Toxic Comment Detection and Classification

Prinslou Tare prinslou@stanford.edu
Department of Computer Science
Stanford University

Abstract

This project aims to build models that detect and classify comments as toxic. In this project, I made use of various models on the data such as Logistic Regression, XGBboost, SVM and a bidirectional LSTM(Long Short Term Memory). The SVM, XGBboost and Logistic regression implementations achieved very similar levels of accuracy whereas the LSTM implementation achieved the best accuracy out of all of the models.

1 Introduction

In today's current society, there is a big problem when it comes to online toxicity. This toxicity tends to negatively impact how a lot of people tend to engage in conversation and deters some from engaging in online conversation entirely. As a result online platforms tend to struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments. The goal of this project is to detect and identify toxic comments using both machine learning models and deep learning models. This will help in deterring people from posting toxic comments and thus help to facilitate more courteous and civil discussion on online forums.

The dataset I used in this project is the Jigsaw/Conversation AI dataset provided for the Kaggle Toxic Comment Classification Challenge. The dataset provided contains a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The types of toxicity classes in the dataset are:toxic, severe_toxic, obscene, threat, insult, identity_hate.

2 Related Work

Over the past couple of years, immense work has gone into research involving toxicity. This has largely been done in the context and sphere of social media. In a broad sense, toxic comment classification falls under the well researched umbrella of sentiment analysis. Early work in this field began at Lehigh University in 2009 where a group of researchers combined TF-IDF with sentiment features. Over time, more and more companies have taken up this task with the most prominent among them being Google who established the company Jigsaw, a company dedicated to stopping online harassment.

In terms of model specificity there have been a couple of different approaches that have been looked at for the purpose of harassment detection. In 2017, Yu and Wang[1] proposed a refinement model based on adjusting the vector representations of words such that they can both be closer to sentimentally similar words and further away from sentimentally dissimilar words. This refinement model could be applied to any pre-trained word vectors (e.g., Word2vec and GloVe). Experimental results from their work showed that the proposed method can improve conventional word embeddings and outperform previously proposed sentiment embeddings for both binary and fine-grained classification on Stanford Sentiment Treebank (SST).

There has also been research conducted into how deep learning models perform in toxic comment detection. Specifically, Van Aken et al[2] compared different deep learning and shallow approaches on a new, large comment dataset and proposed an ensemble that outperforms all individual models.They concluded that the best performing models out of the current existing models were CNN's and LSTM's.

In this paper, I try to extend and emulate previous approaches to the problem by using various machine learning models such as Support Vector Machines, Logistic Regression, XGBboost and a Bidirectional LSTM.

3 Dataset and Features

As mentioned earlier in the paper, the datastet I used in this project was the Jigsaw/Conversation AI dataset provided for the Kaggle Toxic Comment Classification Challenge. The dataset's only feature is the online comments and, as mentioned above, these comments are classified as one or more of the six classes. These classes are are:toxic, severe_toxic, obscene, threat, insult, identity_hate.

3.1 Data Preprocessing

Data processing was done in python and I used the same balanced dataset that was provided by Kaggle. I initially tried random data sampling with the objective of even class distribution but that didn't seem to have an effect on the models. As such, I reverted back to using the dataset in it's original state.

Word Embedding
In the case of word embedding, I used the GloVe 60 dimensional word vectors that have been pre-trained on Wikipedia. These embeddings converted the words in the dataset into their respective vector representations.

LSTM
As for the LSTM,as part of the preprocessing, I implemented comment padding so as to ensure that the each comment in the dataset had equal word length. After different word lengths, I settled for 250 as the maximum word length in this approach. This decision was largely based on calculating the mean and standard deviation of the length of the comments on the dataset.

Splitting Dataset
After doing processing on the data frame and getting rid of all the null values,I split the dataset on a 75/25 basis for training and testing respectively.

4 Methods, Experiments and Results

Once done with pre-preprocessing the data and doing a randomized train-test split on the data, I utilized a number of different machine learning models for the purpose of classification. I used accuracy(label correctness across the dataset), as the sole evaluation metric in the project and I applied this metric to all the models. As for the models in particular, I made use four different models that I'll discuss below.

4.1 Logistic Regression Model (Baseline)

The first approach I took was to implement the Logistic regression algorithm. This was done so as to provide sort of a baseline that I could work with. In implementing logistic regression, I fed as input the normalized GloVe word embeddings for all words into the sklearn default logistic regression package while using a one vs all classifier since it's a multi class classification problem. Logistic regression makes use of the formula

$$h_\theta(x) = g\left(\theta^T x\right) = \frac{1}{1 + e^{-\theta^T x}}$$

where x represents that features. Implementing this algorithm, I was able to obtain a test accuracy of 0.89 on the test data.

4.2 SVM

The next algorithm I implemented was an SVM. In a similar manner to the logistic regression algorithm, I fed as input the normalized GloVe word embeddings for all words into the sklearn default svm package while using a one vs all classifier. The default sklearn package utilises the standard rbf kernel given by the formula

$$K\left(\mathbf{x}, \mathbf{x}^{\prime}\right)=\exp \left(-\frac{\left\|\mathbf{x}-\mathbf{x}^{\prime}\right\|^{2}}{2 \sigma^{2}}\right)$$

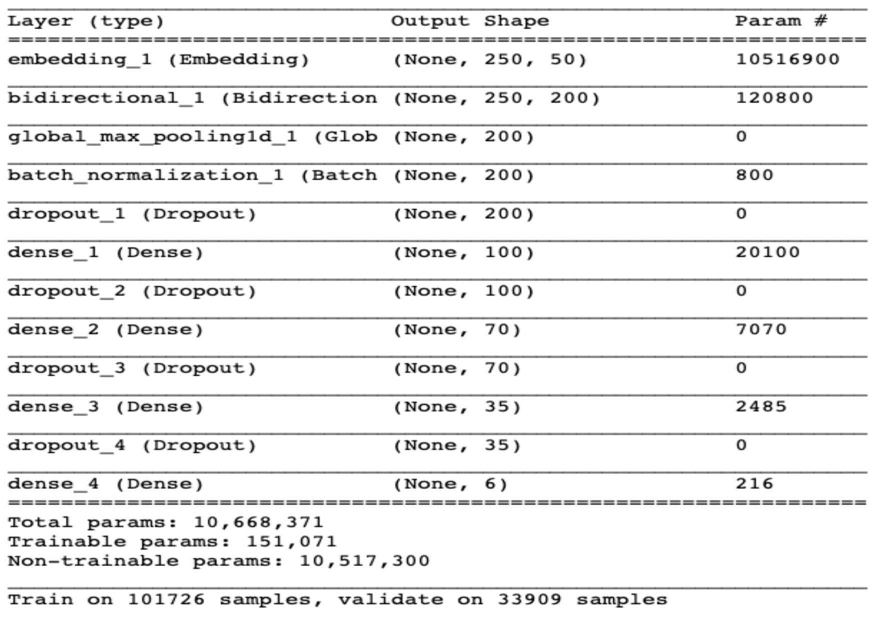
. Using this implementation, I got a test accuracy of 0.88.

4.3 XGB Boost

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. In implementing this model, the approach was similar to the SVM and logisitc regression models.I fed as input the normalized GloVe word embeddings for all words into the sklearn default svm package while using a one vs all classifier.I implemented this approach since 1 felt that the model's laser focus on computational speed and model performance would be helpful in increasing predictive accuracy. Moreover,the model is very useful for predictive modeling in regression and classification, especially when dealing with structured or tabular datasets. When implementing this model, I used 100 estimators, a learning rate of 0.01 and maximum depth of 4. I obtained an accuracy of 0.9 with this model.

4.4 Bidirectional Long Short Term Memory(Bi-LSTM)

In implementing the Bidirectional LSTM model, I first began by tokenizing the comments in the dataset. This included both the training and testing data. I then padded the train and test data after each sequence with a maximum sequence length of 250. I then used the pretrained GloVe vectors as I done on other model implementations to create an embedding matrix which I used as the model weights. In implementing the actual model, I specified the maximum input length to the Embedding layer to be 250(similar to the sequence length) and implemented 4 fully connected layers. Three of the four dense layers each had a dropout of 0.15 and used ReLu as the activation function and a final dense layer used sigmoid activation for the purpose of classification. Finally, I set the model to optimize the loss function using "Adam" optimizer and defined the loss function to be "binary_crossentropy". I also used a learning rate of 0.01. I originally run the model for 4 epochs but reverted to using 2. I did this since I noticed that I was overfitting the training data after plotting the training vs test loss. The plot also helped me tune my hyperparamters including the learning rate and eventually, I ended up with a model architecture with the parameters below.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 250, 50)           10516900
_________________________________________________________________
bidirectional_1 (Bidirection (None, 250, 200)          120800
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 200)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 200)               800
_________________________________________________________________
dropout_1 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               20100
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 70)                7070
_________________________________________________________________
dropout_3 (Dropout)          (None, 70)                0
_________________________________________________________________
dense_3 (Dense)              (None, 35)                2485
_________________________________________________________________
dropout_4 (Dropout)          (None, 35)                0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 216
=================================================================
Total params: 10,668,371
Trainable params: 151,071
Non-trainable params: 10,517,300
_________________________________________________________________
Train on 101726 samples, validate on 33909 samples
```

After fine tuning the hyperparamters, I managed to get an accuracy of 0.97 which was substantially better than all the other models.

4.5 Summary

To table below represents the summary of the performance of all my models.

Model	Logistic Regression	SVM	XGB Boost	LSTM
Accuracy	0.89	0.88	0.90	0.97

5 Conclusion and Future Work

The best model with regard to multi-label classification in the dataset was definitely the LSTM. It outperformed all the other models. I did sort of expect the SVM and the logistic regression implementation to have similar levels of accuracy, however,I expected XGB boost to perform better and I was slightly surprised that it had a very similar level of accuracy to both the SVM and Logistic Regression Models.

In the future, one thing I would be very interested to look into is whether character embedding achieves better performance accuracy as compared to word embedding. I would also like to look into implementing a Convolutional Neural Network(CNN) model and test its performance against the LSTM implementation. Another thing that I didn't get to do was to implement different LSTM models. If given access to the adequate computational memory, I'd like to experiment with different LSTM models. The reason for this is that I chose to implement a bidirectional LSTM from the get go and spent the rest of the time fine tuning the model hyperparamers. As such, I wasn't able to try out different models.

#Github link - https://github.com/Prinslou/Toxic-Comment-Project

References

[1] L.-C. Yu, J. Wang, K. R. Lai, and X. Zhang, "Refining Word Embeddings for Sentiment Analysis," Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, 2017.

[2]Van Aken, Betty Risch, Julian Krestel, Ralf Löser, Alexander. (2018). Challenges for Toxic Comment Classification: An In-Depth Error Analysis. 10.18653/v1/W18-5105.

[3] "GloVe: Global Vectors for Word Representation". Jeffrey Pennington, Richard Socher, Christopher D. Manning. Computer Science Department, Stanford University. [Online]. Available: https://nlp.stanford.edu/pubs/glove.pdf

[4]"Linguistic Regularities in Continuous Space Word Representations" Omas Mikolov, Wen-tau Yih, Geoffrey Zweig. Microsoft Research. [Online]. Available: https://www.aclweb.org/anthology/N13-1090. [Accessed: 15-Dec-2018]

[5]Nadbor, "DS lore," Text Classification With Word2Vec - DS lore. [Online]. Available: https://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/. [Accessed: 15-Dec-2018].