TOXIC COMMENT CLASSSIFICATION

Meven DCunha
Northeastern University,
dcunha.m@husky.neu.edu

Neha Bhangale
Northeastern University,
bhangale.n@husky.neu.edu

ABSTRACT

Maintaining a decorum on an online forum is the need of the hour. Like traditional forms of bullying, cyberbullying is often deliberated and relentless, but it can be even more unnerving because of the anonymous nature of the assault. Automating the process to identify abusive words in a comment would not only save human effort but also increase user safety and improve online discussions. In this paper, we discuss different approaches to develop a series of Deep Learning models capable of predicting the toxicity category and probability of given comment. We begin by exploring the accuracy of several models in detecting toxicity on a test set classified by humans. We will be using the pre-trained FastText embeddings from Facebook to produce a 300-dimension vector for each word in our vocabulary**.** We then tune the best model to achieve competitive results. We have extended our neural network model further to prove that this model is able to detect toxicity level and rightly classify the toxicity level which is not provided by Google's Perspective API. We extend this further to check the accuracy of the best model on Twitter streamed data.

INTRODUCTION

1. Background:

The massive increase in accessibility of the web and freedom of speech has caused in toxic, threat and obscene conversation on the internet. Social networking sites like Facebook, Twitter and online communities like Reddit and Wikipedia have some key victims of online toxicity and hate speech. According to Cyberbullying Research Center, over 80 percent of teens use a cell phone regularly, making it the most popular form of technology and a common medium for cyber bullying. About half of young people have experienced some form of cyber bullying, and 10 to 20 percent experience it regularly. Mean, hurtful comments and spreading rumors are the most common type of cyber bullying. Thus, exploring the Deep Learning models to classify the comments into different toxicity labels. It can help us reduce the content which is toxic and obscene, hence leading to more discussion and user participation in online forum.

Kaggle Challenge:

The Kaggle competition for toxic comment classification about detection of different types of toxic comments by classifying them into one or more labels from 'toxic', 'severe_toxic', 'obscene', 'threat', 'insult' and 'identity_hate'.

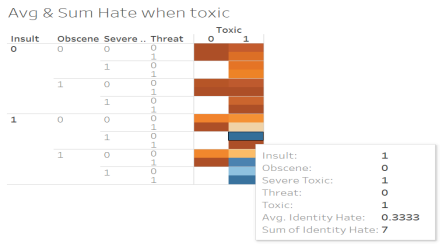
The challenge is particularly interesting because of the heated discussions that toxic content online has influenced the overall health of the society. It is also interesting that the service providers are finally leveraging deep learning to supervise their service in a scalable way.

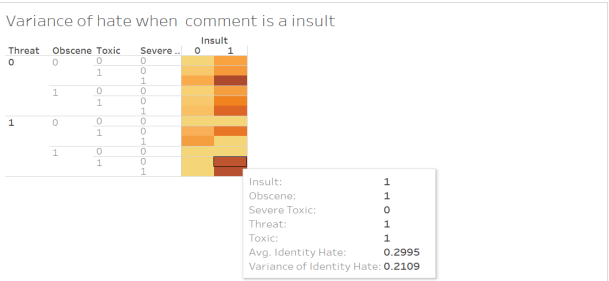
The objective is to develop pre-defined models and improve their accuracy as compared to the outputs provided by Perspective API.

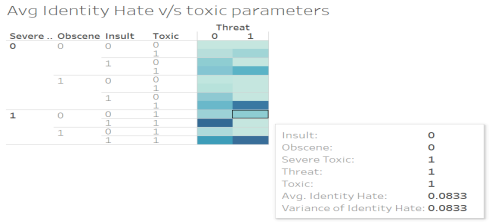
2. Related Work:

Google's Conversation AI team, a research initiative with Alphabet, deals with classifying toxic comments using various models that can be tested using Perspective API. These models have a drawback that it does not have an option to select a toxicity level in an online comment or a given text. For example, some websites may be fine with profanity that is not malicious. In such cases it does not give the user to filter out the classified toxic comments that they are interested in.

DATA ANALYSIS







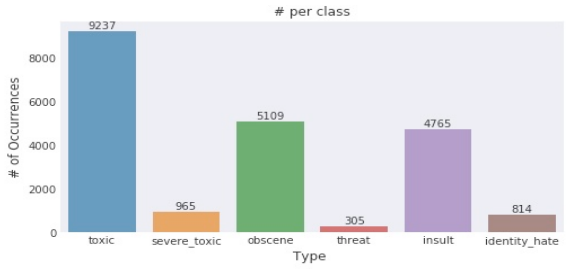
PREPROCESSING

1. Data Cleaning:

The data used consist of many Wikipedia comments which have been labelled by humans according to their relative toxicity. The data includes the following:

- train.csv: the training set, contains comments with their binary labels.
- test.csv: the test set, predict toxicity probabilities for these comments.
- sample_submission.csv: the submission sample with the correct format

On analyzing the train data set, it was noted that the toxic levels in the comments are classified as shown below:



The data consist of 10734904 words out of which the number of unique words are 532299. The 10 most common words are “the”, “to”, “of”, “and”, “a”, “I”, “is”, “you”, “that” and “in”. The English oxford dictionary comprises of 171,476 words. The problem with the above statistics are that we are counting a word uppercased or lowercased as different words altogether. This would not add any relevance to the further processing of our data. To filter and clean our data we perform the following process:

- Remove irrelevant characters (!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n).
- Convert all letters to lowercase (HeLlO -> hello).
- Tokenize our words (hi how are you -> [23, 1, 5, 13]).
- Standardize our input length with padding (hi how are you -> [23, 1, 5, 13, 0, 0, 0]).
- Misspelled words, slang or different word inflections are not combined into single base words. However, the benefit of using a Neural Network is that they do well with raw inputs and hence we would not preprocess the data further for better accuracy.
- Some statistics on the preprocessed data:
  - Vocab size: 210337
  - Longest comment size: 1403
  - Average comment size: 68.22156908210138
  - Stdev of comment size: 101.07344657013672
  - Max comment size: 371
- After preprocessing the data our vocabulary size drops down to a manageable 210,337.

2. Embeddings:

The data representation normally used on our vocabulary is the one-hot encoding where every word is transformed into a vector with a 1 corresponding it its location. For example, if our word vector is [hello, how, are, you] and the words word we are looking for is “hello”, the input vector would be [1,0,0,0]. This tends to work fine unless our vocabulary is huge which is 210000 in this case. This means that we would end up with word vectors that consist mainly of a bunch of 0s.

We have hence taken the approach of using a Word2Vec technique to find continuous embeddings for our words. We will use the pretrained FastText embeddings from Facebook to produce a 300-dimension vector for each word in our vocabulary.

- P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching Word Vectors with Subword Information

The benefits to this continuous embedding is that words with the similar predictive power will appear closer together on our word vector. Although, the downside to this is that it creates more of a black box where the words with the most predictive power gets lost in the numbers.

APPROACH

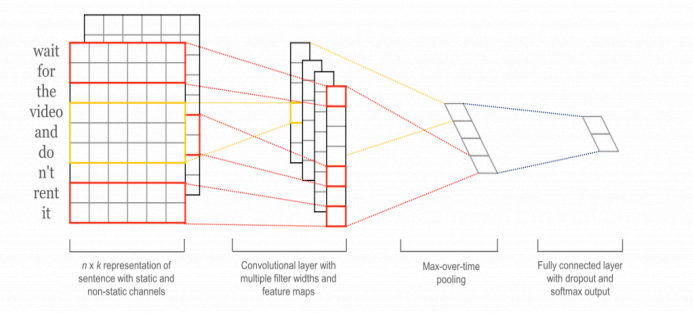
The idea is to use the train dataset to train the models with the words in the comments as predictor variables and to predict the probability of toxicity level of a comment. A pre-trained model that gives the best accuracy to various user comments to combat the ongoing issue of online forum abuse. Our project is focused on developing a series of neural network models. The goal is to find the strengths and weakness of different Deep Learning models on the text classification task. We developed 3 specific Neural Network models for this project which are as follows:

1. Convolution Neural Network (CNN with character-level embedding)
2. Convolution Neural Network (CNN with word embedding)
3. Recurrent Neural Network (RNN) with Long Short Term Memory (LSTM) cells

We intend to test the above models trained with the Wikipedia data and word embedding
Twitter to predict the toxicity levels of the tweets.

MODEL STRUCTURES

Although, Convolution Neural Networks (CNNs) are mainly used for mainly used for image classification problems and computer vision systems, more recently CNN's have also been used to solve problems in Natural Language Processing.



The most natural fit for CNNs is task classifications like Sentiment Analysis, Topic Categorization or Spam Detection. Convolutions and pooling operations lose information about the local order of words, so that sequence tagging or Entity Extraction is a bit harder to fit into a pure CNN architecture. The CNN architecture achieves very good performance across datasets, and state-of-the-art on a few. The input layer is a sentence comprised of concatenated FastText embedding which is followed by a convolutional layer with multiple filters. This is then followed by a max pooling layer and finally a fully connected layer.

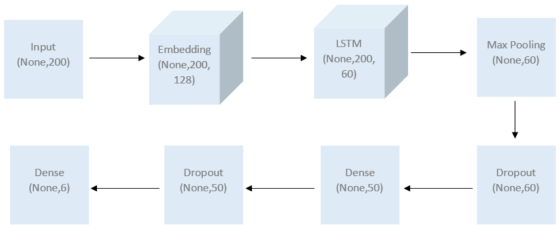
1. CNN (with character embeddings)

   A character level model will use the character as the smallest entity. This can help in dealing with common misspellings, different permutations of words and languages that rely on the context for word conjugations. The model reads characters one by one, including spaces, and creates a one-hot embedding of the comment. The CNN model we used consist of a 1-dimensional convolutional layer across the concatenated character embeddings for each character in the input comment.

2. CNN (with word embeddings)

   The CNN model used consists of one 1-dimensional convolutional layer across the concatenated word embeddings for each input comment. The convolutional layer has 128 filters with a kernel size of 5 so that each convolution will consider a window of 5 word embeddings. The next layer is a fully connected layer with 50 units which is then followed by the output layer.

3. RNN (with word embeddings)



The RNN type model used here is the LSTM model. This model is attractive because the individual cell states in the model can remove or add information to the cell state through gates layers. This is useful in practice because it allows the model to remember insights derived from words throughout the comment. The LSTM model consist of one densely connected layer with 60 units across the concatenated word vectors for each of the words in the comment.

TRAINING

We train our model on 154783 of the comments in the training set and evaluate the performance on the remaining 4788 comments. Given that NLP tasks usually require a huge amount of training data, we wanted to maximize the size of our training set while still maintaining a large enough development set to have a statistically significant evaluation of how well our model generalizes.
The training process for each of our models involves minimizing the mean binary cross entropy loss across the training set, the formula for which is:

$$-\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right]$$

We initially used the mean squared error as our loss function, but we encountered issues that our models would converge to predicting near zero for every single class for every single input. Mean binary cross entropy loss avoided this problem by better handling each toxicity class independently instead of as a 6-dimensional vector.

EVALUATION METRICS

To evaluate our models, we look at its Area Under the Receiver Operating Characteristic (AUC ROC) score. We will be looking at the probability that our model ranks a randomly chosen positive instance higher than a randomly chosen negative one. With data that mostly consists of negative labels i.e. no toxicity, our model could just learn to always predict negative and end up with a high accuracy. AUC ROC helps correct this by putting more weight on the positive examples.

We then created an app for prediction. This app pipeline can be put into production for toxic comment classification. It will take in a string and return the odds that it is any one of the toxic classifications this filling the downfall of Google's Perspective API to classify a comment based on its type of toxicity.

A sample of different inputs to this app for the CNN with word embeddings model is as shown below:

```
Toxicity levels for 'go jump off a bridge jerk':
Toxic:          99%
Severe Toxic:   12%
Obscene:        91%
Threat:         2%
Insult:         93%
Identity Hate:  4%

Toxicity levels for 'i will kill you':
Toxic:          87%
Severe Toxic:   8%
Obscene:        41%
Threat:         63%
Insult:         59%
Identity Hate:  12%

Toxicity levels for 'have a nice day':
Toxic:          0%
Severe Toxic:   0%
Obscene:        0%
Threat:         0%
Insult:         0%
Identity Hate:  0%

Toxicity levels for 'hola, como estas':
Toxic:          0%
Severe Toxic:   0%
Obscene:        0%
Threat:         0%
Insult:         0%
Identity Hate:  0%

Toxicity levels for 'hola mierda joder':
Toxic:          16%
Severe Toxic:   0%
Obscene:        9%
Threat:         0%
Insult:         1%
Identity Hate:  0%
```

A sample of different inputs to this app for the CNN with character embeddings model is as shown below:

```
Toxicity levels for 'go jump off a bridge jerk':
Toxic:          82%
Severe Toxic:   4%
Obscene:        59%
Threat:         0%
Insult:         58%
Identity Hate:  11%

Toxicity levels for 'i will kill you':
Toxic:          75%
Severe Toxic:   11%
Obscene:        37%
Threat:         33%
Insult:         40%
Identity Hate:  15%

Toxicity levels for 'have a nice day':
Toxic:          2%
Severe Toxic:   0%
Obscene:        0%
Threat:         0%
Insult:         0%
Identity Hate:  0%

Toxicity levels for 'hola, como estas':
Toxic:          22%
Severe Toxic:   2%
Obscene:        12%
Threat:         1%
Insult:         7%
Identity Hate:  3%

Toxicity levels for 'hola mierda joder':
Toxic:          25%
Severe Toxic:   0%
Obscene:        6%
Threat:         1%
Insult:         10%
Identity Hate:  2%
```

A sample of different inputs to this app for the RNN with word embeddings model is as shown below:

```
Toxicity levels for 'go jump off a bridge jerk':
Toxic:          100%
Severe Toxic:   10%
Obscene:        96%
Threat:         1%
Insult:         86%
Identity Hate:  0%

Toxicity levels for 'i will kill you':
Toxic:          99%
Severe Toxic:   3%
Obscene:        3%
Threat:         95%
Insult:         6%
Identity Hate:  0%

Toxicity levels for 'have a nice day':
Toxic:          0%
Severe Toxic:   0%
Obscene:        0%
Threat:         0%
Insult:         0%
Identity Hate:  0%

Toxicity levels for 'hola, como estas':
Toxic:          0%
Severe Toxic:   0%
Obscene:        0%
Threat:         0%
Insult:         0%
Identity Hate:  0%

Toxicity levels for 'hola mierda joder':
Toxic:          37%
Severe Toxic:   0%
Obscene:        13%
Threat:         0%
Insult:         0%
Identity Hate:  0%
```

RESULTS

Analysis of all the 3 model performances are as shown below:

Model	Embeddings	AUC	LOSS	Epoch
RNN	Word	0.98488	0.0616	1
CNN with word	Word	0.98603	0.0314	2
CNN with character	Character	0.95649	0.0875	1

We found that our CNN model with word embeddings slightly outperformed the LSTM model with an accuracy of 98.6% with one epoch. On running this model for a second epoch there was no increase in the accuracy although there was an improvement in the loss factor which went down to 3.14%. The CNN model with character encoding gave a comparatively bad performance with an accuracy of just 95.56%.

TWITTER ANALYSIS

We have validated our best model i.e. CNN with word embeddings with data from twitter with a hashtag #TRUMP. Below is the list of some of the tweets and their toxic classification.

The below 2 tweets shown have been rightly classified into their toxic category.

```
Toxicity levels for 'RT @treyinla: I've thought #KanyeWest was a rude, selfish, narcissistic idiot ever since that night he grabbed Taylor Swift's award fro
n he.':
Toxic:        76%
Severe Toxic:  0%
Obscene:       8%
Threat:        0%
Insult:       55%
Identity Hate: 0%


Toxicity levels for '@FoxNews Because unfortunately the Left &amp;amp; MSM narrative - "God is Dead...Gay is good..and #Islam is peaceful" |
dyveqzrDu':
Toxic:        50%
Severe Toxic:  0%
Obscene:       2%
Threat:        0%
Insult:        4%
Identity Hate: 29%
```

CONCLUSION

By going about this project, we could get a much better understanding of how to build, tune and analyze the performance of a Deep NLP model. By working with different types of Neural Network models with word embedding initializations, we could conclude which models may be better suited for the task of toxic comment classification. We found that the best model in our case was the CNN model with word embeddings. Although its performance accuracy is marginally better than the LSTM model, it could gain a better categorization of toxic comment accuracy. This was noted by passing an input threat comment to the app function developed to find the toxicity levels of the comment for different models. The input comment was "I will kill you" and it was passed to the CNN model with word embeddings and the LSTM model. The outputs were as follows:

CNN Model	LSTM Model
Toxicity levels for 'i will kill you': Toxic: 99% Severe Toxic: 3% Obscene: 3% Threat: 95% Insult: 6% Identity Hate: 0%	Toxicity levels for 'i will kill you': Toxic: 79% Severe Toxic: 13% Obscene: 67% Threat: 8% Insult: 42% Identity Hate: 17%

The threat percent for the CNN model is 95% while in the LSTM model it is just 8%. Thus, the classification of toxicity for the CNN model is much better when compared to that of the LSTM model.

REFERENCES

[1] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge#description

[2] https://motherboard.vice.com/en_us/article/qvvv3p/googles-anti-bullying-ai-mistakes-civility-for-decency

[3] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[4] http://nymag.com/selectall/2017/02/google-introduces-perspective-a-tool-for-toxic-comments.html

[5] https://web.stanford.edu/class/cs224n/reports/2762092.pdf

[6] https://www.kaggle.com/sbongo/for-beginners-tackling-toxic-using-keras

[7] https://datascience.stackexchange.com/questions/11619/rnn-vs-cnn-at-a-high-level

[8] https://www.depends-on-the-definition.com/classify-toxic-comments-on-wikipedia/

[9] http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

[10] http://web.stanford.edu/class/cs224n/reports/6838795.pdf

[11] http://dsbyprateekg.blogspot.com/2017/12/can-you-build-model-to-predict-toxic.html

[12] https://arxiv.org/pdf/1802.09957.pdf

[13] http://web.stanford.edu/class/cs224n/reports/6909170.pdf

[14] http://web.stanford.edu/class/cs224n/reports/6838601.pdf

[15] http://web.stanford.edu/class/cs224n/reports.html
[16] https://medium.com/@srjoglekar246/first-time-with-kaggle-a-convnet-to-classify-toxic-comments-with-keras-ef84b6d18328