Toxic Comments

Chris Barcelon
January 2018

I. Definition

Project Overview

Comments that are toxic, such as threats, obscenity, insults, and identity-based hate, are everywhere on the Internet. The large number of toxic comments have prevented people from participating in discussions they are interested in and they have led to communities either limiting or restricting comments altogether. A model that is capable of detecting these types of comments could hopefully help online discussions become more productive and respectful. In the paper Ex Machina: Personal Attacks Seen at Scale the authors discuss solving a similar problem initially using logistic regression and multi-layer perceptrons and planned to in the future use LSTM models. This project is based off the kaggle competition **Toxic Comment Classification Challenge**

Problem Statement

The problem is to build a model that can detect and classify different types of toxicity in comments. Given a list of comments the model will predict the probability that each comment is toxic. It will predict 6 different probabilities one for each 6 different types of toxicity.

Metrics

The model will be scored with a column-wise **log loss** function. The score will the be the average of the log loss of each predicted column, where each column is a different type of toxicity. Log Loss will be used to score the model because it quantifies the accuracy of a classifier by penalising false classifications.

II. Analysis

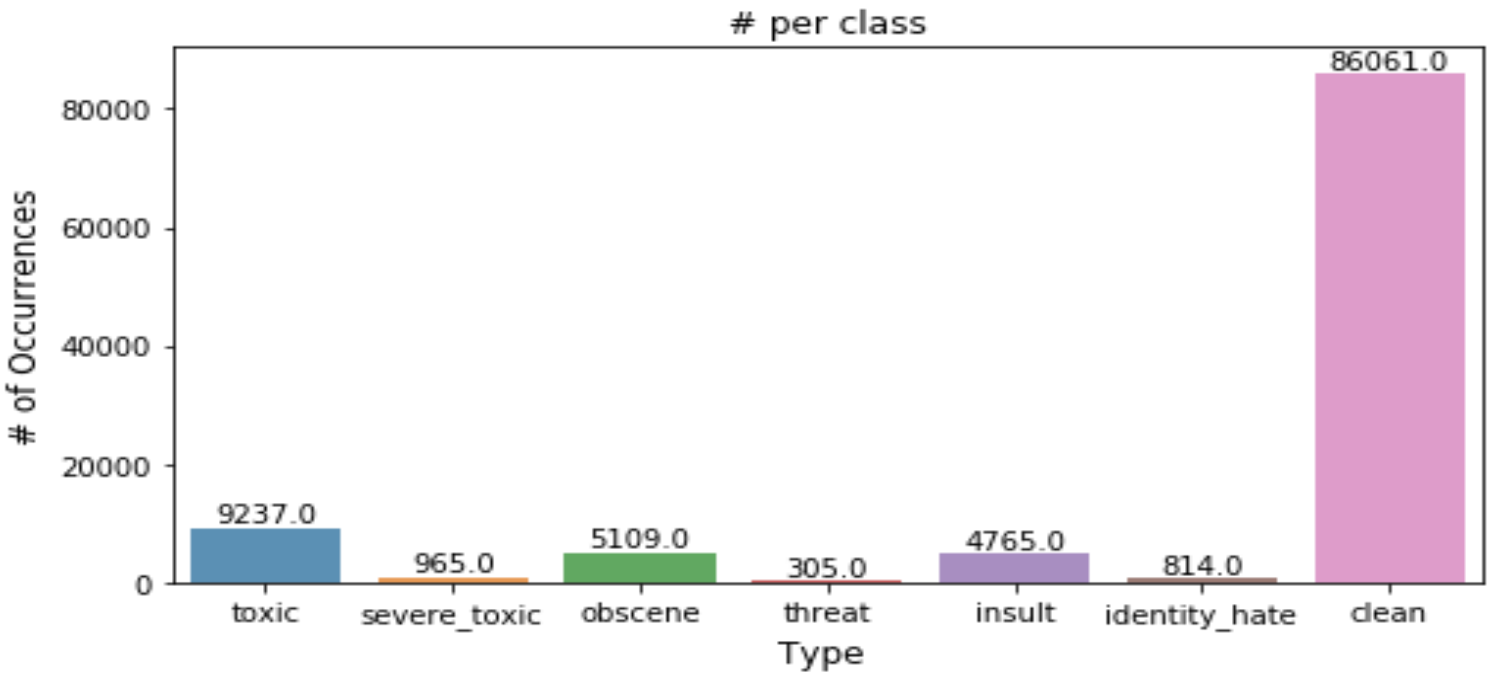
Data Exploration

The dataset is a csv file, train.csv, with a large number of Wikipedia comments that have been labeled by humans for toxicity. Each comment is labeled for 6 different types of toxicity: toxic, severe_toxic, obscene, threat, insult, identity_hate. train.csv contains 95850 comments each comment has an id number and a binary classification for each of the types of toxicity. The first 10 entries in the file can be seen below.

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	22256635	Nonsense? kiss off, geek. what I said is true...	1	0	0	0	0	0
1	27450690	"\n\n Please do not vandalize pages, as you di...	0	0	0	0	0	0
2	54037174	"\n\n ""Points of interest"" \n\nI removed the...	0	0	0	0	0	0
3	77493077	Asking some his nationality is a Racial offenc...	0	0	0	0	0	0
4	79357270	The reader here is not going by my say so for ...	0	0	0	0	0	0
5	82428052	Fried chickens \n\nIs dat sum fried chickens?	0	0	0	0	0	0
6	87311443	Why can you put English for example on some pl...	0	0	0	0	0	0
7	114749757	Guy Fawkes \n\nim a resident in bridgwater and...	0	0	0	0	0	0
8	138560519	as far as nicknames go this article is embarra...	0	0	0	0	0	0
9	139353149	Woodland Meadows\nGood to hear that you correc...	0	0	0	0	0	0

Of the 95850 comments 9,789 are labeled for at least 1 type of toxicity, meaning the other 86061 comments are not toxic. The number of each comments labeled for each type of toxicity are as follows.

- toxic - 9237
- severe_toxic: 965
- obscene: 5109
- threat: 305
- insult: 4765
- identity_hate: 814



The shortest comment in the dataset is only 1 word and the longest comment is 1403 word. the average length for a comment is 67 words.

Algorithms and Techniques

NBSVM

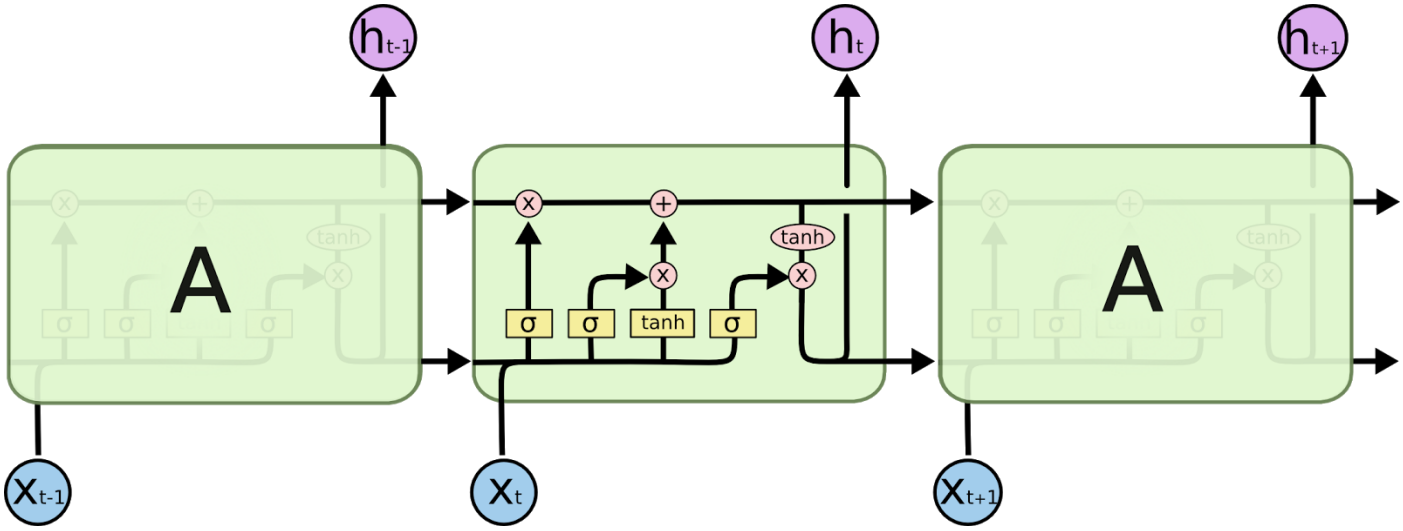
NBSVM(Naive Bayes - Support Vector Machine) is a technique that is often used for text categorization and sentiment analysis. In the paper **Baselines and Bigrams: Simple, Good Sentiment and Topic Classification** authors Sida Wang and Christopher D. Manning discuss the use of NBSVM in sentiment classification and topical text classification. A NBSVM is a model which uses a linear approach on top of naive bayes features. This model will use a bag of words approach where each unique word is a feature in the Naive Bayes formula. This will disregard word order and just look at whether a word appears in each comment and then calculate the probability that a comment is toxic or not based on words that are in the comment. The Naive Bayes equation used to calculate the probability is a follows: the **log-count ratio** r for each word f

$$r = \log \frac{\text{ratio of feature } f \text{ in non-toxic comments}}{\text{ratio of feature } f \text{ in toxic comments}}$$

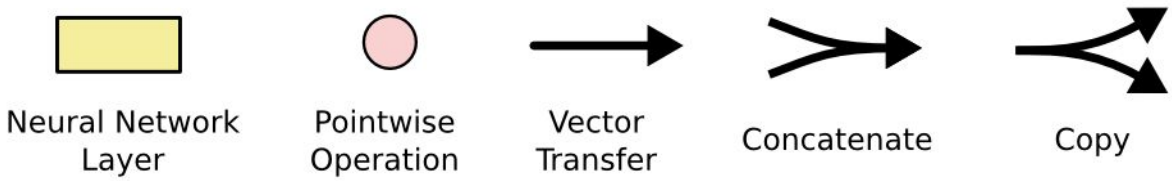
where the ratio of feature f in non-toxic comments is the number of times a non-toxic comment has a feature divided by the number of non-toxic comments. This approach will only give the probability for one type of toxicity at a time so it will be run 6 times one time for each type of toxicity to give a probability for each type of toxicity for each comment.

LSTM

Long Short Term Memory networks (LSTM) are a type of RNN (Recurrent Neural Network). A RNN is a essentially a neural network on a loop, it can be thought of as multiple copies of the same network where each copy of the network passes some information to the next copy of the network. This allows the network to preserve data, this preserved data can then be used to help give context to a future data input. A LSTM is a RNN that is especially good at preserving information over long periods of time. Christopher Olah used the following diagram to help explain how LSTMs work.



The repeating module in an LSTM contains four interacting layers



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

To gain a deeper understanding of LSTM networks I recommend reading the full post by Christopher Olah **Understanding LSTM Networks**

While the NBSVM did not care about the word order and used a bag of words for the text, the LSTM network does care about the word order so it will require the text to be vectorized. Also the LSTM network will be able to calculate the probabilities for each type of toxicity simultaneously.

Ensemble

The ensemble is a combination of the LSTM and NBSVM models. It will be the average of the two models. The idea is that if one of the models gets a prediction wrong the other one will hopefully get it right. So by combining the two models a more robust model will be created.

Benchmark

The benchmark model used is a basic bidirectional LSTM network provided by kaggle user CVxTz. The kernel used can be found here: https://www.kaggle.com/CVxTz/keras-bidirectional-lstm-baseline-lb-0-051 A summary of the model used is provided.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 100)               0
_________________________________________________________________
embedding_2 (Embedding)      (None, 100, 128)          2560000
_________________________________________________________________
bidirectional_2 (Bidirection (None, 100, 100)          71600
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 100)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_4 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 306
=================================================================
Total params: 2,636,956
Trainable params: 2,636,956
Non-trainable params: 0
_________________________________
```

When this model was evaluated after 1 epoch, using the predefined log loss function it produced a score of 0.0749.

III. Methodology

Data Preprocessing

Before any other preprocessing the data was split into 2 sets a training set and a testing set, the testing set is a random 20% of the initial dataset, the training set is the remaining 80% of the data. The data was split using the sklearn.cross_validation.train_test_split function.

To prepare the data for the LSTM model, the text of the comments needed to be converted to sequences to be trained on. The first thing done to the text was to search through the comments and replace any empty comments with filler text in this case "cbarcelon" The text was then converted to sequences using the keras.preprocessing.text.Tokenizer and then to make the training more efficient

the sequences were set to a uniform length using the pad_sequences function. The initial length chosen is 100, 100 was chosen because it is above the average length of 67 yet not too long. It is believed that truncating long comments will not adversely affect the accuracy of the model because most comments will reveal their toxicity within the first 100 words. To verify this hypothesis different sequence lengths will be tested.

To prepare the data for the NBSVM model a bag of words was created. Using ngrams the text was tokenized and split into words and then was transformed into a matrix of word values. Ngrams are defined as a continuous sequence of items in a sequence. A sparse matrix was used because most comments only use a small subset of the the total unique words.

Implementation

The two algorithms used were implemented separately and then their results were averaged together.

NBSVM

The implementation of the NBSVM model was carried out in the jupyter notebook NBSVM_Toxic_Comments. A summary of the notebook is as follows:

- Import the needed libraries and read in the train.csv file using pandas
- Split the data set into a train and a test set using train_test_split.
- Separate the comment text from their ratings
- Transform the comment text into a matrix of word values
    - TfidfVectorizer(ngram_range=(1,2), tokenizer=tokenize, min_df=11, max_df=0.8, strip_accents='unicode', use_idf=1, smooth_idf=1, sublinear_tf=1)
- Define the NBSVMClassifier function
    - The basic naive bayes feature equation
    - def pr(y_i, y):
    - p = x[y==y_i].sum(0)
    - return (p+1) / ((y==y_i).sum()+1)
- Fit the model to the to training set for each type of toxicity
    - model = NbSvmClassifier(C=1, dual=True, n_jobs=-1)
- Predict the ratings based off of the fitted model for each type of toxicity.
- Check the accuracy of the predictions.

LSTM

The implementation of the LSTM model was carried out in the jupyter notebook Toxic_Comments_Sequential_LSTM. Here is a summary of that notebook.

- Import the needed libraries and read in the train.csv file using pandas
- Split the data into train and test sets using train_test_split.
- Separate the comment text from the ratings.
- Tokenize the text data into sequences that can be used in the LSTM model.

    - The sequences are all set to a length of 100, comments shorter are extended using 0's and longer comments are truncated to a length of 100.
- Import a dictionary of word vectors from GloVe to pre weight the words from the comments.
    - The word vectors used was trained off of tweets and can be downloaded here https://nlp.stanford.edu/projects/glove/
- Create a matrix from the downloaded GloVe file using only the words in our dataset.
- Define the LSTM model, here is a summary of the model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 100)          13367600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 100, 400)          481600
_________________________________________________________________
average_pooling1d_1 (Average (None, 50, 400)           0
_________________________________________________________________
bidirectional_2 (Bidirection (None, 50, 400)           961600
_________________________________________________________________
average_pooling1d_2 (Average (None, 25, 400)           0
_________________________________________________________________
bidirectional_3 (Bidirection (None, 25, 400)           961600
_________________________________________________________________
flatten_1 (Flatten)          (None, 10000)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 10000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               5000500
_________________________________________________________________
dropout_2 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 3006
=================================================================
Total params: 20,775,906
Trainable params: 20,775,906
Non-trainable params: 0
_________________________________________________________________
```

- The LSTM model is 3 stacked bidirectional layers and two fully connected dense layers with dropout.
- Compile the model
    - lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
- Fit the model. -The model was trained over 4 epochs; more epochs just led to overfitting.
- Predict the ratings using the now trained LSTM model
- Test the results

Ensemble

The Ensemble model was produced in the jupyter notebook Ensemble. The ensemble was really just the average of the two other models predictions. A copy of the two models were created and then their predictions were averaged together to create a new prediction.

Refinement

NBSVM

To refine the NBSVM model a brute force method was applied to find the optimal parameters. The first parameters tuned were in the TfidfVectorizer: the ngram_range, min_df, and max_df were all optimized. The ngram_range sets the size of the ngrams used in the bag of words, the min_df is the minimum number of comments that a ngram must appear in order to be used in the model, the max_df is the maximum percent of comments that a ngram can appear in before it is no longer used in the model. This was done by running the model through nested for loops with 1 parameter being changed each time. The ngram_range was tested with the value of (1,2) through (1,10), the min_dif with values of 1 through 20, and the max_df with values of .1 through .9. The min_df and max_df values decide which words to not use in the model either because the words appear in too many comments therefore providing no insight or appearing in too few comments and therefore could not used in generalization. The final parameter refined was the C value in the NBSVMClassifier. The C value determines how much importance to give to outliers in the data. C values of 1, 10, 100, 1000 were initially tested with 1 by far producing the best results. So then the values of 1,2,3,4,5,6 were tested and 1 still provided the best results.

LSTM

The LSTM was refined slowly over many iterations. This is because the first lstm model implemented was a simple LSTM model similar to one used to benchmark minus the dropout layers. Once the simple model was working as intended additional layers were added. The first thing added was a LSTM layer, this improved the accuracy of the model so third identical layer was added. This third layer also improved the accuracy of the model. After the third lstm layer was added a dense layer was added. At this point overfitting was noticeable problem so dropout layers were added.

To recap the model was currently of the form, 3 lstm layers followed by 2 dense layers and before each dense layer a dropout layer. With the model at this stage the embedding parameters were experimented with. The embedding layer has multiple parameters that were played with. The input_dim and output_dim were both increased and decreased. Eventually the input_dim was chosen to be the entire vocabulary of the training comments. Using a subset of the entire vocabulary allowed the model to be trained much faster but better results were found when using the entire set of vocabulary. The output dim was set to 100 to match the size of the sequences that were being trained on. Next additional lstm and dense layers were added but they failed to add noticeable improvements upon the model so they were then removed. It was at this point that a pre weighted

word vectors was added. GloVe: Global Vectors for Word Representation provides pre trained word vectors for use. The Twitter 100d word vectors were used. Upon adding the weights from the GloVe the model showed significant improvement..

IV. Results

Model Evaluation and Validation

When the predictions of the NBSVM model were tested using the log loss function it produces a score of 0.051462166523797565.
The log loss score of the LSTM model was 0.0464747992263.
When the predictions of the two models are averaged together and then tested it resulted in a score of 0.0466733915127.
The benchmark model used produced a score of 0.0749.
Using the log loss score the LSTM produced the most accurate predictions of toxicity in the given comments. The ensemble produced a score very close to the LSTM, while the nbsvm model gave a significantly worse predictions than the LSTM or ensemble. But still all 3 models produced scores much better than the benchmark lstm model used.

The ensemble model was predicted to give the best results but the lstm model was shown to outperform it.

Justification

To check how well the lstm model performed from a practical standpoint examples of clean and toxic comments were looked at individually.
For example given the clean comment of

"*You claimed to have ""scavenged the UN and NGO website for a few hours"", yet I found it in minutes by searching for . You are either dishonest or not very thorough."*

The lstm model gave the following probabilities for each type of toxicity.

- toxic - 1.25816162e-03
- severe_toxic - 1.44767694e-06
- obscene - 8.03762960e-05
- threat - 2.61948135e-05
- insult - 3.63398285e-05
- identity_hate - 1.86055913e-05

The model was very accurate for this comment. Because the probabilities for all 6 types of toxicity are well below 1%, it can be concluded that the model is confident that this comment is not toxic.

Given the toxic comment of

"ameriKKKans like to get their anusses raped"

which is rated as toxic, obscene, insult, and identity_hate. The lstm model gave the following predictions

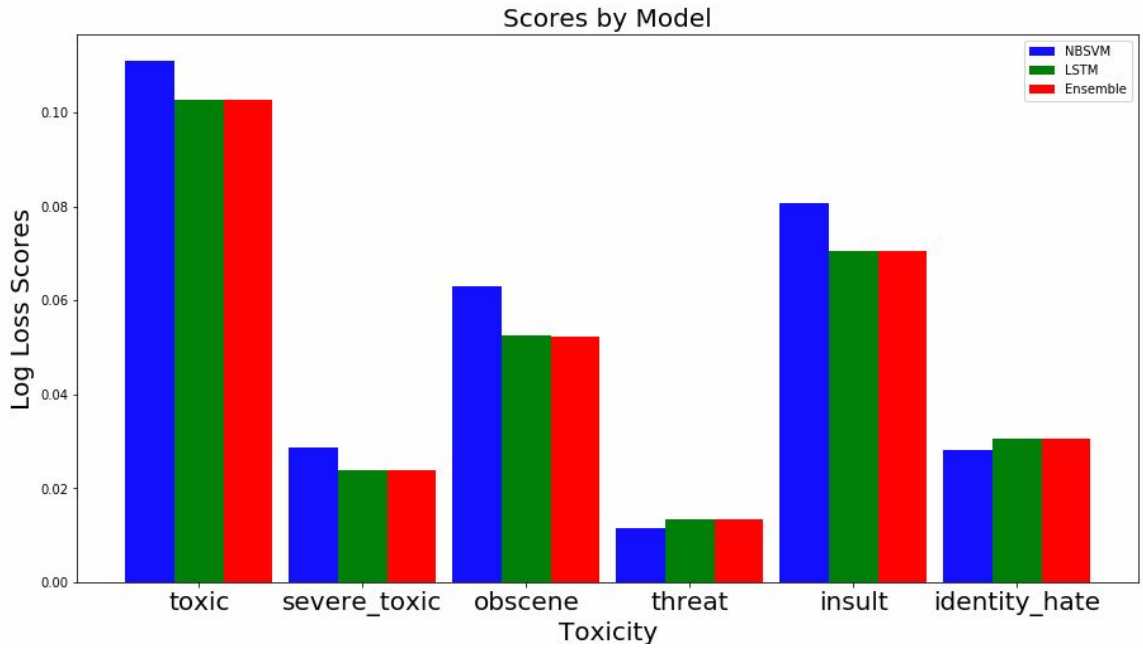
- toxic - 0.86139816
- severe_toxic - 0.02786827
- obscene - 0.25936157
- threat - 0.03552454
- insult - 0.44790095
- identity_hate - 0.08017173

While the model is very confident that the model is toxic(86%) it is less confident on the other types of toxicity. It does not predict very well that the comment is identity_hate(8%) and obscene(25%).

The model does do what it was designed to do but it is still not perfect. It is a good start at detecting toxicity but it alone is not good enough to moderate a discussion.

Conclusion

Free-Form Visualization



The bar chart above shows the log loss scores for each type of toxicity for each model. Even though the NBSVM has a worse overall score it has a better score when it comes to the threat and

identity_hate categories. This shows that the two models have different strengths. So depending on the situation one might prefer the NBSVM model. One such example would be if the owners of a message board were willing to allow obscene comments but still did not want threatening or hate speech, in this case the nbsvm model could be considered the better choice.

Reflection

This project can be broken into 2 distinct parts, the NBSVM and LSTM models. The two models were implemented separately and even though the two models are different they both followed the same general process.

- Preprocess the text, turn the text into usable form.
- Implement a basic form of the prediction model.
- Test the results
- Improve/add to basic model until it achieves a satisfactory result.

One of the parts that I found the most difficult was trying to understand the LSTM model. I previously did not have any experience with LSTM models and I discovered that just adding more layers or parameters did not immediately improve the results. One of the most interesting things to me was that the NBSVM model performed as well as it did. I expected the LSTM model to blow it out of the water when in reality it only performed a little better. I believe that LSTM models are one of the best ways to determine text sentiment but I do not believe it is good enough to moderate comments unsupervised.

Improvement

There are few ways to improve upon the lstm model. The first way to improve the model would be to give it more data, there is almost an unlimited amount of comments being produced online everyday. The difficult part is labeling the data for use. In the absence of additional data two other options come to mind. The first would be to find a way to reduce overfitting the LSTM model, even with dropout it overfits very quickly. Another way to improve the model would be to use the NBSVM model in regards to the threat and identity hate categories and the LSTM model for the other categories.