Classification of Comment Toxicity

Zhihao Guo
zhihguo@umich.edu
School of Information,
University of Michigan
Ann Arbor, Michigan

Shangquan Sun
sunsean@umich.edu
School of Information,
University of Michigan
Ann Arbor, Michigan

Yicheng Zhou
yczhou@umich.edu
School of Information,
University of Michigan
Ann Arbor, Michigan

Abstract

In the recent years, unfriendly comments on the internet have been very common. The toxicity of comments is defined as "anything rude, disrespectful or otherwise likely to make someone leave a discussion". Such toxicity will definitely ruin the internet environment. The purpose of this project is to detect toxicity of comments, which mainly extracted from online conversions. We first embedded each comment into a vector and tried some traditional machine learning models such as SVM and XGBoost to determine whether the comment was toxic and how toxic it was. Then we built a neural network called Long-Short Term Memory (LSTM) and compared these models. It turned out that LSTM outperformed traditional models by significant amount and shown excellent classification power. In addition, we built a user interface that took input sentences from users and determined its toxicity. Finally, We gave some representative words for toxicity and non-toxicity.

Keywords text classification, machine learning, natural language processing, sentence embedding, feature-text mixing data, unbalanced data

ACM Reference Format:
Zhihao Guo, Shangquan Sun, and Yicheng Zhou. 2019. Classification of Comment Toxicity. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Nowadays, with the development of Internet, more and more people are using Internet as a place to express ideas and comments. Internet environment is actually full of freedom because some of them believe that they do not need to take responsibility for what they have said. Thus, a lot of toxic comments appear on the Internet. For example, some people joke about other's races and gender, and some people are very rude and leave offensive comments that make readers feel uncomfortable. Those comments ruin the stability of Internet environment and hurt people's feelings. As a result, our goal is to build a model that recognizes toxicity and minimizes this type of unintended bias. During the process, Dataset labels will be used for identity mentions and to optimize a metric designed to measure unintended bias.

To solve this problem, we started with sentence embedding that converts each sentence into a vector that machine learning algorithms could understand. After that, we first worked on the traditional machine learning models, where we used tri-grams with TF-IDF values for embedding. We tuned a linear SVM as baseline and compared it with other models such as XGBoost. However, the results only shown a little improvement to our baseline. Then we built a LSTM neural network and combined glove, crawl and GoogleNews in the embedding. The neural network generated very good classification performance and outperformed other models significantly, with 94.4% accuracy as well as high precision and recall rate. We further analyze the problem and summarized why we should use neural network over traditional machine learning models for these certain types of questions.

1.1 Data Inspection

Our data set is from the Kaggle competition: Jigsaw Unintended Bias in Toxicity Classification.[1] The data is available directly in csv format and has already been divided into train and test files, where train dataset has 1.8 million records and the test dataset has 97.3 thousand records.

There are totally 1804873 instances of data samples, with each containing not only documents of comments, i.e. comment_text, but also some other attributes about the comment, such as obscene, identity_attack and sexual_explicit, and some features about the commenter such as male, female, homosexual_gay_or_lesbian.

A special character about the data is that it is very imbalanced. In both datasets, the ratio of non-toxic comment versus toxic comments is about 12:1. Specifically, training data contains 1660540 non-toxic samples and 106438 toxic samples. The public test data contain 89649 non-toxic samples and 5649 toxic samples. The private test data contain 91671 non-toxic samples and 7671 toxic samples.

[1] https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The respond is called "toxicity", which is scored manual among 0 to 1. In order to classify whether a comment is toxic or not, we set up threshold of 0.5 to determine the toxicity.

2 Methods

We have tried several classical machine learning algorithms in solving the problems. Since some models such as FastText and TextCNN are not fine tuned, in the report, we will mainly discussed about the performances of and comparisons among SVM with linear kernel, XGBoost, and LSTM.

2.1 Traditional Machine Learning Models

2.1.1 Word Embedding

Word embedding is essentially to transfer a sentence of words, which is human language, to a sequence of numbers that models are able to understand and process. There are a lot of word embedding techniques, and here I chose to combine the ngrams method with TF-IDF values.

An n-gram is a contiguous sequence of n items from a given sample of text or speech. For each sentence, it will collect all the single words, pair of words, and triple of words. Each n-gram is also labeled with the number of times it appeared in the corpus, which is all the comments in the dataset. Since there will be too many possible n-grams, we need to specify a vocabulary size so that it only picks the top n-grams that are most frequent in the corpus. In other words, using n-gram method will convert each sentence into a vector of n-grams, where each n-gram is labeled by its count in the corpus. Put all comments in the train dataset together, it will be a matrix with row being comments and column being n-grams.

However, simply using n-gram has a drawback. Since the method represents n-gram by its count, it will favor stopping words or words that appear very frequently but not help differentiating different comments apart. For example, if all the comments are talking about president election, then keeping word "president" will not tell you whether this comment is toxic or not. To prevent this, I replace the count value by the TF-IDF value of the n-gram.

The TF-IDF value will penalize words for appearing in too many comments, in which case they might just be the common topic but have nothing to do with the toxicity. By combining n-grams and TD-IDF together, we make sure that the n-grams we keep will commonly appear only in certain portion of the comments but not existent in every comment, and those n-grams will very likely be related the toxicity of the comment.

2.1.2 Linear SVM

Based on the embedding matrix, we tuned a SVM with linear kernel. Since SVM uses the closest data points from two labels to the hyper-plane to do the classification, it is immune to the imbalanced data by nature. We calculated the classification accuracy, precision, recall and f1-score to evaluate the classifier.

2.1.3 XGBoost

Other than SVM, which uses distance as the classification criterion, we implemented decision tree methods such as XGBoost. The advantage of using decision tree is that it draws non-linear decision boundary which might be closer to the reality in our problem. We used the same metrics we used for SVM and we also drew the ROC curve and obtained the AUC value.

XGBoost Data Balancing

Decision tree is vulnerable to imbalanced data since it will make the tree to favor the dominate group when splitting. To balanced, we will reassign the weights based on the population ratio of the groups and blend the weights into the gradient and second order gradient.

2.2 Neural Network

2.2.1 Word Embedding

Comparing to traditional methods, LSTM should use a different kind of word embedding method, since the word position and sequence in a sentence should be preserved so that LSTM could be able to understand contextual information of a phrase and sentence. TF-IDF could not preserve contextual relation when transforming sentence to a vector representation. Therefore, two popular embedding tools are employed, i.e. GloVe [2], and Crawl [1]. Crawl is a "2 million word vectors trained on Common Crawl", and GloVe is "an unsupervised learning algorithm for obtaining vector representations for words". Here it denotes a pre-trained word vectors by GloVe algorithms. Both of the methods generate a vector of 300 dimensions for a word. Two pre-trained word vectors are concatenated to form a matrix containing 328389 word vectors of 600 dimensions.

Since the maximum of document length is 220 words, training data could be transformed into a 220 × 1842770 matrix with each (i,j)-th entry being the word's index representing a word appearing at i-th document and j-th position of the document. Since there are totally 328389 number of words in the corpus of training and test documents, there are totally 328389 index ranging from 0 to 328388, with each index uniquely representing a word. Then the embedded vectors and pre-trained word vectors are together fed into LSTM.

2.2.2 Sample Weight Reallocation

Besides text data, some auxiliary and helpful identity information could be used. To leverage them, we decide to reallocate sample weights by evaluating samples' identity information. First of all, the values of identity columns are set to be boolean value with threshold of 0.5. Then the sample weights are initialized to be one for all samples. Then there are three steps of reallocating weights. First, sample weights are added by the number of "True" in identity entries for each sample, indicating the more special identity one commentator has, the higher training weight the sample has. Second, For those ground-true toxic samples, sample weights are added by the number of "False" in identity entries for each sample, indicating the more normal identity one toxic commentator has, the higher training weight the sample has. Third, For those ground-true non-toxic samples, sample weights are added by 5 times of the number of "True" in identity entries for each sample, indicating the more special identity one non-toxic commentator has, the higher training weight the sample has. Finally, sample weights are normalized by divided by its mean. The logically reallocated sample weights are also fed into LSTM.

2.2.3 LSTM

Then we utilize LSTM and train the model by batch size of 512 and 4 epochs. The architecture of the LSTM is shown in Figure 7.

The framework of LSTM, the sample reallocation method and the embedding method are inherited from [3].

3 Results and Evaluation

3.1 Linear SVM

The metrics of our baseline is shown in the heatmap below. From the heatmap, we observed that the SVM had very great performance on label 0, which is non-toxic comments, while the recall for label 1– toxic comments are very low. This shows that the among all the toxic comments in the test dataset, only 30% is actually predicted as toxic. And the overall f1-score was only affected by the low recall. The poor performance of the model on toxic comments are mainly due to the imbalanced data.

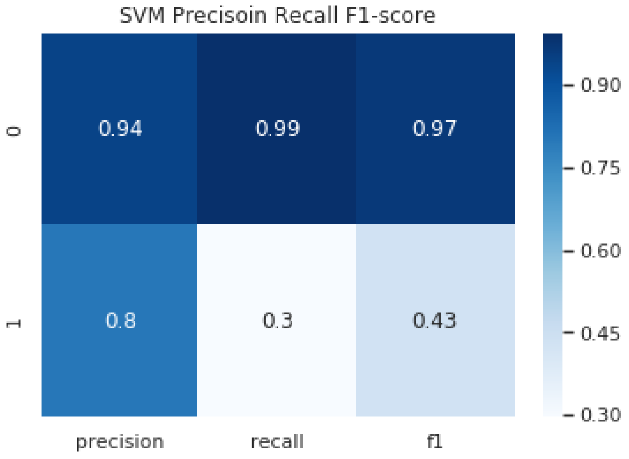


Figure 1. Fine tuned SVM Heatmap

3.2 XGBoost

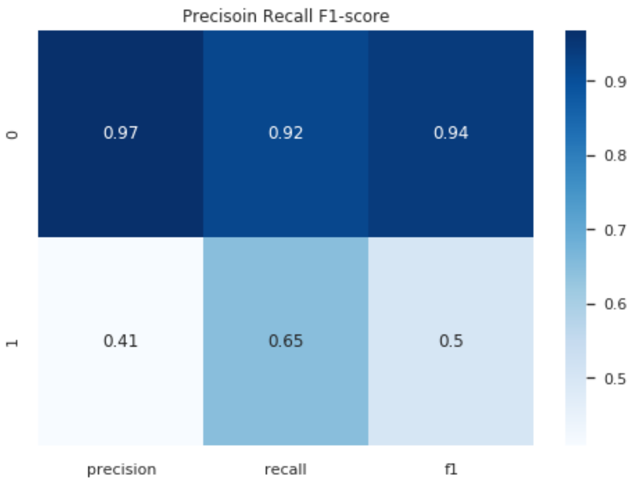


Figure 2. Fine tuned XGBoost Heatmap

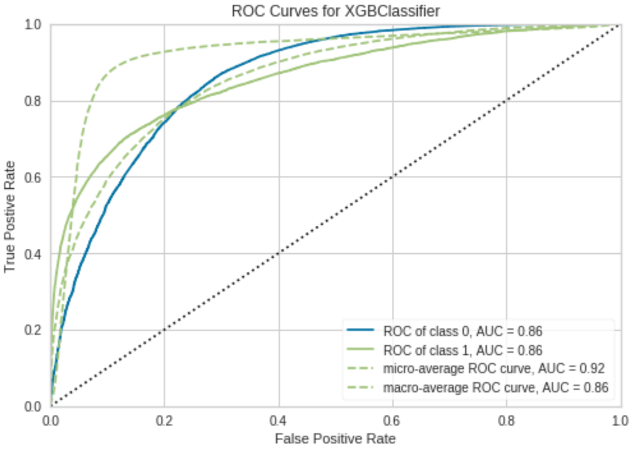


Figure 3. XGBoost ROC

The heatmap of XGBoots shows a little improvement over the SVM. It still has great classification power on non-toxic comments, and on toxic precision, it achieves higher recall than SVM but lower precision. This suggests that the tree is trying to predict more comments to be toxic, and it captures more actual toxic comments but also mislabel more non-toxic comments as toxic. The overall f1-score is only a little increased. The ROC curve and AUC values for two labels are relatively fine, indicating that the classifier has good classification capability.

The comparison between SVM and XGBoost lies in what aspect of the error we are more interested in. If predicting a non-toxic comment wrong is more costly than predicting a toxic comment as non-toxic, we need to use SVM which has lower false positive rate on toxic comments. Vice versa, we will use XGBoost when we want to capture as many toxic comments as we want and don't care about the non-toxic included accidentally. To sum up, the usage of two models come from the actual scenario we encounter.

3.3 LSTM

For LSTM, the classification for non-toxic comment was very good, with high scores in both Precision and Recall. It only slides a little bit on the Recall of non-toxic comments, which means it is not capturing a lot of non-toxic comments. However, the LSTM classifier did a great job and shown great classification power, with ideal ROC curve and AUC being 0.97. Some plots are shown in Figure 4:

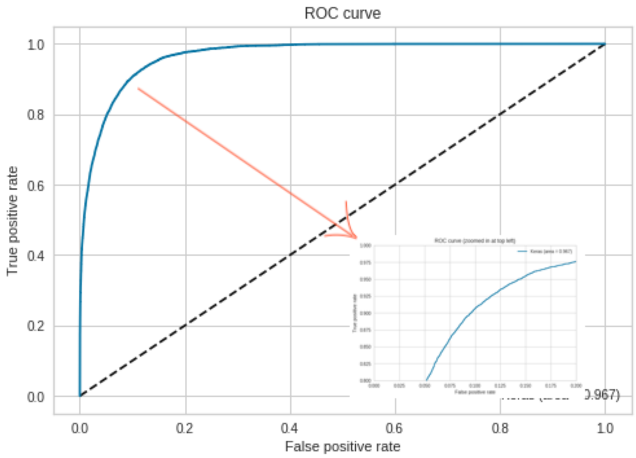


Figure 4. LSTM ROC

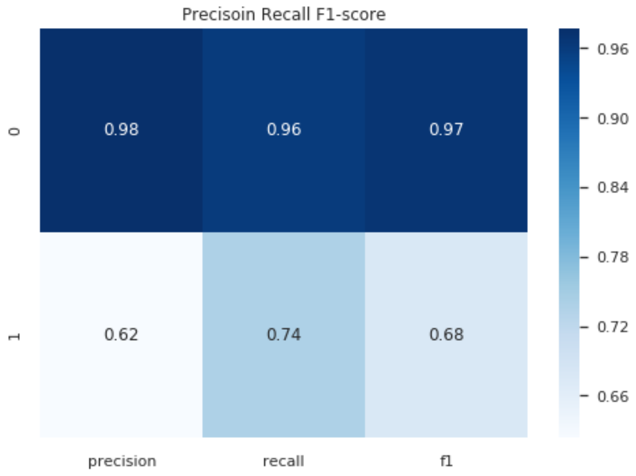


Figure 5. LSTM Heatap

4 Discussions and Future Work

4.1 Discussions

In this project, after comparing different models like LinearSVM, XGBoost, LSTM and so on, we find that the result of LSTM showed the best classification power. The AUC(Area Under Curve) for LSTM model is about 0.967 and all its Precision and Recall values are above 50%. In reality, this kind of method should be very useful and meaningful to to help maintain a stable Internet environment. It can help to detect toxicity comments as soon as possible with a relatively high accuracy, so that such aggressive and disrespectful ones can be eliminated quickly. If we have to use traditional machine learning model for sake of time and computing power such as SVM and XGBoost, the choice will be made on based on what type of error we want to avoid.

4.2 Future Work

Although our model has showed a strong toxicity detection power, there are some works need to be done in the future.

1) we use LinearSVM, XGBoost and LSTM model in this project and we can test more different models to find whether they can improve the accuracy or not.

2) We can try different embedding method during the text pre-processing. In this project, we use n-grams combined with TF - IDF method. In the future, we can search for better embedding method that can fit for our project.

4.3 Why Neural Network?

The reason why neural network models outperform traditional models is based on the nature structure of the data.

Neural Networks versus Decision Trees

A decision tree is a greedy algorithm that can only make progress when each base feature carries some reasonable amount of information. If they do, it can process one feature at a time. However, neural network is updating parameters that affect all parts of the object at all scales. In other problem, a decision tree only looks at one n-gram at a time, and it can never really improve from where it was and never really progresses discover a good separate boundary. Since the base unit of our problem, which is the n-gram, is not individually informative, using decision trees will likely unable to learn the actual meaning of a comment by only looking at one n-gram at a time.

Neural Networks versus Linear SVM

The main advantage of neural networks is that it can stack multiple layers on each other, which will further improve the prediction power. Moreover, linear SVM only draws linear boundary, which might not be the actual case to decide whether a comment is toxic or not. However, using other kernels is very computationally expensive and we don't have sufficient computing power.

5 Acknowledgments

Acknowledgments

The authors would like to thank Prof. Qiaozhu Mei, Wei Ai and all GSI for their supervision and support.

References

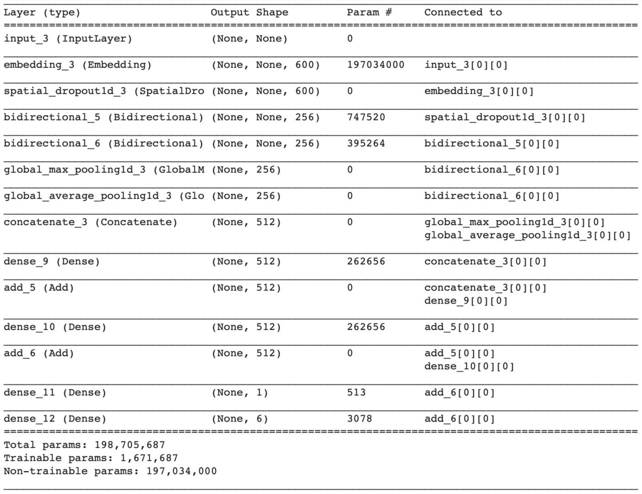
[1] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in Pre-Training Distributed Word Representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162

[3] Thousandvoices. 2019. Simple LSTM. https://www.kaggle.com/thousandvoices/simple-lstm

A LSTM Structure

A.1 LSTM



```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            (None, None)         0
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, None, 600)    197034000   input_3[0][0]
__________________________________________________________________________________________________
spatial_dropout1d_3 (SpatialDro (None, None, 600)    0           embedding_3[0][0]
__________________________________________________________________________________________________
bidirectional_5 (Bidirectional) (None, None, 256)    747520      spatial_dropout1d_3[0][0]
__________________________________________________________________________________________________
bidirectional_6 (Bidirectional) (None, None, 256)    395264      bidirectional_5[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_3 (GlobalM (None, 256)          0           bidirectional_6[0][0]
__________________________________________________________________________________________________
global_average_pooling1d_3 (Glo (None, 256)          0           bidirectional_6[0][0]
__________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 512)          0           global_max_pooling1d_3[0][0]
                                                                 global_average_pooling1d_3[0][0]
__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 512)          262656      concatenate_3[0][0]
__________________________________________________________________________________________________
add_5 (Add)                     (None, 512)          0           concatenate_3[0][0]
                                                                 dense_9[0][0]
__________________________________________________________________________________________________
dense_10 (Dense)                (None, 512)          262656      add_5[0][0]
__________________________________________________________________________________________________
add_6 (Add)                     (None, 512)          0           add_5[0][0]
                                                                 dense_10[0][0]
__________________________________________________________________________________________________
dense_11 (Dense)                (None, 1)            513         add_6[0][0]
__________________________________________________________________________________________________
dense_12 (Dense)                (None, 6)            3078        add_6[0][0]
==================================================================================================
Total params: 198,705,687
Trainable params: 1,671,687
Non-trainable params: 197,034,000
__________________________________________________________________________________________________
```

Figure 6. LSTM model structure [3]

A.2 Word Cloud

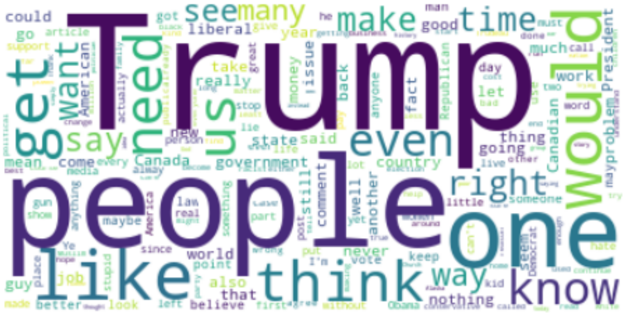


Figure 7. Representative Words of Toxic Comments