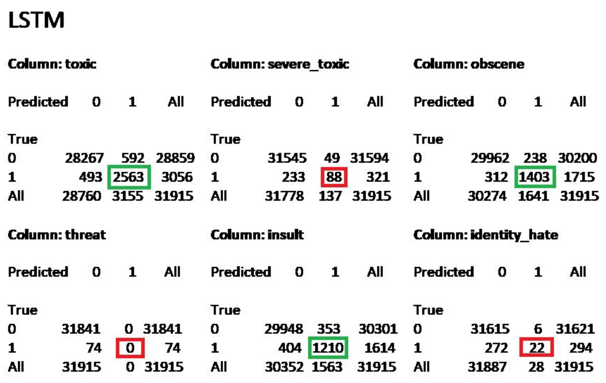
This model is particularly suited to build a language model due to its capability to retain information far longer than traditional LSTMs, as context is important in language tasks such as predicting the next word based on the previous one. As the required context gets further away from the point where it is needed to know, traditional RNN starts breaking down due to "forgetfulness".

6. Results

On our test set, our model achieve a auroc score of 0.7028 and a Kaggle score of 0.9687.

7. Evaluation

While that is an acceptable score to us, we feel that there are some limitations to our model. Below is the confusion matrix of the LSTM model.

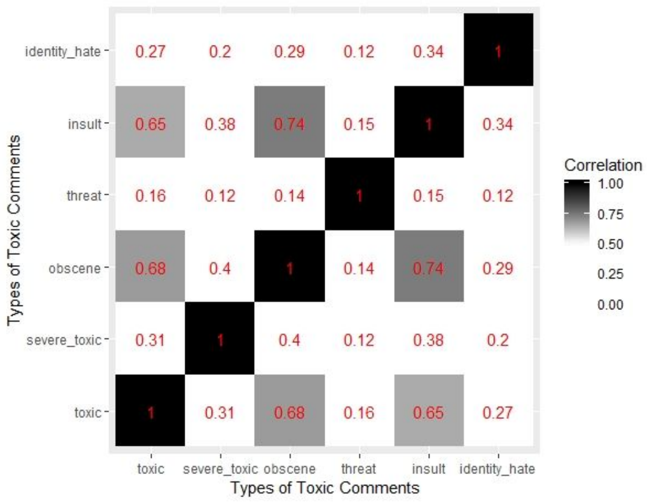

LSTM

```
Column: toxic                  Column: severe_toxic           Column: obscene

Predicted     0     1    All   Predicted     0    1    All    Predicted     0     1    All

True                           True                           True
0         28267   592  28859   0         31545   49  31594    0         29962   238  30200
1           493  2563   3056   1           233   88    321    1           312  1403   1715
All       28760  3155  31915   All       31778  137  31915    All       30274  1641  31915

Column: threat                 Column: insult                 Column: identity_hate

Predicted     0     1    All   Predicted     0     1    All   Predicted     0     1    All

True                           True                           True
0         31841     0  31841   0         29948   353  30301   0         31615     6  31621
1            74     0     74   1           404  1210   1614   1           272    22    294
All       31915     0  31915   All       30352  1563  31915   All       31887    28  31915
```

ROC_AUC score on test set : 0.7028

Kaggle score : 0.9687

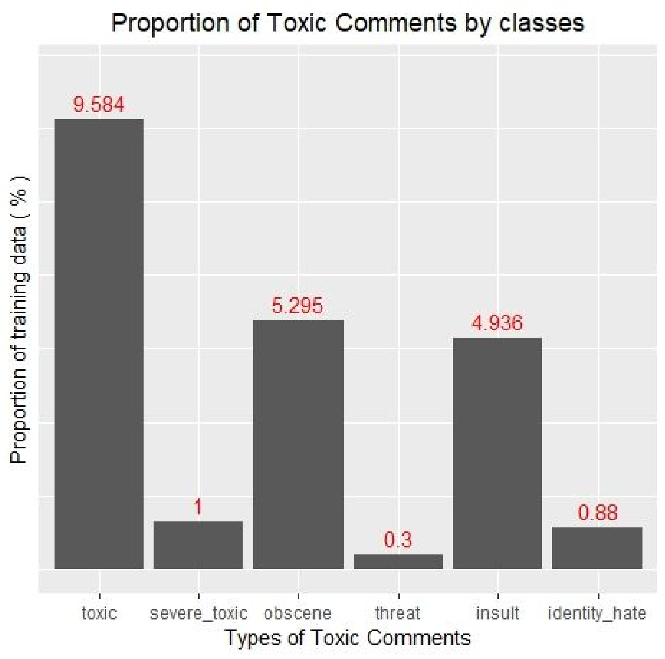
As you may have realised, the model is not able to predict any threat comments at all and did badly on identity_hate and severe_toxic. This is a huge problem and can be explained due to the correlation of the labels.



As per the figure above, the model did well for toxic, obscene and insult comments which have high correlation between them. While it performed badly for the rest due to its relatively low correlation score.

Another possible reason for why the model performed much worse for threat, identity hate and severe_toxic might be the fact that there is too few examples of comments that is of the abovementioned labels.

[5]Diagrams from http://colah.github.io/posts/2015-08-Understanding-LSTMs/



This seems to be the trend since sever_toxic, identity hate and threat each takes up less than 1% of the total data whereas the better performing labels each occupies at least 4% of the entire dataset. We hypothesise that having too way few data severely hampers the model's ability to learn the characteristics of the categories properly.

8. Discussion

We will now discuss the reasons for us choosing LSTM as our final model. Figure 1 through 3 in the appendix contains the results and confusion matrices of the rest of the models we have tried,each trained on a 80-20 split of the training data.

Naive Bayes vs Logistic Regression(TFIDF)

As expected, Logistic Regression performed better than Naive Bayes. Naive Bayes, being a generative classifier, learns the model of joint probability,P(x,y), before applying Bayes' rule to predict the posterior probability, P(y|x). On the other hand, Logistic Regression, being a discriminative classifier, learns the posterior probability directly. The indirect calculation of posterior probability in Naive Bayes, assumes that learned model is reflective of the actual model and is the reason why Naive Bayes does not perform as well as discriminative classifiers on large datasets.

Is Logistic Regression(TFIDF) the best?

From the scores, we can see that Logistic Regression with TFIDF vectoriser performs the best. However, a closer look at the confusion matrices will reveal the fact that Logistic Regression(TFIDF) only performs well due to its ability to classify clean comments well and not toxic comments. This is a huge problem as the consequences of wrongly classifying toxic comments as clean is catastrophic.

This problem surfaces due to the skewed and imbalanced dataset provided by Kaggle. The training data contained much more clean comments (89%) than toxic comments which could result in poor detection of toxic comments in the first place. Because many approaches use cost reduction approaches (e.g. Gradient Descent) to eventually find the appropriate weights to assign to classifiers, this can create a huge problem in the case of skewed datasets because the algorithm can maintain a low cost even if it classifies all of its examples to the majority class.

This led us to exploring for better options - a model that can classify toxic comments well, through word embeddings.

TFIDF vs Word Embeddings

Based on the confusion matrices, we can see that there is a significant improvement in classifying toxic comments when using word embeddings, in both word2vec and LSTM embedding layer. In this task, where context and sentiment are important, word embeddings might be a better choice as it captures the semantics between words.

Logistic Regression(Word2vec) vs LSTM

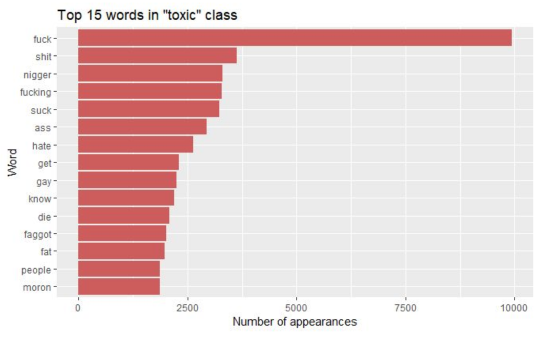
While results of Logistic Regression(Word2vec) looks better in general, we chose to go with LSTM because it only does poorly due to the small number of toxic comments in certain classes. Also, LSTM manages to learn the relationships between the classes well and provides a more holistic approach to the problem, as compared to the supervised methods we implemented which classifies the classes independently. We believe that with more training examples of toxic comments, our model will surpass supervised methods.

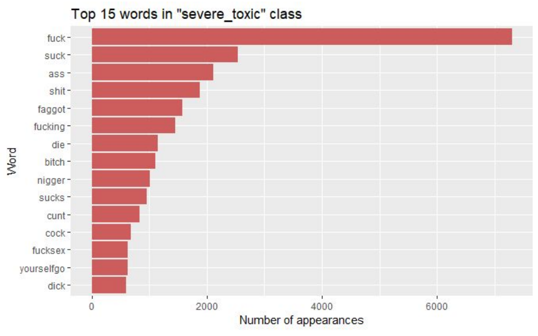
8. Additional Remarks

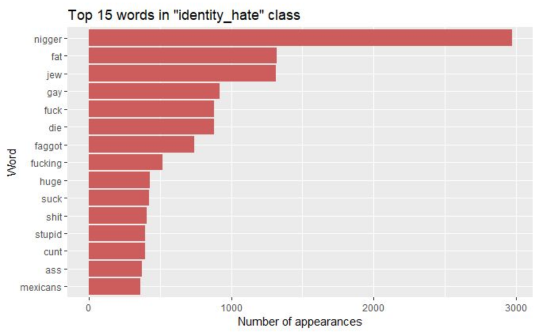
Relationship between classes

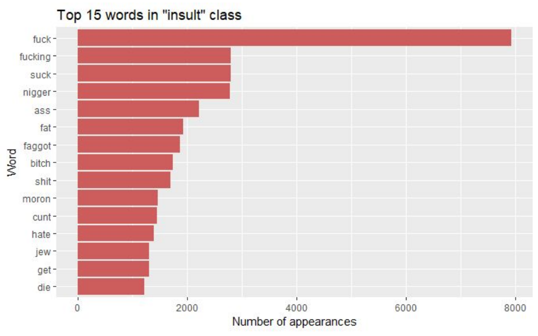
We will now look into the usage of words with respect to each class.

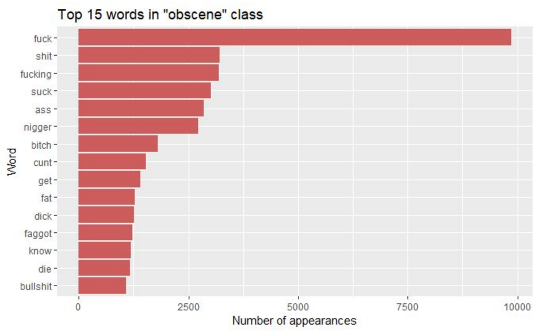
From the heat maps (see Evaluation), it is clear that the classification classes are not independent of one another. This can be further seen when plotting the bar charts of each class as below :

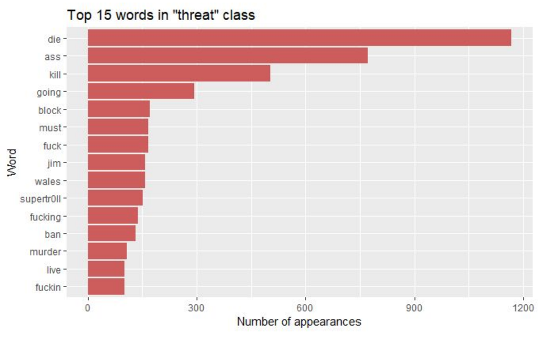












Given that all comments classified as *"severe_toxic"*, *"identity_hate"*, *"insult"*, *"obscene"* are also classified as *"toxic"*, we expect to see some duplicated words with the aforementioned categories and that of the *"toxic"* category. This is seen in the most popular word in the *threat* class of "die" also present in the *toxic* category. However, it appears that common swear words are prominent throughout all the different classifications. This shows that recognising the presence of the word in the comment is insufficient as the sole means to categorise the comments into the various classes - we would see many comments recognised as category 3 or 4 or even all the categories. Hence, it is important to learn the relationships between the classes well.

Something else that is noteworthy regarding the relationship between the different classes is that just as in a practical sense where it is difficult to measure the intensity of toxicity in a comment, it is difficult for methods to assign words with similar sentiments into vectors and analysing the words in space At best, one is able to see from past data what types of comments are deemed as a certain class and try to draw a relationship between that.

Spam

We also intend to analyse how spam comments affect our model.

To detect spam comments within the dataset, we used the criteria:

$$Spam = \frac{Number\ of\ unique\ words\ in\ comment}{Total\ length\ of\ comment}$$

After much consideration and checks using the dataset provided, we settled with a threshold of 0.3. The threshold is important because if it is set is too low, it would not be able to correctly detect for spam, whereas if it is too high, non-spam comments would be wrongly detected as spam.

At this threshold value, comments with values lower than 0.3 would be deemed spam. The training set had 519 spam comments, while the testing set had 1092. We tested this spam-free dataset against our LSTM with word embeddings model.

Our initial hypothesis is that removing spam comments will result in our model performing better - better predictions of true positives and negatives, and overall better score in Kaggle.

Surprisingly, removing the spam comments in training data resulted in our model having slightly lower classification accuracy. Training without spam comments resulted in 96.87 % accuracy, as compared to 96.67 %.

We suspect the reason for this was because the spam comments we removed were mostly toxic comments. Removing them from our dataset adversely affected our model's learning capabilities. Given that our dataset was already skewed, removing the toxic spam comments worsened our model's ability to learn and detect for toxicity.

Further checks of the training dataset found that 178 of the 519 spam comments were clean comments. Furthermore, examples of spam comments that were removed were "HATE YOU HATE YOU HATE YOU HATE YOU HATE YOU HATE YOU…" and "aryour retarded your retarded your retarded your retarded your retarded your retarded your retarded…". These comments are amongst the less vulgar ones, but are important to our model's learning process.

We concluded that considering how valuable toxic comments were in our dataset, removing them would be counter-productive to our model's capabilities. Also, there is insufficient data to properly test and evaluate the effects of spam comments on our final model e.g. testing the effects of removing non-toxic spam comments only.

9. Conclusion

Through this project, we have come up with a model that performs decently. There still remains much room for improvement because the model is rather insensitive to categories where the distribution is extremely skewed. Furthermore, in most online comments, we see that the quality of the dataset obtainable is inherently low; there are many instances of misspelled words. There could also be many different spellings of a word the user was meaning to say; for instance the word “you”, which can be spelt in short for “u” or elongated for instance “yooou” or “yoouuuuuuu”. This variation in spelling could result in poor classification.

With that said, one of our greatest takeaways from this project was the opportunity of carrying out a machine learning experiment on a problem that we were interested in. We challenged many assumptions that we had (for example, that removing spam comments would make the dataset more relevant) as well as learnt about common industry techniques to solve word sentiment related problems. From vectorizing words through a bag of words approach to word embeddings which assign words vectors based on similar meanings.

We also researched and learnt from material online to come up with several approaches and solutions and we were able to choose a combined approach which performed well but beyond all these results, just the process of tackling problems that we have not been exposed to before using the machine learning techniques that we have learnt through this module has been invaluable.

10. References

[1] Abhishek (2018) Approaching (Almost) Any NLP Problem on Kaggle - [Online].
Available at:
https://www.kaggle.com/abhishek/approaching-almost-any-nlp-problem-on-kaggle

[2] Beng (2018) Classifying multi-label comments - [Online].
Available at:
https://www.kaggle.com/rhodiumbeng/classifying-multi-label-comments-0-9741-lb

[3] Howard (2018) NB-SVM Strong Linear Baseline - [Online].
Available at:
https://www.kaggle.com/jhoward/nb-svm-strong-linear-baseline

[4] Jagan (2018) Stop the S@#$ - Toxic Comments EDA - [Online].
Available at:
https://www.kaggle.com/jagangupta/stop-the-s-toxic-comments-eda

[5] Kumar (2018) Logistic Regression TFIDF - [Online]. Available at:
https://www.kaggle.com/sudhirnl7/logistic-regression-tfidf

[6] Wang and Manning (No date) Baselines and Bigrams: Simple, Good Sentiment and Topic Classification - [Online]. Available at
https://nlp.stanford.edu/pubs/sidaw12_simple_sentiment.pdf

[7] Waseem et al (2018) Understanding Abuse: A Typology of Abusive Language Detection Subtasks

[8] Ng and Jordan (No date) On Discriminative vs Generative classifiers: A comparison of Logistic Regression and Naive Bayes [Online]. Available at:
https://ai.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf

11. Appendix

Fig 1

Naïve Bayes with TFIDF

Column: toxic

Predicted	0	1	All
True			
0	28766	93	28859
1	3017	39	3056
All	31783	132	31915

Column: severe_toxic

Predicted	0	1	All
True			
0	31482	112	31594
1	301	20	321
All	31783	132	31915

Column: obscene

Predicted	0	1	All
True			
0	30096	104	30200
1	1687	28	1715
All	31783	132	31915

Column: threat

Predicted	0	1	All
True			
0	31709	132	31841
1	74	0	74
All	31783	132	31915

Column: insult

Predicted	0	1	All
True			
0	30196	105	30301
1	1587	27	1614
All	31783	132	31915

Column: identity_hate

Predicted	0	1	All
True			
0	31503	118	31621
1	280	14	294
All	31783	132	31915

ROC_AUC score on test set : 0.5111

Kaggle score : 0.7506

Fig2

Logistic Regression with TFIDF

Column: toxic

Predicted	0	1	All
True			
0	28695	164	28859
1	2313	743	3056
All	31008	907	31915

Column: severe_toxic

Predicted	0	1	All
True			
0	30870	724	31594
1	138	183	321
All	31008	907	31915

Column: obscene

Predicted	0	1	All
True			
0	29857	343	30200
1	1151	564	1715
All	31008	907	31915

Column: threat

Predicted	0	1	All
True			
0	30961	880	31841
1	47	27	74
All	31008	907	31915

Column: insult

Predicted	0	1	All
True			
0	29940	361	30301
1	1068	546	1614
All	31008	907	31915

Column: identity_hate

Predicted	0	1	All
True			
0	30928	693	31621
1	80	214	294
All	31008	907	31915

ROC_AUC score on test set : : 0.8798

Kaggle score : 0.9709

Fig3

Logistic Regression with Word2Vec

Column: toxic

Predicted	0	1	All
True			
0	24921	3980	28901
1	1463	1550	3013
All	26384	5530	31914

Column: severe_toxic

Predicted	0	1	All
True			
0	26343	5284	31627
1	41	246	287
All	26384	5530	31914

Column: obscene

Predicted	0	1	All
True			
0	25740	4501	30241
1	644	1029	1673
All	26384	5530	31914

Column: threat

Predicted	0	1	All
True			
0	26358	5467	31825
1	26	63	89
All	26384	5530	31914

Column: insult

Predicted	0	1	All
True			
0	25801	4566	30367
1	583	964	1547
All	26384	5530	31914

Column: identity_hate

Predicted	0	1	All
True			
0	26315	5327	31642
1	69	203	272
All	26384	5530	31914

ROC_AUC score on test set : : 0.7502

Kaggle score : 0.8306