Milestone: To Create a Chatbot for Detecting Hate Speech

Wesley Aldridge
waldridge@knights.ucf.edu

Miles C. Crowe
miles.crowe@knights.ucf.edu

Mauricio De Abreu
mabreu@knights.ucf.edu

1. MOTIVATION & PROBLEM STATEMENT

Hate speech is defined broadly as aggressive language targeted at a person for attributes typically beyond their control. Traits such as nationality, gender, race, sexual preference or disability are popular targets of hate speech[1]. Aggression towards the victim may manifest as insults, personal attacks or even threats. As online communication between people is becoming more commonplace, exposure to hate speech is almost certain to reach more people. Unfortunate as it is, this allows the spread of underlying beliefs that contribute to such behavior.

The motivation for this project is rooted in the desire to curtail the spread of hateful language. There are countless paths for hate speech to spread. Providing a human intervention is infeasible given the amount of communication that occurs over the internet. Naturally, this presents an excellent opportunity to contribute an automated approach to detecting and flagging hate speech by utilizing NLP in real time.

2. RELATED WORK

Hate speech has earned enough interest to inspire governments into getting involved[2] by passing laws explicitly prohibiting hate speech. Given this, many have examined various methods for hate speech detection.

Manoel Horta Ribeiro, et al.[3][4] used a Twitter hate speech dataset to characterize Twitter users as either hateful or not, in order to analyze patterns and trends among the users labeled as hateful. Instead we will use this dataset, among others, to train a neural network to detect hate speech, and we will use this neural network to create a Discord realtime hate speech detection bot.

Davidson et al. went further into separating offensive speech from hate speech. Their approach used hate speech compiled by Hatebase.org, consisting of a lexicon of both words and phrases, identified by internet users. Following their lead, we intend to use Hatebase.org's academia API to obtain modern hate speech examples. This is important as a major shortcoming noted by MacAveney et al.[5] is those who spread hate speech are well aware that their views and texts are being suppressed and removed, and continue to evolve to evade detection.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Our work distinguish from the previous because we focus on online binary classification of hate speech tweaked for the discord platform.

3. STOPPING THE SPREAD

This project aims to curtail the spread of such speech by detecting hateful dialog and providing an alert mechanism to raise events for handling the occurrence automatically as it happens. Clearly stated, the chat bot would be a silent participant in a chat, regardless of the number of participants, which would monitor the text exchanges. When hate speech is detected, the event would be passed to a host framework or external agent to handle the event.

A simple implementation of this would be a chat bot, connected to a Discord server which is trained utilizing a robust hate speech data set. A neural network would be trained to detect hate speech. This neural network would be the critical part of the chat bot which would read user messages and detect hate speech in real time, and either alert the moderators and admins of the Discord server or (and more effectively) actively kick out hate speech users from the monitored channels.

4. APPROACH

Our solution is a binary classifier that predicts whether each discord message is hate speech or not. The classifier was trained through supervised learning using publicly available labeled Twitter data that from Davidson et al. work [2] We expect the classifier trained with Twitter data will be able to accurately classify discord messages as their content although have different capabilities in average are similar in length and language verbiage. On this section we describe our approach in details by explaining each component of the machine learning pipeline as follow.

4.1 Training and Test Data

We used labeled data from Davidson et al. work [2]. The work include details on that data set was built we summarizes here some key points. They began by using a lexicon compiled by Hatebase.org [?]. That lexicon contain words and sentences that have been classified as hate speech by internet users. Then they used Twitter API to look for tweets that contain terms from the lexicon. That search resulted in collection of tweets from about 33,000 different users. They downloaded all tweets from those users' timelines (around 85 million tweets) and randomly selected 25,000 tweets containing terms from the lexicon. Those 25k tweeds were classified by human workers. They classified each tweet as hate speech or not hate speech and also whether offensive or not offensive which is not currently relevant for the scope of our work. Workers were given specific instructions on how to classify including a explanation that only the presence of a particular offensive word is not necessarily a indication of hate speech and also they were provided with not only the tweets but also the context tweets around it. Each tweet was classified by at least 3 workers and they kept in the data set only the tweets in which the majority of the workers decided for the same class and discarded those with no majority decision. That process resulted in a labeled data set of 24,802. Starting from their data set, as our goal is slightly different from them, and for the sake of having more relevant data for our purpose, we considered hate speech the tweets which any worker classified as hate speech as opposed as the majority of the workers. From the total labeled data set of about 25K tweets we used 20,000 as our training set and reserved around 5,000 as our validation data set.

4.2 Pre-processing

The dataset we used labelled Tweets on a scale from 0 to 7, with 0 representing that the seven original researchers unanimously agreed the Tweet was not hate speech, up to 7 representing that all seven researchers agreed that a Tweet was hate speech. We reclassified the data to represent a binary distinction, either hate speech or not hate speech, with anything originally labeled higher than a 0 as hate speech (1), and anything originally labeled as a 0 remained a 0. This left us with 4993 Tweets labelled as hate speech and 19790 Tweets labelled as not hate speech. We then transformed all of the text in the Tweets to lowercase to avoid redundancy in vocabulary based on capitalization. Similarly we removed all punctuation and stemmed the data. The stemmer reduces key words to the same stem without removing relevant information, for instance *days* and *day* stem to *day*. We also padded the Tweets with the character "0" to keep them all the same length. 0 was chosen because it is a neutral word/character that has no known affiliation with hateful speech. We identified 40,817 distinct words in the Tweets (before stemmer) and used 50k as our vocabulary size for our first model.

After the stemmer 36,280 distinct words were identified. So for our second model we set our vocabulary size as 36,300 for generating the one-hot vectors.

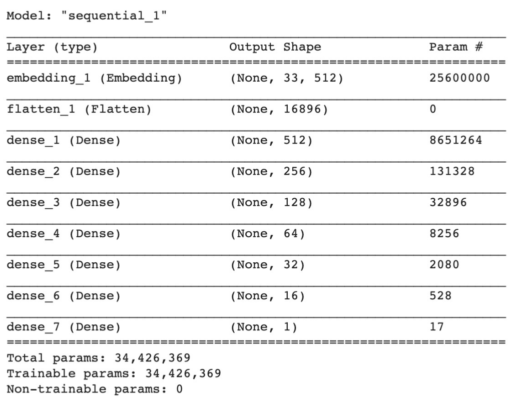
The maximum input size found on the input data set was 33 words and so we padded all inputs to size 33.

4.3 Embeddings

For our first ANN model we used a dense embeddings layer to get 512 dimensional dense vectors from each tweet input value (original dimensionality was 33 (word length) x 50,000 (dimension for one-hot vector for the vocabulary size).

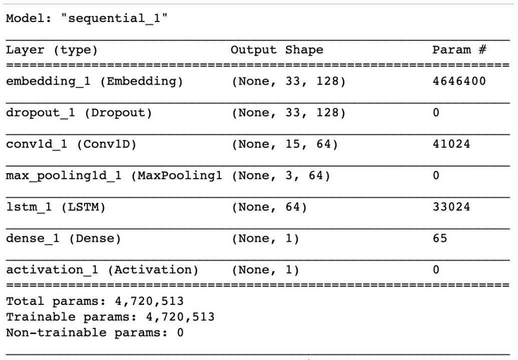
For the second model we used 128 as the embeddings dimension. So the embedding layer reduced the dimensionality of the input considerably from the 36,800 original dimensional for one-hot vectors.

4.4 Network Architecture



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 33, 512)           25600000
_________________________________________________________________
flatten_1 (Flatten)          (None, 16896)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               8651264
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_6 (Dense)              (None, 16)                528
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 17
=================================================================
Total params: 34,426,369
Trainable params: 34,426,369
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Architecture 1 (ANN)



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 33, 128)           4646400
_________________________________________________________________
dropout_1 (Dropout)          (None, 33, 128)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 15, 64)            41024
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 3, 64)             0
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                33024
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 4,720,513
Trainable params: 4,720,513
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Architecture 2 (CNN and LSTM)

Our first classifier attempt which is depicted in figure 1, is a seven layer deep ANN with decreasing number of nodes for each subsequent layer starting from 512 nodes on the first dense layer and then reducing to 256, 128, 64, 32, 16 and finally 1 node for the output layer. The six first layers uses the relu activation function while the output layer uses the sigmoid activation function.

Our second approach uses CNN and LSTM. Its summarized representation is shown in figure 2. The full architecture includes a dropout layer to address over-fitting, followed by a 1 dimensional CNN block (conv 1d and max pool) for extracting features from the tweets, followed by a LSTM and scoring layer block for the classifier. The output layer uses the sigmoid activation function. Our first classifier attempt was a seven layer deep ANN with decreasing number of nodes for each subsequent layer starting from 512 nodes on the first dense layer and then reducing to 256, 128, 64, 32, 16 and finally 1 node for the output layer. The six first layers uses the relu activation function while the output layer uses the sigmoid activation function. Our second approach uses CNN and LSTM. The full architecture includes a dropout layer to address over-fitting, followed by a 1 dimensional CNN block (conv 1d and max pool) for extracting features from the tweets, followed by a LSTM and scoring layer block for the classifier. The output layer uses the sigmoid activation function. Current testing accuracy with second approach is 80.95%. We will test a third approach based on RNNs for the classifier and will be appropriate for any size of input.

4.5 Optimizing

We trained our network optimizing for accuracy using the binary cross-entropy loss function through 10 epochs with batch size of 32 points for the first network architecture. The resulting accuracy on the test data was 75%. For the CNN and LSTM classifier architecture we used only 2 epochs for preventing over-fitting, and we received 80.95% accuracy on the test data.

5. EXPECTED OUTCOMES AND RISK MANAGEMENT

As in most software projects, expectations will need to be curtailed and priorities need to be established. At the bare minimum, this project should yield a simple input/output text classifier that is accurate with meeting the expectation of identifying hate speech. Formally stated, this implies that core functionality should be established prior to moving towards the novel aspects such as portability and integration with external platforms. This can be realized though a simple design that exposes only the necessary API to validate and test the outcome of the core goal. Once this goal is achieved, a narrow set of integration targets can be established for demonstration purposes.

In terms of the main goal, there is risk of biases and over fitting of the training data. Over-fitting errors will need to be carefully analyzed and bias will need to be mitigated with tools we will discover later in this course.

Another risk would be training data mismatch with regards to the platform. For example, users in Discord are allowed 2000 characters for a message versus 280 characters for a Tweet on Twitter. Size constraints may dramatically affect how hate speech is formulated due to the compression of ideas on platforms that severely restrict text lengths. Careful validation can easily address this by restricting the test data to mimic similar sizes to that of the training data.

Lastly, temporal changes in hate speech may present classification errors as slang and dialect may change over time. This risk will have to be accepted unless fresh and up to date training data becomes immediately available.

6. PLAN AND ROLES OF COLLABORATORS

Wesley will code the neural network, train it with the hate speech data, and test it. The neural network will be what the bot uses to label messages as hate speech or not hate speech.

Miles will integrate the neural network into an online hate speech detection service that he will code and test. The service will be used by the Discord bot to do its message analysis and hate speech detection.

Mauricio will incorporate the online hate speech detection service into a Discord server bot that he will code and test. The bot will detect hate speech in Discord server messages in real time.

Everyone will work on the write-up and presentation.

For the first three weeks, the main focus was on gathering training data, designing and coding the framework and resolving architectural challenges. The next four weeks will consist of building the neural net, training, testing and optimization. The final three weeks, or remaining time to completion will focus on integration with chat bot client, analysis of hate speech detection and writing of the report and presentation.

7. PROGRESS AS OF 3/5/2020

Wesley has accomplished preliminary work on building an initial neural net and some training. Results from the neural net have been inconclusive as of yet. This is due to the classification of the training data. The training data is not classified into HATE/NOT HATE, rather is it scaled from normal speech through various levels of offensive language to various levels of hate speech.

Mauricio has performed a training task using the same data set but with a network structure based on a 1 dimensional convolutional network block followed by a LSTM block. That network outperformed the first vanilla ANN architecture we have tested.

Miles has downloaded the entire data set of hate speech vocabulary from hatebase.org. This vocabulary can be used to annotate speech as containing hate speech indicators. Miles also created a Python based Discord bot that is a member of our group discord. Design of the API therein to mesh the Python bot to Wesley's classification system is underway.

Mauricio has created a NodeJS implementation of a Discord bot in parallel to the work that Miles has accomplished. Mauricio was able to complete some rudimentary behaviors such as warning, kicking, etc. This affords the project flexibility if we want to switch to a NodeJS implementation.

All team members participated on discussions about how the work would progress including brainstorms around how we would design our classifier architecture, and how we would test and evaluate it, and the best selection of annotated data to use, etc, Also all team members have participated on writing out this report.

Challenges moving forward will always be time availability and lack of training data. Each member will continue to gather more examples of Hate Speech so that a more robust classifier will ultimately be created.

8. REFERENCES

[1] "Dictionary.com." https://www.dictionary.com/browse/hate-speech. Accessed: 2020-01-21.

[2] M. Davidson, Warmsley and Weber, "Automated hate speech detection and the problem of offensive language," *arXiv:1703.04009v1*, 2017.

[3] "Hateful users on twitter." https://github.com/manoelhortaribeiro/HatefulUsersTwitter. Accessed: 2020-01-23.

[4] M. H. Ribeiro, P. H. Calais, Y. A. Santos, V. A. Almeida, and W. Meira Jr, "'like sheep among wolves': Characterizing hateful users on twitter," *arXiv preprint arXiv:1801.00317*, 2017.

[5] Y. R. G. F. MacAvaney, Yao, "Hate speech detection: Challenges and solutions," *PLOS|ONE*, 2019.