F*** You Detector: Identifying Offensive and Obscene Comments

Akshay Gupta, Sai Anurag Modalavalasa, & Alex Samardzich

{akshaygu, anuragms, asamardz}@stanford.edu

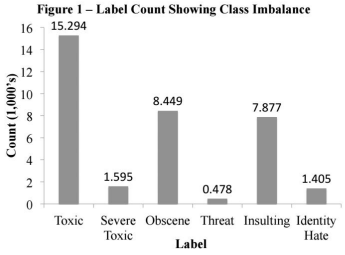
Introduction

The goal of this project is to classify a text comment as being toxic, severely toxic, obscene, threatening, insulting, and/or a form of identity hate. With a model to identify such abuses, online platforms can effectively flag comments that threaten the safety or mental health of their users, something companies like Twitter have recently announced plans to do [1]. Input data is in the form of text comments and each model outputs a length-six prediction vector with binary classifications in each category. DNN + character n-gram, CNN + character n-gram, and LSTM RNN + GloVe models were applied. After testing, it was found that LSTM RNNs proved most promising at this task with 93.0% accuracy and recall on the test set.

Data

The dataset for this project was taken from kaggle's "Toxic Comment Classification Challenge". Each text comment has a binary classification within six potential labels – Toxic, Severe Toxic, Obscene, Threat, Insulting, and Identity Hate. The dataset included approximately 160,000 examples which were divided into training and test sets of 155,500 and 2,070 examples, respectively.

Figure 1 – Label Count Showing Class Imbalance



Models

Since there are common features associated with the domains into which the text comments are classified, a multi-task learning architecture with sigmoid activation functions for each neuron in the last layer was implemented. The objective is to maximize recall with accuracy as satisfying condition. Three different models were implemented as part of this project. Each model employed Adam Optimization, Xavier initialization, and the weighted cross entropy loss function $\mathcal{L} = -W[y\log(\hat{y})] - (1-y)\log(1-\hat{y})$ was used to deal with the inherent class imbalance in the dataset. Finally, a threshold was used to convert the final probabilities vector into a binary classification vector.

Deep Neural Network (DNN) + Character n-gram

- Input Features: Character level embedding using the ord() function in python with length equal to the maximum characters in a text comment in the data set, 6000
- 6 Layers with 6,000, 500, 100, 25, 12, and 6, fully connected nodes, with ReLU activation except for the last layer

Convolutional Neural Network (CNN) + Character n-gram

- Input Features: Same as input features for DNN + Character n-gram
- 3 Convolution layers followed by a Global Max Pooling 1-D layer and 3 layers of fully connected network

LSTM Recurrent Neural Network (RNN) + GloVe

- Input Features: Word2Vec embedding using GloVe (50d), first 200 words in a comment
- 2 Layers of LSTM followed by 1 layer of fully connected network

Figure 2 – LSTM RNN + GloVe Architecture

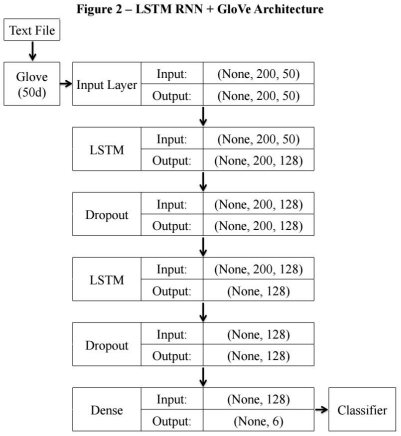
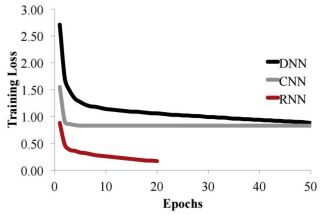


Figure 3 - Results

	DNN	CNN	RNN
Epochs Trained	50	50	20
Parameters	3,054k	2.5k	224k
Train Accuracy	85.1%	82.9%	93.4%
Train Recall	66.3%	43.5%	99.6%
Test Accuracy	84.0%	83.1%	93.0%
Test Recall	56.1%	46.7%	93.0%

Figure 4 - Training Loss for Each Model



Results and Discussion

After training, it was found that the most successful model when it came to both recall and accuracy on the training and test sets was the LSTM RNN. The results indicate that the models did not see the problem of over-fitting as test and training set accuracy was comparable. This result was expected as RNNs have proven successful in a wide range of natural language processing tasks. One feature of the RNN used was that the words in the data set which do not appear in the GloVe vectors have been mathematically represented using the unk token. Such a representation coupled with a LSTM model likely captures the contextual meaning better than character level n-gram models. It is of note that the results were extremely sensitive to the weight factor in the weighted cross entropy loss function and the threshold for converting probabilities into binary classification.

Future Steps

To improve upon the results further, future model iterations would include training a Bi-directional LSTM RNN. A dictionary of all worlds in the dataset would be trained as opposed to just using the pre-trained GloVe embeddings in order to allow the model to better detect which words and phrases are common to abuse comments. Additionally, sentiment lexicon features, count of special characters(hashtags), number of characters/words/syllables features may be used along with word2vec embedding to capture context [2].

References

[1] Kircher@4evrmalone, M. M. (2018, May 15). Twitter to Start Hiding Bad Tweets. Retrieved from https://nymag.com/selectall/2018/05/twitter-to-start-hiding-badtweets

[2] Davidson, Thomas & Warmsley, Dana & Macy, Michael & Weber, Ingmar. (2017). Automated Hate Speech Detection and the Problem of Offensive Language.