Hate Speech Detection in Text

Abishek Shyamsunder
NYU, Courant
ats9767@nyu.edu

Muthukrishna Krishnakumar
NYU, Courant
mk7516@nyu.edu

Samvid Zare
NYU, Courant
sz3369@nyu.edu

Abstract

The exponentially increasing usage of social media has its drawbacks such as the spread of hate speech at a large scale. To prevent the spread of hate speech and its impact in real life such content has to be detected at early stages. Moreover, implementing regulatory measures for such tasks needs automation to handle massive amounts of data. One of the key challenges in hate speech detection is a differentiation between hate speech and merely offensive sentences due to similar words in both types of sentences. In this paper, we implemented the 5 states of art models on the publicly available textual datasets to determine the most effective method for hate speech detection. The models were compared in terms of accuracy and other measurable performance metrics. Further, this paper concludes that the transfer learning method and support vector machine method achieve better results on the larger datasets compared to other methods.

1 Introduction

The hike in the usage of Social media in recent years has impacted the lives of many. The proportion of social media users has increased by 5% from 2019 to 2020 worldwide (Statista, 2021). As an example, Reddit reported 52 million active daily users in 2020 (Reddit.com, 2020). This proliferation has enhanced the ease of communication and information sharing while at the same time increasing the hate campaigns. If such hate campaigns are not prohibited can lead to sexual predation and negative mental and physical consequences.

For the scope of this paper, we will consider hate speech as a speech that can express prejudice against a person or a group based on factors such as race, color, ethnicity, gender, sexual orientation, nationality, religion, etc. (Nockleby, 2020). Although there is a hierarchical classification of hate speech based on the intensity, for the scope of this paper, any speech that doesn't qualify as hate speech will be considered as an ordinary speech (Brown, 2017).

The key challenge is to differentiate between merely offensive sentences and hate speeches as both types will have similar words most of the time. Along with the analysis of this problem, this paper aims to deliver an end-to-end overview of the hate speech classification problem and showcase the performance comparisons between various machine learning models used as solutions in section 4. Based on the model's ability to handle offensive speeches that are not Hate speeches, further fine tuning operations were done to enhance the efficiency. Finally, we discuss different ways of improving the performances and additional data requirements for the task in section 5.

2 Corpus

The Data used for this problem was the Reddit dataset, sourced and made publicly available as mentioned in the paper (Qian et al., 2019). Each entry in the dataset was a Reddit conversation segment, and a corresponding list of indices of which posts in the segment were classified as Hate Speech.

Each segment consisted of a single post and a set of related comments. Each of the segments were broken down into individual posts with a corresponding binary indicator of whether the it was hate speech or not. This gave a corpus of 16,000 posts of which 5000 were classified as Hate speech.

The data was then further cleaned by removing stop-words and punctuation, and performing stemming. Finally to offset the skew between the number of positive and negative labels, we randomly sampled from the negative class, the same number of records as present in the positive class. Finally, this was divided into two sets, for training and testing.

3 Models

3.1 Naive Bayes

Before testing the performance of the Naive Bayes model on the data, further processing was performed. The TF-IDF vectors were calculated for each of the entries. These provided a better association between the words. This output was then pipe-lined to serve as the input to the classifier.

The model gave an accuracy of $\mathbf{73}\%$ and an F1

Score of $\mathbf{0.70}$. Clearly the property of the Naive Bayes to assume conditional independence of the probabilities of words given a class resulted in the accuracy and F1 scores taking a hit. However, the model gave an insight into the challenges posed by the dataset and served as a baseline for other models to beat.

3.2 Logistic Regression

To bring the data to a suitable form so that it can be fed as input to a logistic regression model further processing of data was performed. Apart from the negative sampling that was performed for Naive Bayes Model, all text in the dataset within square brackets, parentheses were removed since most of these text does not add meaning to the existing sentence. Then numbers, extra whitespaces, quote characters, '&' character, reddit usernames, hashtags, were also removed from the dataset. The data was then tokenized and stop words were removed. Lemmatization was performed on the data. All these pre processing steps were carried out to achieve better results. The TF-IDF vectors were calculated for each of the entries. The output was then fed as input to the Logistic Regression Model.

The model gave an accuracy of $\mathbf{86.9}\%$ and an F1 Score of $\mathbf{0.86}$. The drastic increase in terms of performance when compared to Naive Bayes Model can be attributed to the property of the discriminative Logistic Regression Model that it does not make the same assumptions which were made by the Naive Bayes Model.

3.3 Neural Networks

Given the hugely successful impact of various neural network architectures in the field of NLP, different Neural models were applied to the classification task.

Despite being simple models with not more that three layers, each of the models mentioned below have thousands of trainable parameters. This brings about a high probability of over-fitting on the training data, resulting in a drop of testing accuracy.

To circumvent this problem, the training data was further divided into two parts (in a ratio of 80:20), one for the actual training of the model and the other to serve as a validation set respectively. Each of the models were then trained with early callbacks to stop training when the validation loss drops.

The data was first tokenized at a word level, and converted to sequences of integers, before being fed into the different neural models. Each of the models mentioned below had an embedding layer to extract information from these vectors.

3.3.1 Simple Neural network

Layer (type)	Output Shape	Param
Embedding	(None, 120, 16)	225024
Flatten	(None, 1920)	0
Dense	(None, 10)	19210
Dense	(None, 1)	11

Table 1: Simple Neural network structure

A simple 3 layered neural network (excluding the flatten layer) was created from scratch (with architecture as can be seen in Table 1).

The Simple Neural network gave an accuracy of $\mathbf{84.5}\%$ and an F1 score of $\mathbf{0.83}$. Although this beat the baseline Naive Bayes model, its performance was far behind that of the logistic regression classifier and the Support Vector Machine model.

3.3.2 Recurrent Neural network

To take advantage of the temporal nature of textual data, a 3 layered neural network containing an embedding layer, Simple RNN layer and output layer was used. This, gave a performance better than the simple Neural network model with an accuracy of $\mathbf{87}$ and F1 score of $\mathbf{0.862}$

Layer (type)	Output Shape	Param
Embedding	(None, 120, 16)	225024
SimpleRNN	(None, 6)	138
Dense	(None, 1)	7

Table 2: Recurrent Neural network structure

3.3.3 LSTM

Based on the work done by Sepp Hochreiter and Jürgen Schmidhuber (Hochreiter and Schmidhuber, 1997), to help retain information longer, the Simple RNN layer as mentioned in section 3.3.2 was replaced by a layer of LSTM cells and as expected, the accuracy increased to $\mathbf{87.9}$ and F1 score increased to $\mathbf{0.876}$

Layer (type)	Output Shape	Param
Embedding	(None, 120, 16)	225024
LSTM	(None, 6)	552
Dense	(None, 1)	7

Table 3: LSTM Neural network structure

3.3.4 Gated Recurrent Units

Next, an architecture where a layer of Gated Recurrent Units replaced the layer of LSTM units (of section 3.3.3) was tested. Having lesser number of parameters to train (than the LSTM layer of with the same number of nodes), and having input sentences that were not very long (because they were Reddit posts), this architecture performed better than the LSTM model mentioned in section 3.3.3 giving an accuracy of $\mathbf{88.3}\%$ and a F1 score of $\mathbf{0.87}$

Layer (type)	Output Shape	Param
Embedding	(None, 120, 16)	225024
GRU	(None, 6)	432
Dense	(None, 1)	7

Table 4: GRU Neural network structure

3.3.5 Bi-Directional LSTM

To try and extract context from both directions of the input, a Bi-Directional LSTM layer replaced the LSTM layer in the architecture mentioned in section 3.3.3. This did increase the accuracy to $\mathbf{88.5}\%$ and a F1 score to $\mathbf{0.88}$. However, the SVM model still outperformed this architecture.

Layer (type)	Output Shape	Param
Embedding	(None, 120, 16)	225024
Bidirectional	(None, 12)	1104
Dense	(None, 1)	13

Table 5: Bi-Directional LSTM Neural network structure

3.4 Transfer Learning Using GloVe

Given the widely successful vector space with meaningful substructure obtained by the GloVe vector representation of text, as shown in the work (Pennington et al., 2014). The GloVe (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors) data was used to initialise the embedding layer weights of a neural network.

This was done by first, reshaping the tokenised sequences of the input to match the shape of the vector embeddings, filtering out words from the vector embeddings that did not occur in our corpus and then creating a embedding matrix.

This was then used to initialize the weights of the embedding layers in the architecture given in table 6

Layer (type)	Output Shape	Param
Embedding	(None, 300, 300)	5300100
GRU	(None, 6)	5544
Dense	(None, 1)	7

Table 6: Transfer Learning model Structure

This increased the maximum accuracy obtained on the dataset to $\mathbf{88.7}\%$ and also provided an F1 score of $\mathbf{0.881}$

3.5 SVM

Support vector machine models are preferred for sentiment analysis due to their ability to perform well on the sparse input and test data with large features such as n-grams (Cortes, 1995). The sentences from the dataset were cleaned using standard pre-processing techniques as mentioned in section 2 followed by feature extraction based on TF-IDF method. The TF-IDF method is mainly used to penalize more frequent words which can lead to excessive noise in SVM model designing.

The support vector machine attempts to fit the hyperplane in the training data with the objective of maximizing the relative distance of the labeled classes from the hyperplane. The same can be expressed in form of the dual optimization problem as given in (Bottou and Lin, 2007),

$$max(W(\alpha)) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (\mathbf{\Phi}(x_i)\mathbf{\Phi}(x_j))$$

$$\text{such that} \quad \forall i \;\; \alpha_i \geq 0 \text{ and } \sum \alpha_i \cdot y_i = 0$$

where (x_i, y_i) is i^{th} input data , $\alpha_i s$ are supported vectors and Φ is a feature extractor function

L2 regularization was used to avoid over-fitting. Due to the linearity of the data, a linear kernel was selected for SVM. Further hyper-parameter tuning was performed to find the optimal regularization parameter C and kernel scaling factor gamma. The trained model with tuned parameters achieved an accuracy of $\mathbf{88.6}\%$ and an f1-score of $\mathbf{0.88}$ on the test data. The SVM was found to perform far better than the baseline model as well as computationally easy to design.

4 Results and Conclusion

From Table 7, we can infer that Transfer Learning model performs the best. However, The difference

Model	Metrics	
Naive Bayes	**Acc**: 73.7	**F1**: 0.707
	Recall: 0.640	**Prec**: 0.791
Logistic Regression	**Acc**: 86.9	**F1**: 0.869
	Recall: 0.851	**Prec**: 0.883
SVM	**Acc**: 88.6	**F1**: 0.880
	Recall: 0.827	**Prec**: 0.930
Simple NN	**Acc**: 84.5	**F1**: 0.832
	Recall: 0.770	**Prec**: 0.904
RNN	**Acc**: 87.0	**F1**: 0.862
	Recall: 0.820	**Prec**: 0.905
LSTM	**Acc**: 87.9	**F1**: 0.876
	Recall: 0.860	**Prec**: 0.893
GRU	**Acc**: 88.3	**F1**: 0.877
	Recall: 0.833	**Prec**: 0.925
BiLSTM	**Acc**: 88.5	**F1**: 0.880
	Recall: 0.849	**Prec**: 0.913
Transfer Learning	**Acc**: 88.7	**F1**: 0.881
	Recall: 0.840	**Prec**: 0.927

Table 7: Results Summary

in performance between SVM and Transfer Learning is very small which shows that SVM is relevant among other advanced models given the right data.

With respect to the various Neural network architectures implemented, there was a clear and gradual increase in performance obtained that matched expectations, with the accuracies and F1 scores of models adhering to Bi-LSTM > GRU > LSTM > RNN > Simple Neural Network.

Given that the challenges of the reddit dataset presented in (Qian et al., 2019) such as the distribution of hate and non-hate speech labels is more unbalanced and ambiguity in classifying offensive words still persist, our models have surpassed models presented in the aforementioned paper in terms of both Accuracy and F1 Score.

5 Related work and Future scope

The work done by (Kwok and Wang, 2013) shows that the presence of offensive words can lead to the misclassification of tweets as hate speech, where 86% of the time the reason a tweet was categorized as racist was because it contained offensive words

As mentioned in (Davidson et al., 2017), hate speech and offensive language can be well separated if the additional labeling about whether a given sentence is offensive or not is available.

Currently Regex based filters are used to detect hate-speech, however the work (Nobata et al., 2016) shows how this method fails to recognise more subtle forms of Hate Speech

Additionally, to improve the performance with the existing data, we can use linguistic knowledge-based features to get a better classification between offensive language and hate speech (Schmidt and Wiegand, 2017).

References

Léon Bottou and Chih-Jen Lin. 2007. *Support Vector Machine Solvers*, pages 301–320.

Alexander Brown. 2017. What is hate speech? part 1: The myth of hate. *Law and Philosophy*, 36.

Vapnik Vladimir Cortes, Corinna. 1995. Support-vector networks. *Machine Learning*, pages 1573–0565.

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Irene Kwok and Yuzhou Wang. 2013. Locate the hate: Detecting tweets against blacks. In *Twenty-seventh AAAI conference on artificial intelligence*.

Chikashi Nobata, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang. 2016. Abusive language detection in online user content. In *Proceedings of the 25th international conference on world wide web*, pages 145–153.

John T. Nockleby. 2020. Hate speech. *Encyclopedia of the American Constitution (2nd ed., edited by Leonard W. Levy, Kenneth L. Karst et al*, page 1277–1279.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Jing Qian, Anna Bethke, Yinyin Liu, Elizabeth Belding, and William Yang Wang. 2019. A benchmark dataset for learning to intervene in online hate speech. *arXiv preprint arXiv:1909.04251*.

Reddit.com. 2020. Reddit's 2020 year in review.

Anna Schmidt and Michael Wiegand. 2017. A survey on hate speech detection using natural language processing. In *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*, pages 1–10, Valencia, Spain. Association for Computational Linguistics.

Statista. 2021. Number of social network users worldwide from 2017 to 2025.