Machine Learning Engineer Nanodegree

Identify and Classify Toxic Comments Online

Kirupakaran Harikrishnan
November 06, 2018

I. Definition

Project Overview

The idea of online forums and discussion sections is to promote meaningful conversations. Nowadays every website has a comment section and it is very difficult to deal with toxic comments. Every comment needs to be moderated which takes a lot of time. The emergence of massive social networking websites like Facebook and Reddit lead to millions of people posting content and commenting on them. People including children interact in the internet and there is a need to control the content and encourage a healthy environment and facilitate conversations, and ensure that they don't give up on stop expressing themselves. Many news websites now have started disabling comments sections. Most of these websites have human moderators behind them monitoring them constantly which is very inefficient.

There is a need for tools to automatically monitor, categorize and flag comments. Also, different websites may need to control different kinds of content. There is a need to build a multi-headed model that's capable of detecting different types of of toxicity like threats, obscenity, insults, and identity-based hate.

In this project, I've created a machine learning model that is capable of identifying toxic comments as well as classifying them so it is easy for the moderators and other tools to easily filter them. This project is based on the Kaggle competition "Toxic comments challenge" by Jigsaw and the datasets are obtained from kaggle. The dataset represents comments from wikipedia talk page discussions.

Problem Statement

The problem comes under Natural Language Processing, more specifically a sentiment analysis multi classification problem. The problem is solved by, given a text, identify whether the text is clean or toxic. If it is toxic, classify the text as toxic, severely toxic, insult, threat, obscene, identity-based hate. Also, this is a multilabel classification problem. So a given text can have more than one toxic tag.

The tasks involved to solve the problem are:

1. Download and preprocess the wikipedia comments dataset
2. Tokenize the text data as vectors so they can be passed to neural networks
3. Create a model that can process the text data and classify the text
4. Train and test the model
5. Use the model to make further predictions

The problem can be solved by using an algorithm that takes comment as input and outputs a list of probabilities whether it is toxic and the type of toxicity, i.e.,
Given a comment C -> algorithm -> r [toxic, Severe_toxic, Obscene, Threat, Insult, identity_hate]

Metrics

The Kaggle competition suggests using the Area under a Receiver Operating Characteristic[1] (AUROC or AUC) and that'll be used as the evaluation metric.

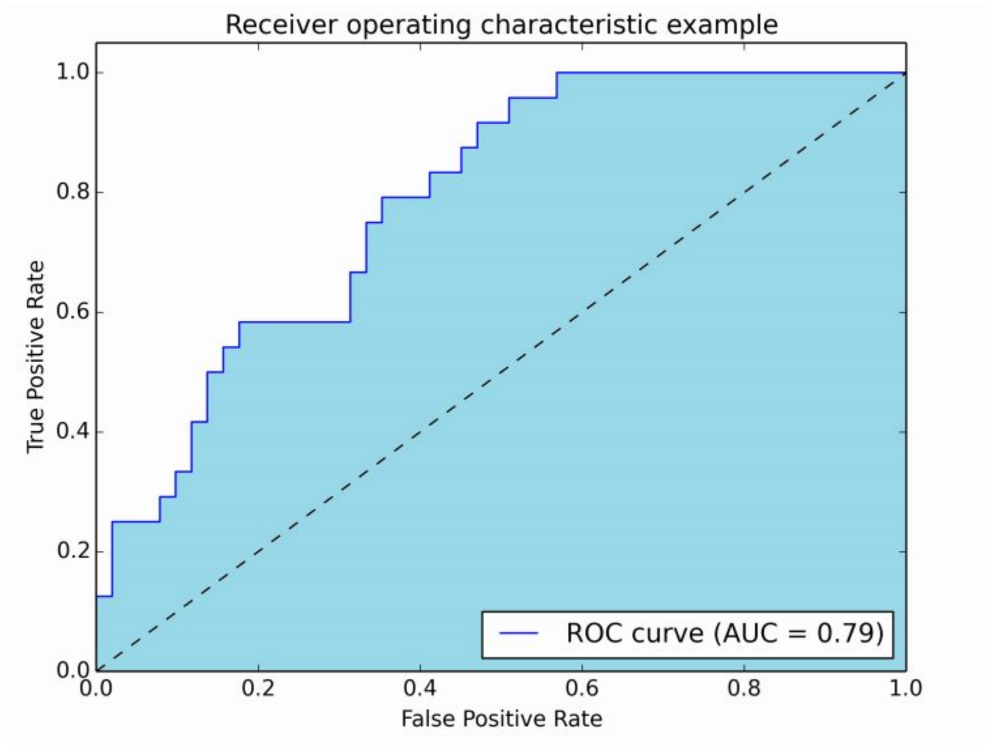
The AUROC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example, i.e. $P(score(x^+) > score(x^-))$

The AUROC curve is based on True positive rate and False positive rate.

TPR = TP / (TP + FN)
FPR = FP / (FP + TN)

TP - True positives, FP - False positives, TN - True negatives, FN - False negatives

The TPR and FPR are computed with many different threshold between 0 and 1 and plotted as a graph. This curve is called the ROC curve and the area under the curve is called AUROC. Higher the AUROC value, lower the false positives and false negatives.



(Fig. 1)

As we will see below, the class distribution is imbalanced and AUROC is a much better metric than accuracy in this case, as the curve represents how well the probabilities from the positive classes are separated from the negative classes, and is much more suited for multilabel and multiclass classification problems.

II. Analysis

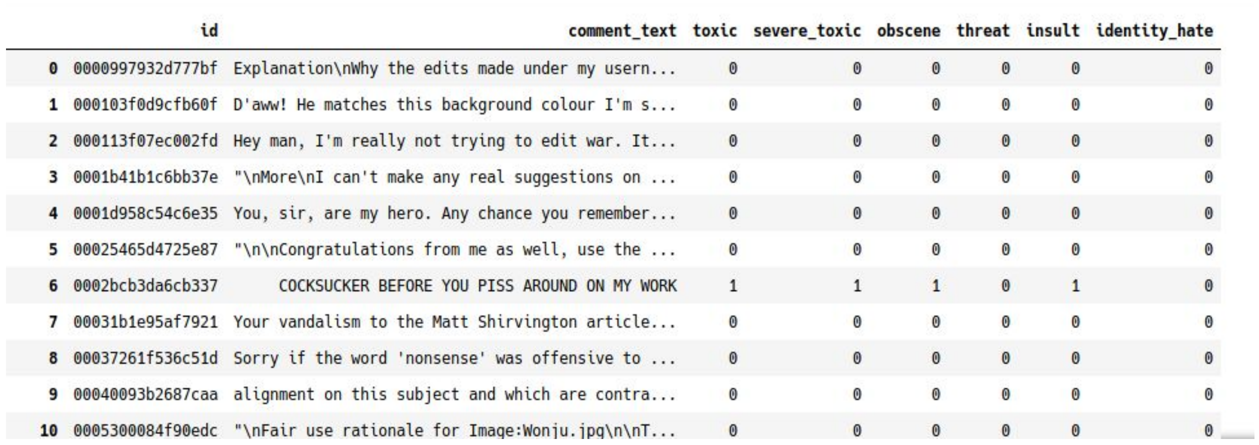
Data Exploration and Visualization

The primary data is wikipedia talk comments provided by Kaggle[2] labeled by human raters for toxic comments. The training data contains 160k rows and the test data contains 153k rows. Of the test data, 63958 are labelled and the rest are not labelled. Only the label part is going to be used for testing.

The training data contains the following columns:
Id, Comment_text, Toxic, Severe_toxic, Obscene, Threat, Insult, Identity_hate

Id is an alphanumeric string and is not used. Comment text is the actual text that is used for classification. The rest of the columns are the output labels that indicate the probabilities of various tags (0 or 1).

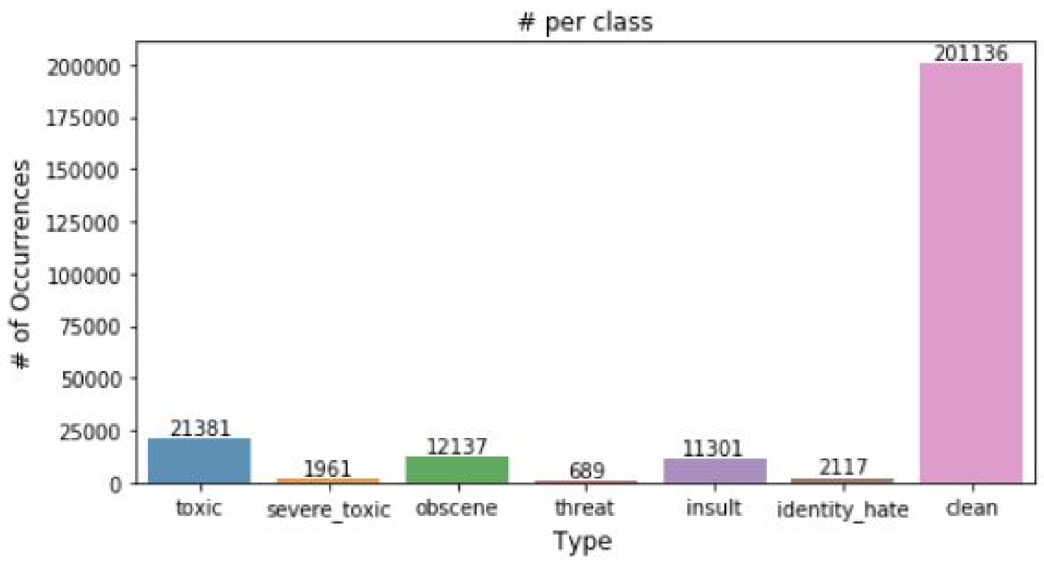
Here's a peek of the sample training data



	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0
5	00025465d4725e87	"\n\nCongratulations from me as well, use the ...	0	0	0	0	0	0
6	0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
7	00031b1e95af7921	Your vandalism to the Matt Shirvington article...	0	0	0	0	0	0
8	00037261f536c51d	Sorry if the word 'nonsense' was offensive to ...	0	0	0	0	0	0
9	00040093b2687caa	alignment on this subject and which are contra...	0	0	0	0	0	0
10	0005300084f90edc	"\nFair use rationale for Image:Wonju.jpg\n\nT...	0	0	0	0	0	0

And some statistics of the total data. The clean data far outweigh the rest of the data. Even among the various tags, some tags have very few samples only. Also, the total number of tags is greater than the total number of comments indicating that some comments have multiple tags.

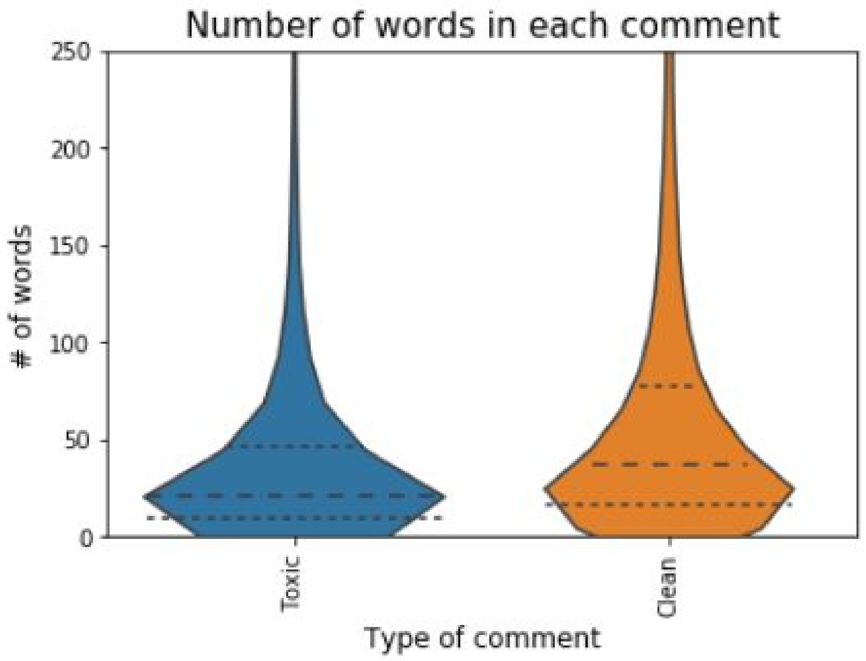
Total comments	223533
Total clean comments	201136
Total tags toxic	21381
severe_toxic	1961
obscene	12137
threat	689
insult	11301



(Fig. 2)

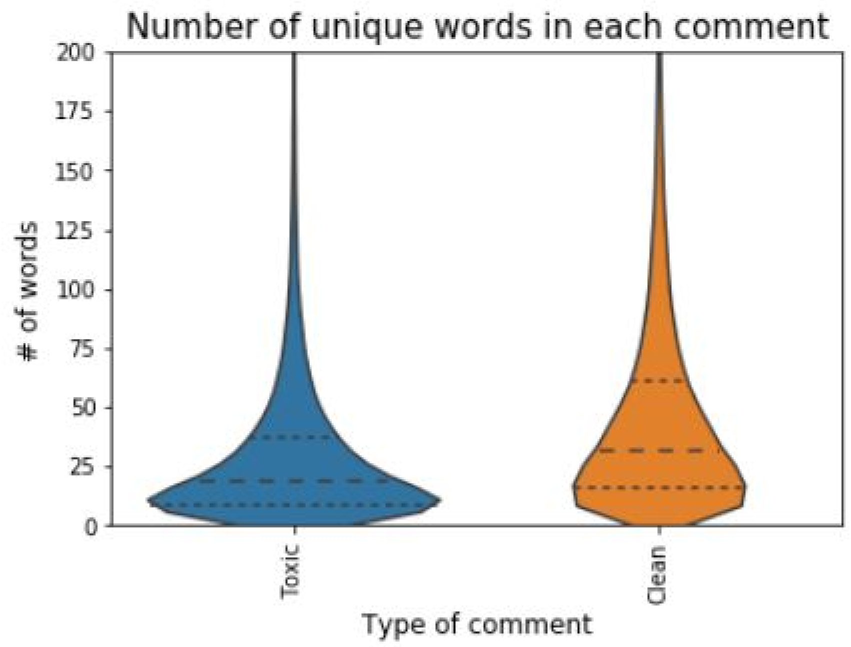
The input training and test data have been collected by volunteers and may be slightly subjective. Also the training set doesn't have any null values. So we don't have to worry about fixing null issues here.

The plot below shows the number of words in each comment for both toxic and clean (non-toxic) comments. As we can see, the bulk of the comments have 0 to 50 words for both toxic and clean comments. However, the toxic comments is much more likely to have fewer words than clean comments.

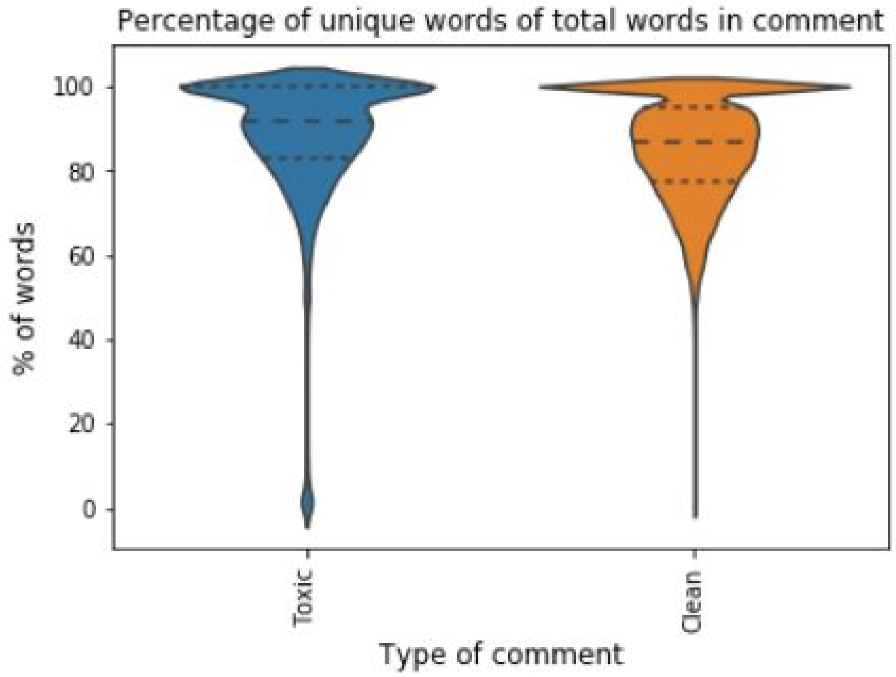


(Fig. 3)

Fig. 4 shows that plot for the number of unique words is also similar to plot for number of words. And Fig. 5 confirms that by showing that the bulk of the words in each comment is unique.


Number of unique words in each comment
# of words
0
25
50
75
100
125
150
175
200
Toxic
Clean
Type of comment

(Fig. 4)


Percentage of unique words of total words in comment
% of words
0
20
40
60
80
100
Toxic
Clean
Type of comment

(Fig. 5)

All of these plots indicate that we don’t have to worry much about duplicate words (even though there is a slight bulge in the toxic comments in fig. 5, it is very small) or very long comments. These findings will help in fine-tuning the model.

Algorithms and Techniques

The problem can be solved by using an algorithm that takes comment as input and outputs a list of probabilities whether it is toxic and the type of toxicity, i.e.,
Given a comment C -> algorithm -> r [toxic, Severe_toxic, Obscene, Threat, Insult, identity_hate]

The most used and simple baseline models for text are Naive Bayes classifiers and support vector machines. An algorithm that combines both NB and SVM classifier and works reasonably well is Naive Bayes - Support Vector Machine (NB SVM)[3] which will be used as the baseline model.

However, the problem with these models and other feedback models like CNNs is they don’t keep track of the sequential data, i.e., they don’t keep track of the context of the words in sentences and they don’t perform well for long texts and they tend to overfit training data. To overcome this, Recurrent Neural networks are used. In a recurrent neural network, each node has a feedback loop that takes previous input at a given time step.

Long Short Term Memory network[4], a form of Recurrent Neural Network algorithm is an algorithm that is designed for natural language processing and one that is proven to works well and that’ll be the basis of the solution. LSTM networks overcome the disadvantage of traditional RNNs where they fails to derive context from the long running sequential data. A form of LSTM called Bidirectional LSTM is used which has two networks one for the normal sequence of text and one for the reversed sequence text. Also, a Convolutional Neural Network layer is used after LSTM layer which extracts local features from the network. This combination is inspired by this post which successfully used this model to improve performance.

Before feeding the data to the network, data needs to be tokenized. Text data needs to be converted to numeric form where each word is represented by a number. Then this numeric data is used to convert each comment as a vector representation with indices representing the actual word (One-hot encoding).

However, this vector representation is sparse and inefficient as each unique word will be occupying space in the vector. To overcome this, the data will be preprocessed using word embeddings that represent each word as a vector and this representation is based on the usage of words and words with similar meanings will have similar representation. The word embedding used in my model is FastText[5][6][7] developed by Facebook after evaluating Word2Vec, Glove, FastText embeddings.

Once the model is embedding is done, the model is defined and data is fit to the model and the model is tested using test data.

To summarize:

1. Prepare and preprocess the data. In this step, remove numbers and punctuations and replace words with their stems.
2. Create vector representation of the data using word embeddings
3. Define the model and define various layers of the model. The model I defined receives embedded data as the first layer, followed by LSTM layer and CNN layer. Output layer has a sigmoid function.
4. Train and test the model using AUROC as a metric.

Benchmark

Support Vector Machines are one of the most used algorithms for text classification and that can be used as the Benchmark model. An algorithm based on SVM and Naive Bayes is SVM-NB[3] which gives better performance than traditional SVM is the recommended benchmark in the kaggle competition and that is used as the benchmark. The overall AUROC score achieved by this model is 73.9%.

III. Methodology

Data Preprocessing

The Wikimedia talk data provided by Jigsaw is mostly clean without any null values. But as seen in the previous section, the data has a class imbalance issue. Most of the data is clean and even among the toxic data some classes are under represented. However, the metric chosen, AUROC, is unaffected by this class imbalance issue.

The preprocessing steps done over the dataset are:

1. Identify and handle all the null values in the dataset - the dataset doesn't contain any null values.
2. Remove special characters and numbers in the dataset - They don't contribute anything to the model and is removed using simple regexes. This also takes care of emoticons, ip addresses etc.
3. Convert all the text to lowercase and split each comment into array of individual words

4. Remove stop words from the text - stop words are words like the, a, to, of etc. These words don't add give any context to the text and don't add value to the model.
5. Each individual word is replaced with its stem - Stemming replaces words with their stem values i.e., remove any prefixes and suffixes. This reduces the vocabulary space.
6. Comments are trimmed and padded to uniform length of 100 words - Each individual comment may be of different length and this needs to be normalized as neural network expects each comment to have same number of features. Length is chosen as 100 because as seen in the Data exploration section (Fig. 3 and 4) most of the comments contain less than 100 words and less than 100 unique words (bulk of them between 10 to 50).

All of these preprocessing steps significantly reduces the size of the input.

Once this is done, text data needs to be encoded as numeric value.. A dense representation is created using word embeddings.

Embeddings

To feed text data to neural networks, we need to tokenize them as numbers which represents each word in the text. The common form, one-hot encoding, is very ineffective as it creates very sparse vector model. To overcome this, Vector Space Model is created which represents words in a vector space where similar words are embedded nearby each other. The VSMs depends on Distributional Hypothesis which states that which states that word appearing in the same context share similar meaning. Two approaches that follow this principle are:

1. Frequency based embedding - Find the frequency of each word in the dataset to create dense vector for each word.
2. Prediction based embeddings. - Create word embeddings based on the context of each word in the given vocabulary using neural networks.

I'm going with prediction based embedding as it is denser and efficient and there are pre-trained word embeddings available such as GloVe[8] by Google & stanford and FastText[7] by Facebook. Both of them contain word embeddings for millions of words trained over a very large set of general english vocabulary.

Specifically, I've chosen FastText as the pre-trained word embedding as it performed better on average than GloVe (not by a huge margin though).

FastText embedding model used is trained on Common Crawl dataset which contains 600 billion tokens and contains 2 million word vectors. And created using skip-gram model based on these two papers.

Implementation

Model description:

1. The input layer accepts list of word indices of length 100 (As described in the previous section, each comment is transformed into a vector of length 100).

   ```
   input = Input(shape=(maxlen, )) # maxlen = 100 as defined earlier
   ```

2. The first layer of the network is the embedding layer. The embedding layer receives word vectors containing integer sequences as input and converts those integers to dense vectors based on the weight. The weights are provided by fastText. The output of this layer is trainable feature vectors of dimension 300 (embedding vector dimension)

   ```
   x = Embedding(max_features,
           embed_size,
           weights=[embedding_matrix])(input) # max_features = 20000 (size of the
   vocabulary i.e., most frequent 20k words in the dataset are taken), embed_size = 300
   (fastText pre-trained vector dimension)
   ```

3. A dropout layer is added for regularization to avoid overfitting and spatial dropout is added because the adjacent vectors are correlated and is recommended after the first layer. Spatial dropout drops the entire feature vector (embedded representation of a word).

   ```
   x = SpatialDropout1D(0.2)(x)
   ```

4. The next layer is the LSTM layer and a Bidirectional LSTM layer[9] is used. Bidirectional lstm layer can get information from both past and future states by training 2 LSTMs, one on the normal input sequence and one a reversed copy of the input sequence. Bidirectional LSTMs are shown to converge faster and perform better than normal LSTMs[10]. The network size is defined as 120 based on trial and error.

   ```
   x = Bidirectional(LSTM(120, return_sequences=True,name='lstm_layer'))(x)
   ```

5. The next layer is a 1D CNN layer with a kernel size of 3 which is shown to perform better than simple LSTM models[11].

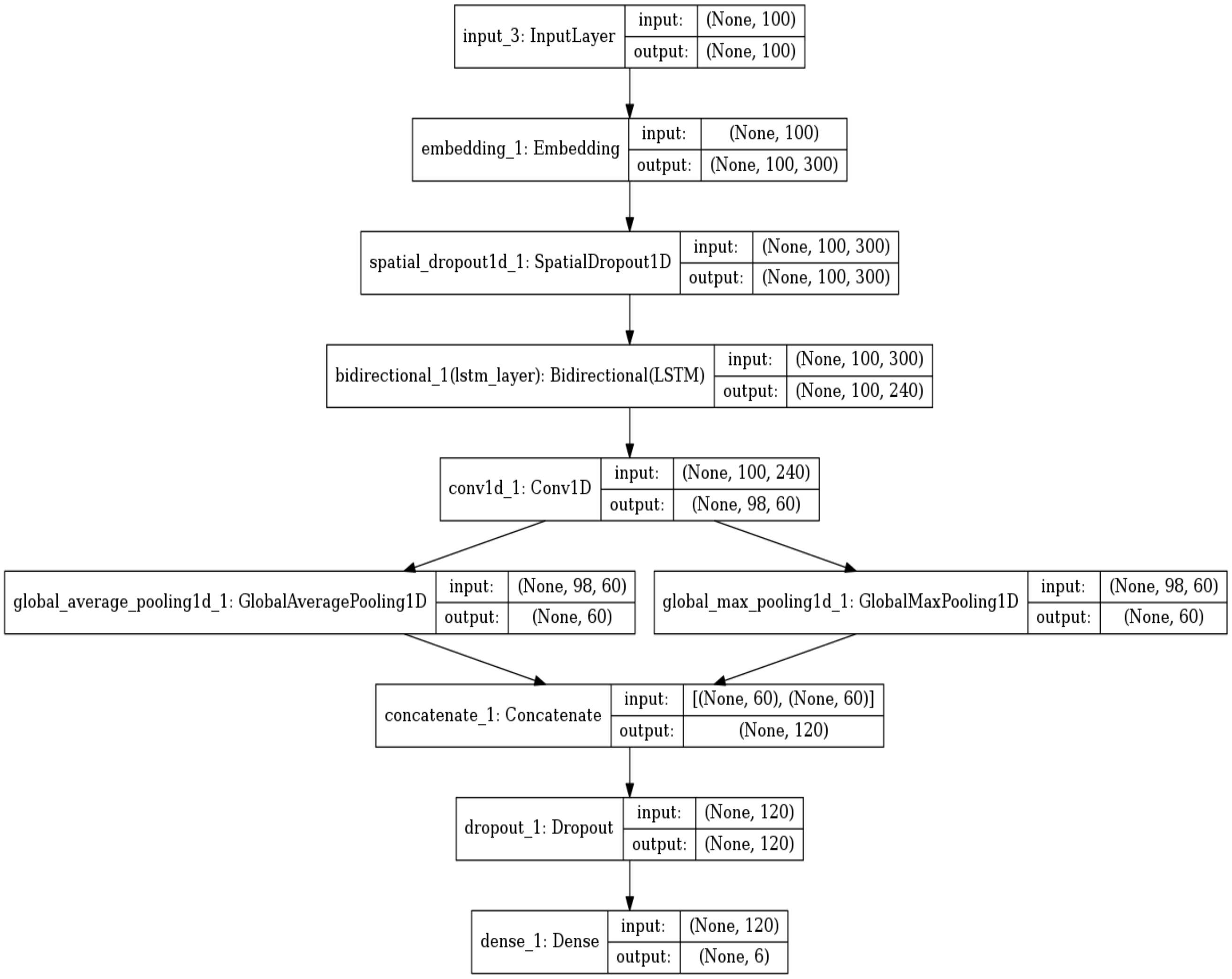
```
x = Conv1D(60, kernel_size = 3, padding = "valid", kernel_initializer = "he_uniform")(x)
```

6. The next layer is the pooling layer which is a combination of global max pooling and global average pooling and reduces input data to 1d vector

```
avg_pool = GlobalAveragePooling1D()(x)
max_pool = GlobalMaxPooling1D()(x)
x = concatenate([avg_pool, max_pool])
```

7. The next layer is the dropout layer
```
x = Dropout(0.2)(x)
```

8. The output layer has 6 nodes and is applied sigmoid function to get predictions for the 6 output labels


input_3: InputLayer | input: (None, 100) | output: (None, 100)
embedding_1: Embedding | input: (None, 100) | output: (None, 100, 300)
spatial_dropout1d_1: SpatialDropout1D | input: (None, 100, 300) | output: (None, 100, 300)
bidirectional_1(lstm_layer): Bidirectional(LSTM) | input: (None, 100, 300) | output: (None, 100, 240)
conv1d_1: Conv1D | input: (None, 100, 240) | output: (None, 98, 60)
global_average_pooling1d_1: GlobalAveragePooling1D | input: (None, 98, 60) | output: (None, 60)
global_max_pooling1d_1: GlobalMaxPooling1D | input: (None, 98, 60) | output: (None, 60)
concatenate_1: Concatenate | input: [(None, 60), (None, 60)] | output: (None, 120)
dropout_1: Dropout | input: (None, 120) | output: (None, 120)
dense_1: Dense | input: (None, 120) | output: (None, 6)

(Fig 7)

Once the model is defined, model is compiled using binary cross-entropy as loss function and adam as optimize and trained in 2 epochs with a batch size of 32.

Refinement

The model began as a simple LSTM model with no embeddings and no pre-processings and got an auroc of 96.7%.

Refinement steps taken in the model are:

1. Data pre-processing was done including removing symbols and numbers, replace words with their stems, remove stop-words.
2. Several word embeddings were tried starting with word2vec which is trained with the words in the dataset, followed by pretrained embeddings, gloVe and fastText and fastText was selected in the final model based on the best auroc and accuracy.
3. The feature length (length of the words in each sentence) has been chosen as 100 based on the visualizations Fig 3-5 and based on trials.
4. The input bidirectional layer is given 120 cells based on trial and error for best performance considering training time.
5. The CNN layer has 60 filters and kernel stride of 3
6. A spatial dropout layer and global dropout (combined average and max dropout) layer are added for regularization to prevent overfitting.
7. Data was trained on ~128k samples and validated on ~32k samples. Test data contained ~64k samples. Training was done in 2 epochs with batch size of 32. Adding more epochs resulted in data overfitting where the difference between training loss and validation loss increased.

The final model produced an AUROC score of 97.70% which is much better than the benchmark model with a score of 73.9%.

IV. Results

Model Evaluation and Validation

The model test results for various metrics are given below.

	Toxic	Severely Toxic	Obscene	Threat	Insult	Identity Hate	Overall
AUROC	0.9616	0.9829	0.9737	0.9862	0.9678	0.9765	0.9770

Accuracy	0.9241	0.9942	0.9564	0.9965	0.9603	0.9910	0.8819
Precision	0.9413	0.9931	0.9638	0.9962	0.9613	0.9895	0.5898
Recall	0.9241	0.9942	0.9564	0.9965	0.9603	0.9910	0.7331
F1 Score	0.9300	0.9935	0.9593	0.9964	0.9608	0.9899	0.6462

The first 6 columns indicate the micro metric for each label. The final column indicates the weighted average of all the labels. Weighted average takes care of class imbalance issue by multiplying each class by its occurrence in the dataset. The sklearn metrics for multilabel classification based on subset accuracy which matches exact label for each labels. This resulted in significantly lower accuracy, f1 score, precision and recall. However, the individual class scores are still high.

To verify the final model, it has been validated with an external twitter dataset[12] containing ~25k tweets with good results. The model is also validated with an external dataset containing ~25k twitter tweets. The dataset was classified for hate comments and offensive comments and since both of them comes under toxic, all the label predictions are taken into consideration for validation.

The model scored an f1 score of 85% and auroc of 91.5% compared to the model described in the paper which has an f1 score of 90% but considering that the paper model is tuned for the dataset, this is a pretty good result. Also many of the tweets in dataset in contained * as filler letters forming masked words in the swear words (which wasn't preprocessed) and this resulted in lower score.

Justification

The final model score is much better than the benchmark model. However, it is not much better than the unoptimized simple lstm model since it already gave a very high auroc score.

	Benchmark (NB-SVM with ngram and tf-idf embedding)	Initial LSTM model (Unoptimized with no refinements)	Final Model
AUROC	0.7395	0.9670	0.9770
Accuracy	0.8813	0.8721	0.8819

Precision	0.6276	0.5289	0.5898
Recall	0.6578	0.7174	0.7331
F1 score	0.6337	0.6002	0.6462

But on comparison with validation data (twitter data), the final model is much better. The simple starter lstm model performed poorly with an AUROC score of 0.49 and f1 score of 0.14 compared to final model's AUROC score of 0.91 and f1 score of 0.85.

Also, each prediction takes around 10-30 ms which is very fast, so this model can be used real-time in browser addons for validation and filtering or during post/comment submissions in websites.

The final model can still be improved by handling masked text (text where words and letters are replaced with symbols or numbers).

V. Conclusion

Free-Form Visualization

Here are some tweets and the predictions by the classifier.
Warning: Below text contain swear words

```
tweet :         !!! RT As a woman you shouldn't complain about cleaning up your house.
&amp; as a man you should always take the trash out...
toxic :  0.60939586
severely toxic :        0.0053854487
obscene :       0.097927704
insult :        0.0026107992
threat :        0.16030043
identity hate :         0.014763704
```

```
tweet :         !!!!! RT  boy dats cold...tyga dwn bad for cuffin dat hoe in the 1st place!!
toxic :  0.5702319
severely toxic :        0.008312527
obscene :       0.18103847
insult :        0.0009736546
threat :        0.12653936
identity hate :         0.011574051
```

tweet :   !!!!!!! RT Dawg!!!! RT @: You ever fuck a bitch and she start to cry? You be confused as shit
toxic : 0.9979918
severely toxic :   .5285919
obscene :   0.99674463
insult :   0.006409194
threat :   0.95108026
identity hate :   0.028900187

tweet :   !!!!!!!!! RT she look like a tranny
toxic : 0.14079762
severely toxic :   0.0033998839
obscene :   0.031504747
insult :   0.0013635928
threat :   0.068856455
identity hate :   0.003289402

tweet :   !!!!!!!!!!!!! RT The shit you hear about me might be true or it might be faker than the bitch who told it to ya &#57361;
toxic : 0.98146135
severely toxic :   0.118581496
obscene :   0.9716401
insult :   0.0026767256
threat :   0.76674604
identity hate :   0.008403031

tweet :   !!!!!!!!!!!!!!!!!!" The shit just blows me..claim you so faithful and down for somebody but still fucking with hoes! &#128514;&#128514;&#128514;"
toxic : 0.9883427
severely toxic :   0.13088322
obscene :   0.977857
insult :   0.005566295
threat :   0.60957795
identity hate :   0.0099708345

tweet :   !!!!!!" I can not just sit up and HATE on another bitch .. I got too much shit going on!"
toxic : 0.9936034
severely toxic :   0.28125128
obscene :   0.9855457
insult :   0.004489582
threat :   0.9165784
identity hate :   0.032002512

tweet : !!!!&#8220; cause I'm tired of you big bitches coming for us skinny girls!!&#8221;
toxic : 0.95407957
severely toxic : 0.078073
obscene : 0.8871417
insult : 0.0042873956
threat : 0.596044
identity hate : 0.019098056

tweet : " &amp; you might not get ya bitch back &amp; thats that”
toxic : 0.9655077
severely toxic : 0.12621342
obscene : 0.9461841
insult : 0.008392385
threat : 0.7525458
identity hate : 0.010969288

tweet : 'why should there be an age limit on movies? it aint their fault if a 13yr old is traumatized from an R rates film bunch of p***s'
toxic : 0.07721218
severely toxic : 0.00163186
obscene : 0.02794344
insult : 0.00030664
Threat : 0.02798235
Identity hate : 0.00236926

As we can see, the classifier is pretty accurate. However, it failed on the final tweet where a word is masked with *.

Reflection

The process used for this project can be summarized as follows:

1. A relevant problem and a public dataset was found
2. Preprocessed and cleaned the data
3. Identified various possible models that can be useful for solving this particular problem
4. Decided on a benchmark model
5. Designed the final architecture of the model based on evaluation of various models
6. Trained the model with data and saved the weights
7. Improved the model with additional preprocessings and tried various word embeddings.

8. Validated the model with external dataset

Step 8 proved very difficult as finding a relevant dataset with labels was a challenge. The data had to be manually examined for the predictions to validate the predictions which was hard since the data didn't contain the relevant labels. Finding the benchmark model and the final model was easy since it was a kaggle competition and the results was public. But, the parameters and embeddings needed to be tuned to ensure the data didn't overfit the data. The kaggle competition models were tuned to achieve highest score in the test data but this is a generic model that can be used for a variety of scenarios, those tunings and tricks had to be avoided. Also, the word embeddings and recurrent networks were new concepts and I had to learn all of them.

Improvement

1. This model is trained with english language. It can be trained with data collected from multiple languages to be effective at regional language forums.
2. This model doesn't deal very well with masked words and typos. Need more preprocessing steps done to handle that.
3. Model is trained with limited data only from wikipedia. Need to collect and train with much more data from variety of forums spanning multiple demographics.

References:
1. The meaning and use of Area Under a Receiver Operating Characteristic
2. Wikipedia talk data for Toxic comments classification challenge by Kaggle and Jigsaw
3. Baselines and Bigrams
4. Long Short Term Memory
5. Enriching Word Vectors with Subword Information
6. Advances in Pre-Training Distributed Word Representations
7. fastText
8. gloVe
9. Bidirectional Recurrent Neural Networks
10. Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures
11. Twitter Sentiment Analysis using combined LSTM-CNN Models
12. Automated Hate Speech Detection and the Problem of Offensive Language