Hate Speech Detection

Harsh (170020037), Karthik (160020102), Kaushal (160020104), Utkarsh (150020087)

November 27, 2018

Abstract

In this project, we attempt to approach Hate Speech Classification using Long Short Term Memory (LSTM) and analyzing its accuracy

1 Introduction

In the recent years, circulation of fake news and hate speech has become a pressing issue. They are known to be decisive in politics including the 2016 US Presidential election and also in case of the Rohingya issue in Myanmar. Thus, no wonder, social media giants are trying to reduce the amount of hate speech circulated as much as possible without the requirement of a user to manually flag the post or tweet for the human moderators to take a call. Deep Learning approach to hate speech detection would greatly reduce and keep in check the growing resentment in social media.

2 Literature Survey

Most of the earlier work revolving around Hate Speech Detection included manual feature extraction (1). Recently, the problem has attracted methods from Machine Learning and Natural Language Processing. These approaches can be divided into two categories: classical methods - one relies on manual feature engineering that are then consumed by algorithms like Logistic Regression (2); deep learning methods - employs neural networks to automatically learn multi-layers of abstract features from raw data (3)(4) (deep learning methods).

3 Approach

3.1 Dataset and Pre-Processing

The data Contains 24,783 Labelled tweets and three possible labels:
i) Hate Speech - 0
ii) Offensive Language - 1
iii) Neither - 2

It is first cleaned by using WordPunctTokenizer from tokenize package in nltk, Beautiful Soup. Then the tweets are analyzed for their length and number of words in each tweet.It was found that the maximum number of words in our dataset is 19. Word Cloud is used to generate some graphics of common words used in Hate Speech, offensive Language and other general words

3.2 Word-Embedding Learning

Initially, it was tried to train a word embedding particularly for the project and choose gensim package for training but the code took a long time to run to produce a embedding. After almost 10 hours of running the code it still didn't produce a result. Then it was decided to use a pre-trained embedding.
The pretrained word embedding was taken from the Global Vectors for Word Representation (Glove) which can be found in the webpage [Link1] in particular 'glove.6b.50d.txt'. This file from the Glove package was used for getting the word embedding for the general words used in tweets. It contains 50 dimension for each word. For the uncommon words which are not found in common dictionary random vectors were assigned. Finally there was a word embedding of dimension (17555 x 50) were 17555 comes from the total unique words in complete text and 50 is the dimension of each vector.

3.3 Experiments

Machine with 8gb ddr3 RAM was used and the project was completed on ubuntu 18 using Python3 language in a Jupyter Notebook almost 250 lines of code from the beginning was written for data pre-processing and word embedding learning. The frameworks included keras, tensor flow and scikit learn. References were taken from (4). All related code can be accessed in Link2.In the data pre-processing phase, we convert each word of a sentence into a vector of 50 dimension using glove.

The vaious layers in our experimental architecture were:
1. Embedding layer
2. Dropout layer (frequency- 0.25)
3. LSTM from keras
4. Dropout layer (frequency-0.5)
5. Sense layer
6. Activation layer - softmax
7. Compile layer- for loss and matrix calculation, optimization

We ran the training code for 8 hours after which there was a memory error. Then we tried to reducing the batch size and other pre-processing but due to some compilation errors we were not able to complete the code.

3.4 Final Approach - Feed Forward Network

We added the vectors for each word in a sentence and used them as input to a 8 layer Deep Feed Forward Network. In training we used validation factor as 0.25 and the loss function used was sparse categorical cross-entropy and optimizer used was stochastic gradient descent. After training the data for 200 epochs we found that the model was over fitting and hence we used 30 epochs. The accuracy obtained on test dataset came out to be 82.56%.

The layers in the final model used for training and the graphs of loss and accuracy are :

In [24]:

```
model = Sequential()
model.add(Dense(1000, input_dim=50, activation='relu'))
model.add(Dense(300, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(25, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(3, activation='softmax'))
sgd = keras.optimizers.SGD(lr=0.001, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='sparse_categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

Figure 1: Layers in the used model

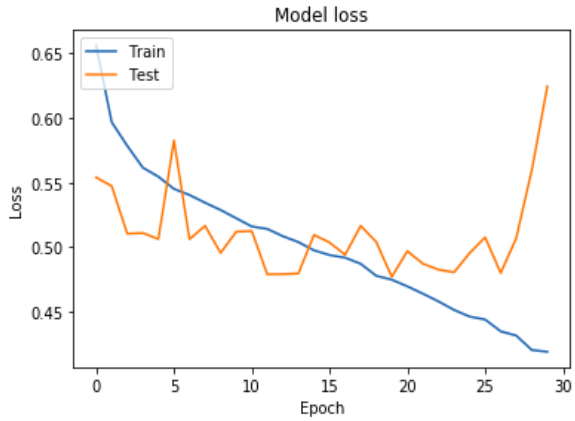

Figure 2: Loss vs iteration number

Figure 3: Accuracy vs iteration number

4 Result and Conclusion

Results obtained in the training process are as follows :
1. Minimum training loss : 0.4190
2. Minimum training Accuracy : 84.30%
3. Minimum validation loss : 0.4770
4. Minimum validation Accuracy : 82.30%

Test Accuracy on a separate data set was found to be 82.56%. Hence we were able to achieve a good accuracy using a basic feed forward network. In future we would explore Long Short Term Memory (LSTM) Network to known how does the prediction results change on this dataset .

5 Effort

Among other work done in the project, the most difficult part was generating the word embedding for the given text and it took the most time. Internal Work Distribution is as follows:
Harsh Maheshwari : Data pre-processing; Word Embedding; Model Architecture
Karthik Vignesh : Data Collection; Literature Survey; Analysis; Final Report
Kaushal Kishore : Model Architecture; Training and Debugging, Analysis and final report
Utkarsh Saraswat : Analysis and Final Report

References

[1] Z. Waseem and D. Hovy, "Hateful symbols or hateful people? predictive features for hate speech detection on twitter," in *Proceedings of the NAACL student research workshop*, 2016, pp. 88–93.

[2] P. Burnap and M. L. Williams, "Cyber hate speech on twitter: An application of machine classification and statistical modeling for policy and decision making," *Policy & Internet*, vol. 7, no. 2, pp. 223–242, 2015.

[3] P. Badjatiya, S. Gupta, M. Gupta, and V. Varma, "Deep learning for hate speech detection in tweets," in *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, 2017, pp. 759–760.

[4] Z. Zhang, D. Robinson, and J. Tepper, "Detecting hate speech on twitter using a convolution-gru based deep neural network," in *European Semantic Web Conference*. Springer, 2018, pp. 745–760.