CS230

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

Akshay Gupta
06191248
akshaygu@stanford.edu

Sai Anurag Modalavalasa
06181125
anuragms@stanford.edu

Alex Samardzich
06108597
asamardz@stanford.edu

Abstract

The goal of this project is to assign the probability of a text comment being toxic, severely toxic, obscene, threatening, insulting, and/or a form of identity hate. In order to achieve this goal, three models were tested for their ability to correctly identify abuse sentences with an objective of maximizing recall with accuracy as a satisfying condition. A deep neural network (DNN) and a convolutional neural network (CNN) were trained using character level embedding as features. Additionally, a LSTM recurrent neural network (RNN) was trained using GloVe (50d) word2vec embeddings as input features. After testing, it was observed that the RNN was most successful in classifying these sentences. Test accuracy and recall were at 93.0% while training accuracy and recall were at 93.4% and 99.6%, respectively.

1 Introduction

Human interactions in online spaces are increasing, be it through social networks such as Facebook or through comment sections of articles and posts. Many individuals feel emboldened to use extreme speech and threatening language when communicating with others in online spaces because they are not face-to-face with their victim. While free speech is protected under the First Amendment, threatening the safety of others is not. Although some would argue that users could simply ignore online comments, this is clearly not the first instinct of the youngest generation. News stories describing teenagers being cyber bullied to the point of taking their own lives have become all too common in the age of the Internet. Instead of bombarding the Facebook pages and YouTube comment sections of children with insulting, racist, and threatening remarks, online platforms should have the capability to quickly and accurately identify such remarks and flag them for review.

Improving the online experience of users by flagging such remarks is not only a pertinent task in today's world; it is also an extremely interesting deep-learning challenge. While the algorithm should detect hateful, offensive remarks, it should also have the capability to understand the context: it is not as simple as looking for trigger words. For example, while the phrase "I will kill you" would be flagged as threatening, "the SF Giants killed it today" should be flagged as neutral. Additionally, many innocuous online comments are made sarcastically or with certain tones that the algorithm will have to learn to detect.

The input for this task is a text comment that has been taken from Wikipedia's comment section. Using this input, each model outputs a length-six vector containing a binary classification for toxic, severely toxic, obscene, threatening, insulting, and identity hate comments, with a 1 meaning the comment fell into that category. For example, an output label of [1 0 0 1 0 0] meant that the comment was classified as toxic and threatening.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

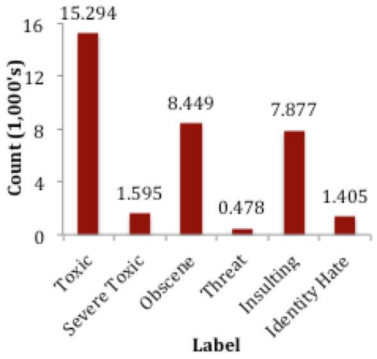
2 Related work

The goal of flagging hateful posts has already been adopted by large online platforms such as Twitter, who is testing new algorithms to remove some of the most toxic content on its site [1]. Additionally, natural language processing tasks such as this one have been hot topics in the deep learning community as of late. Features such as Character Level n-grams, word embeddings, presence of specific negative words (using a hate related terms word list) can be used and recurrent neural network may be applied for the classification task [2]. Higher F1 scores were observed in the task of classifying a tweet as sexist, racist or neither when LSTM/CNN + Random embedding/GloVe embedding + GBDT classifier models were used [3]. Application of CNNs with word2vec architecture resulted in better F1 scores as compared to CNN with character n-grams/random vectors/word2vec + character n-grams architectures [4]. SVM and LSTM architectures with sentiment polarity and word embedding features were applied on strong hate, weak hate, no hate classification tasks. F score associated with a no hate classification task and a strong hate classification task was the highest and lowest respectively among the three [5]. Using TF-IDF weighted unigram, bigram, trigram features, syntactic structure features, sentiment lexicon features, count of special characters (hashtags), number of characters/words/syllables features, various models such as logistic regression, decision trees, linear SVMs were applied for the classification task. The model was biased towards classifying less hateful as compared to human annotations [6].

3 Dataset and Features

The dataset for this project has been taken from kaggle under the "Toxic Comment Classification Challenge" [7]. The set includes approximately 160,000 manually labeled comments from Wikipedia's talk page edits ranging from a few words up to 6,000 characters. Each comment has a binary classification within six potential labels – Toxic, Severe Toxic, Obscene, Threat, Insulting, and Identity Hate. A comment may be labeled in multiple categories. Figure 1 below shows the total number of times each label had been applied in the dataset. Because examples can be labeled in multiple categories, only around 16,000 entries, or 10% of the total examples, were not labeled as all 0's as seen in Figure 1 below.

Figure 1: Data Label Count



The example "COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK" was labeled as toxic, severely toxic, obscene, and insulting. In contrast, the example "bbq be a man and lets discuss it-maybe over the phone?" was not flagged in any category. The second example highlights the difficulty of this labeling task because one individual might consider the second example insulting due to the inclusion of "be a man".

The dataset was divided into training and test sets of 155,500 and 2,070 examples, respectively. The input features to the DNN and CNN were character level embedding using the ord() function in python with length equal to the maximum characters in a text comment in the data set, 6,000. The input features to the RNN were word2vec embeddings of the first 200 words in the text comment using the GloVe (50d) database. Before the comments were converted to word vectors, they were

made lowercase and stripped of special characters. If a word did not appear in the GloVe database, it was assigned the unk token.

4 Methods

4.1 Structuring the deep learning project

Since the data has a binary classification for six variables and because there are common features associated with the six variables, a multi task learning architecture with sigmoid activation functions was implemented for each neuron in the last layer of the deep learning models developed here. The objective of the neural network is to have high recall value associated with the predictions (the ratio of the number of times the model predicted abuse sentences correctly to the total number of abuse sentences) with an accuracy value associated with the prediction being a satisfying condition. By analyzing bias and variance, efforts will be put in to gain insights into any under fitting/over fitting problems that arise.

4.2 Model Architectures

A. Deep Neural Network + character n-gram

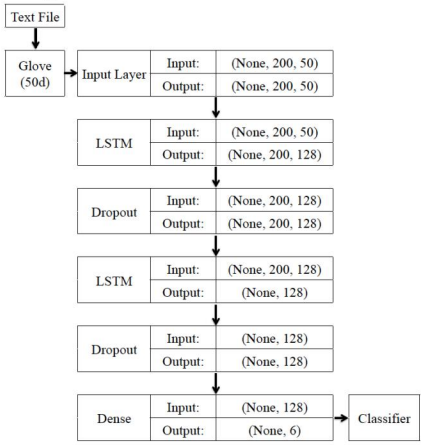
Each comment was converted into a 1x6000 vector with each entry in the vector containing a numerical representation of a character in the comment. The benefit of this processing was that unique characters could be captured, such as a**hole. The input vector was then fed into a 6-layer model with 6000, 500, 100, 25, 12, and 6 nodes in each layer, respectively. Layers 1-5 used ReLU activation while layer 6 used a sigmoid activation. The output of layer 6 was a 1x6 vector containing the classification predictions for each comment. A threshold was used to convert the prediction probability vector to a binary classification. The threshold was a hyperparameter for this model and a value of 0.7 was chosen after tuning. Each layer's weights were initialized using Xavier initialization and the bias vectors were set to zero. To deal with the inherent class imbalance in the dataset, the weighted cross entropy loss function $L = -W[y\log(\hat{y})] - (1-y)\log(1-\hat{y})$ was used. Weighted cross-entropy loss was summed over the six category types as the cost function, which was then minimized using Adam optimization. The weight in the loss function was tuned to 30.

B. Convolutional Neural Network + character n-gram

To reduce the number of parameters and capture patterns in abuse comments, a convolutional neural network was employed. The input features to the CNN were the same as the DNN. The model architecture included 3 convolution layers (filter size 3, 4, and 5) with same padding and ReLU activations followed by a global max pooling 1-D layer and 3 layers of fully connected network (size 50, 12, and 6). Xavier initialization was used and weighted cross entropy loss (weight tuned to 15) was minimized using Adam optimization. A threshold of 0.5 was used.

C. LSTM Recurrent Neural Network + GloVe

Figure 2: LSTM RNN Architecture



The first 200 words in each comment were converted into word vectors using GloVe (50d). If the word was not featured in the database, it was given the unk token. The model architecture included two layers of LSTM (128 dimension units, the return sequence was True for the first layer and the return sequence was False for the second layer) followed by two fully connected layers of size 12 and 6 respectively (ReLU activation for the first layer and sigmoid activation for the second layer) with Xavier Initialization as seen above in Figure 2. Weighted cross entropy loss (weight tuned to 20) was used with Adam optimization for minimization. A threshold of 0.1 was used.

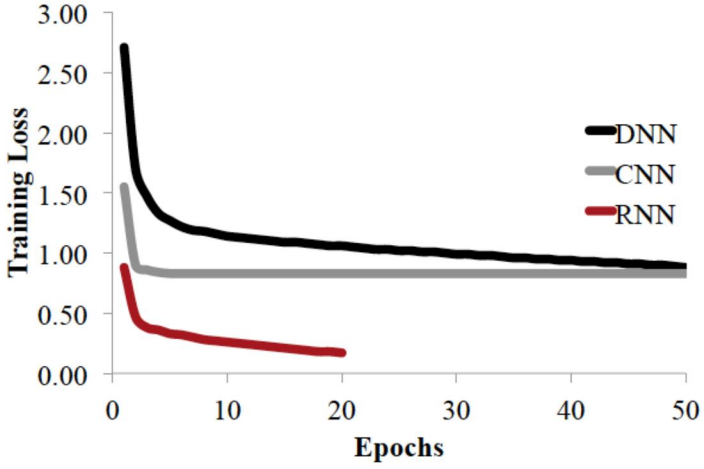
5 Experiments/Results/Discussion

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 as seen below in Figure 3. The DNN performed better than the CNN in terms of recall, which was likely due to the fact that the DNN had significantly more parameters allowing it to better capture context. The high recall and lower loss of the RNN compared to the other two models was expected, as RNNs have proven successful in a wide range of natural language processing tasks. Even though the RNN was only trained for 20 epochs where as the CNN and DNN were trained for 50, the RNN had significantly lower loss to begin with and at the end of training compared to the other two models as seen in Figure 4 below. The RNN was trained for fewer epochs due to the computational expense; the input to the RNN was a 200x50 dimensional matrix. Part of the RNN's success is likely due to the fact that the GloVe vector embeddings captured the contextual meaning of the comments better than character level encoding since GloVe has been pretrained on large datasets. Additionally, the results indicate that the models did not see the problem of over-fitting as test and training set accuracy was comparable.

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 Error



In early models, it was observed that the accuracy was extremely high and the recall was low. By performing error analysis, it was found that setting the threshold for converting the output probability vectors to binary classifications as a hyperparameter can remedy this issue. Increasing the weight to a high enough value may eliminate the need for a threshold hyperparameter, but the process would require significantly longer training and is computationally expensive. The learning rate and layer dimensions used in each model were found iteratively by testing various sizes. Each model was initially tested on a smaller dataset using a variety of hyperparameter combinations to identify the right blend for full training. During model training, a mini-batch size of 32 or 64 comments was used in order to perform a large number of gradient descent steps in each epoch.

While the RNN had high recall on the test set, one area of difficulty for the model was classifying comments as severely toxic. The obscene comments had common qualities such as profanities, the threatening comments used similar verbs, and the identity hate comments used words associated with race, gender, and sexual orientation. However, the severely toxic sentences did not posses these common features and the labeling seemed relatively arbitrary. Unsurprisingly, this led to many of the missed recall examples in the test set for each model being those that should have been classified as severely toxic. Eliminating this category entirely may lead to better results as the distinction between severely toxic and toxic comments is not a great enough for web platforms to treat them differently in their attempts to protect users from abuse.

6 Conclusion/Future Work

After training a DNN, CNN, and RNN to identify obscene and offensive comments, it was found that the LSTM RNN was most successful at this task with test recall and accuracy of 93.0%. The LSTM RNN's success was due to its ability to better capture the context of comments using the GloVe (50d) database and the sequential nature of the data. Future steps for this project include testing a bi-directional LSTM and fine-tuning the word embeddings instead of the static GloVe database. Acquiring data from other websites would also be beneficial as comments on platforms like Twitter are fundamentally different than those on Wikipedia as tweets must be below a certain character limit and abbreviations/slang are more common. Additionally, with more computational power, it would be interesting to compare the results of the DNN and CNN using word2vec embeddings and the RNN on character level embeddings. Due to the long training times (5-8 hours per model), this was not possible during the first iteration of this project.

7 Contributions

Each group member was present for the majority of meetings and contributed equally to the project. All members took part in the coding of each model and preparation of final report and poster.

References

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

[2] Schmidt, Anna & Wiegand, Michael. (2017). A Survey on Hate Speech Detection using Natural Language Processing. 1-10. 10.18653/v1/W17-1101.

[3] Badjatiya, Pinkesh & Gupta, Shashank & Gupta, Manish & Varma, Vasudeva. (2017). Deep Learning for Hate Speech Detection in Tweets. 10.1145/3041021.3054223.

[4] Gambäck, B., & Sikdar, U.K. (2017). Using Convolutional Neural Networks to Classify Hate-Speech.

[5] Del Vigna, Fabio & Cimino, Andrea & Dell'Orletta, Felice & Petrocchi, Marinella & Tesconi, Maurizio. (2017). Hate me, hate me not: Hate speech detection on Facebook.

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

[7] Kaggle.com Toxic Comment Clasification Challenge https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge