Multiclass Hate Speech Detection

Mudit Gupta 2020315
Diksha Sethi 2020056
Niranjan Sundararajan 2020090
Vibhu Dubey 2020150

Abstract

Hateful and toxic content generated by a large portion of users on Social Media Platforms (SMP) has motivated several researchers to dedicate substantial efforts to the challenging task of hate speech detection. The ability to distinguish hate speech from other offensive language is a significant obstacle for automatic hate-speech detection on social media. In this paper, we use the H3 Multiclass Hate Speech Detection dataset. The aim of the task is to perform 3-way classification to predict the label of the given text to be one of these classes – Hate, Offensive, Neither. We have experimented with various ML and DL models ranging from the simplest traditional method to advanced transformer based methods for the task, including Linear Regression, BERT, etc.

1 Introduction

Social networking sites like Instagram, Twitter, Facebook, YouTube, and others are increasingly being used by people to exchange information and express their ideas. Although user interactions on these platforms can result in positive discussions, they are also being used to spread hate and plan hate-based events, mainly because of the accessibility and anonymity of these online platforms. Hate generally is subjective, temporal, and cultural in nature. The United Nations defines hate speech as any kind of communication in speech, writing, or behavior that attacks or uses pejorative or discriminatory language with reference to a person or a group based on who they are, in other words, based on their religion, ethnicity, nationality, race, color, descent, gender or other identity factors. This is often rooted in and generates intolerance and hatred and, in specific contexts, can be demeaning and divisive. " Hate speech threatens democratic principles, social harmony, and peace. Eliminating violence against women and other significant human rights violations and promoting peaceful, inclusive, and just societies all depend on combating hate speech.

1.1 Problem Definition

In this paper, we use the label tweets into three categories: hate speech, offensive language, or neither. We train a model to differentiate between these categories and then analyze the results to understand better how we can distinguish between them.

2 Related Work

There have been multiple studies and work done on Hatespeech detection as well as Muticlass Hatespeech detection in the past. We have mentioned a few which we feel were the most significant along with their observations and ideologies. Initially, traditional machine learning models were used and reported decent results. However, with the introduction of multiple word embedding techniques capable of remembering the context,more advanced methods were proposed, including transfer learning and transformer models.

The primary factors separating the various approaches in classical machine learning were the features utilised, and it was discovered that surface-level features like bag of words, word-level and character-level n-grams, etc., were the most predictive features. Waseem et al. utilised a logistic regression model to categorise tweets and found character n-grams to be the most suggestive features. Other factors included word and character n-grams up to four, gender, length, and location. When Davidson et al. tested different multi-class classifiers, they discovered that the Logistic Regression with L2 regularisation outperformed them all. Automated classification methods helped to achieve relatively high accuracy at distinguishing between classes. Further, from the results obtained it was concluded that the presence or absence of hateful or offensive terms can both help and hinder classifying the samples accurately. Transfer learn-

ing utilized pre-trained word vector representations that were extracted from a sizable amount of text data. With the help of notable innovations like Universal Language Model Fine-Tuning (ULMFiT), Embedding from Language Models (ELMO), OpenAI's Generative Pre-trained Transformer (GPT), and Google's BERT model, the year 2018 marked a turning point for several NLP tasks. By pre-training a universal language model on a general-domain corpus and then fine-tuning the model on target task data via discriminative fine-tuning, Howard et al. created ULMFiT, which may be used to any NLP task. Both OpenAI GPT and BERT, two transformer-based language models, were created by Radford et al. and Devlin et al. While BERT is the first deeply bidirectional, unsupervised language representation, pre-trained using only a plain text corpus, OpenAI GPT is a unidirectional language model. In a number of downstream NLP tasks, the pre-trained BERT model greatly beat ELMo and OpenAI GPT. However, as most of other attention-based deep neural networks, BERT mainly focused on local consecutive word sequences, which provided local context information. However, it might be difficult for BERT to account for the global information of a language. This gave rise to use of Graph Convolutional Networks (GCN) for text classification. In GCN, global relations between words in a language can be represented as a graph, in which words are nodes and edges are relations. A number of variants of GNN have been proposed and applied to text classification tasks of which Kipf 4 Lu et al. et al. creatively presented Graph Convolutional networks (GCN) based on spectral graph theory. Text GCN is a special case of GCN for text classification proposed by Yao et al. recently. GCN and its variants are good at convolving the global information in the graph into a sentence, but they do not take into account local information such as the order between words. When word order and other local information are important, GCN may be insufficient. Lu et al. proposed a combination of Vocabulary GCN (VGCN) and BERT. They introduced a vocabulary graph which helped capture this global relation while also remembering the local information. Instead of using only word embeddings of the input sentence in BERT, we feed both the vocabulary graph embedding and the sequence of word embeddings to BERT transformer. This way, not only the order of the words in the sentence is retained, but also the background information obtained by VGCN is utilized.

As observed from previous studies, the major challenge lies in identifying the difference between offensive and hatespeech. Kwok and Wang's study on anti-black racism reveals that offensive terms were present in tweets 86 of the time, which is why they were labelled as racist. Hate speech detection is particularly difficult due to the comparatively high prevalence of swear words and "curse words" on social media (Wang et al. 2014). Subtle linguistic distinctions are frequently used to distinguish between hate speech and other offensive words. For instance, tweets using the word n*gger are more likely to be classified as hate speech than tweets containing the word n*gga (Kwok and Wang 2013). Although earlier research has used training data having a broad definition of hate speech, neural language models show promise in the task (Djuric et al. 2015)

3 Methodology

3.1 Dataset Analysis

There are 4,957 tweets in the test set and 19,826 tweets in the training set of the H3 Multiclass Hate Speech Detection dataset. It is provided as a part of a Kaggle challenge. We divided the provided training set into a train and test set to compare the results using metrics, as the test set didn't have any labels. Therefore, we limited the scope of our data analysis to the training set of 19,826 tweets. The dataset consists of the following fields -

- id: Each tweet has a unique id.
- tweet: English text representation of the tweet
- label: This class label contains three values. If a tweet belongs to hate speech, it has label 0, if it belongs to offensive language, it has label 1. And if it belongs to neither, then it has label 2.

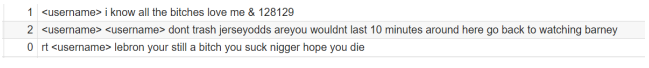

Figure 1: Instances of hate speech, offensive language and neither

Out of the 19,826 tweets, 1,137 (5.73%) have a label 0, 15,398 (77.66%) have a label 1, and 3,291 (16.59%) have a label 2 as described in Fig 2.

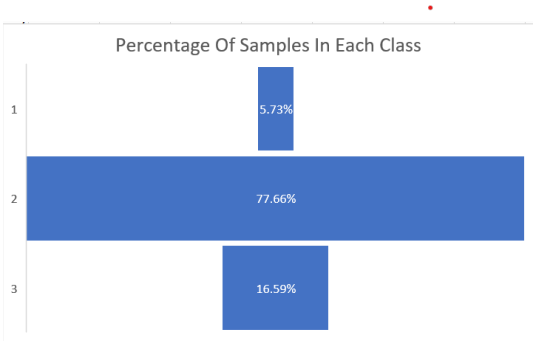

Figure 2: Proportion of data samples in each class

3.2 Data Preproceessing

The first and most crucial step in resolving any NLP problem is preprocessing the text data. We performed a few preprocessing steps to ensure that all the text inputs were uniform. This was done to make sure that any of these textual inconsistencies wouldn't affect the output in any way. The following steps were performed in the order in which they are listed:

1. Lower Case: To ensure that words with different case styles are not perceived in different contexts, we have lowercase each term in our dataset
2. Removal of URLs
3. Removal of usernames
4. Punctuation Removal
5. Tokenization: We have broken down the text sentences into smaller chunks or units using the Tweet Tokenizer. This step would essentially help to understand the meaning of the text by analyzing smaller units at a time.

Several words, such as usernames and URLs, and special characters, such as punctuation marks, have been removed because we do not require them in our context.

3.3 Dataset Augmentation

We realised that the bias introduced in the results towards the offensive language could be because of the highly unbalanced dataset. As observed from above, the hate speech and neither categories has very few samples in comparison to the offensive category. So, we generated samples for these two categories, and created a dataset of about 41K samples with the following distribution:

Class Label	Number of Samples
{0}	12507
{1}	15398
{2}	13164

For generating the samples, we used a number of techniques such as synonym replacement, random swap and random deletion. While generating the samples, we also took care of the length of the sample, and set the minimum length to 15, and maximum to 70. However, due to the large nature of this generated dataset, and the high dependence on the augmented data, we decided to append only a few of the generated samples in our original dataset for the experiments.

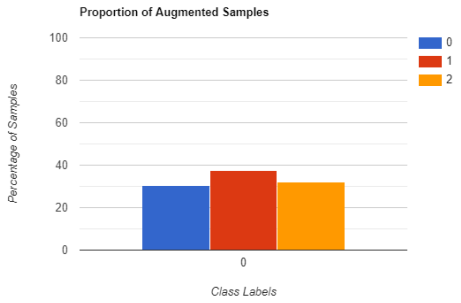

Figure 3: Proportion of data samples in each class

3.4 Classifier Architectures

Following the observations from all the above-mentioned literature, we decided to base our models on the BERT base classifier. The BERT base classifier is a BERT model consisting of 12 layers of Transformer encoder, 12 attention heads, 768 hidden size, and 110M parameters. it is pre-trained by utilizing the bidirectional nature of the encoder stacks. This means that BERT learns information from a sequence of words not only from left to right, but also from right to left. It helps give context for our natural language sentences. It outputs an embedding vector of size 768 for each of the tokens. Further, we use the categorical cross entropy loss as our loss function because our task involves multi-class classification. In each of the architectures, we use the model that gives us the best results on the validation set. This result could be achieved after any number of epochs.

3.4.1 SentenceBert Embedding + Logistic Regression

We encoded the textual sentences into vectors by using SentenceBERT. Based on the Davidson et Al 2017, we used a one-versus-rest framework. Essentially, we trained a separate classifier for each class: hate, offensive, and neither. The final label that was assigned to a sentence was the one that had the highest probability across all classifiers.

3.4.2 Decision Tree Classifier

The decision tree classifier creates the classification model by building a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes.Each node in the tree specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute. We implemented this tree with varying max depths belonging to [10, 50, 100, 1000].

3.4.3 Bert Classifier + Dense layer

One of the main objectives of BERT include Masked Language Modelling (MLM), in which we get append the [CLS] token and store the input's context in it along with getting the embeddings for the inputs. In this model, we focus our attention on the embedding vector output from the special [CLS] token. We use the embedding vector of size 768 as input for our classifier and pass it to the final layer containing 3 neurons, corresponding to our 3 classes respectively. We use softmax activation function at the output layer to get the final prediction. We train the model for three epochs and use Adam as the optimizer

3.4.4 Bert Pool Classifier + Average Pooling + Dense Layer

In this model, we utilize all the other embedding vectors except for the [CLS] token one. Since our max sequence length is 512, the output dimension from the BERT base is 512 x 768. To reduce the dimension of the 2D matrix while also maintaining the context and meaning, we apply average pooling to the output and bring it down in dimensionality. Finally, this vector is flattened and passed to the final dense layer containing three neurons to perform the classification task using softmax activation. We trained the model for 5 epochs, with a learning rate of 1e-3 for the dense layer and 1e-5 for the BERT model. Further, we used ReLU as the activation function.

3.4.5 Bert Pool Lin Classifier + Average pooling + 2 Dense Layers

In the previous architecture, we realized that there was a huge difference between the number of input features and output features in the final dense layer. To mitigate this, we decided to introduce an additional layer in between to reduce the loss of information when passing the data from one layer to another. So another dense layer with input dimension 100 and output dimension 3 was connected to the final dense layer with input dimension 768 and output dimension 100. This resulted in the architecture following 768 -> 100 -> 3.

3.4.6 Bert Ensemble Classifier

In this model, three separate BERT models were trained for each individual class a binary classifier. All three BERT models followed the same architecture as 3.4.5. We then concatenated the results from all the models to get the model ensemble for classification.

We tried the same model structures again, but this time used RoBERTA for our sentence embedding task. However, the results were slightly worse compared to BERT. Thus, we will not go into details regarding its methodology.

Further, we performed the same experiments on the augmented data to see if a much more balanced dataset helps in better classification.

4 Evaluation Metrics

We used Macro F1 score as our evaluation metric for the given task. Macro F1 score is computed using the arithmetic mean of all the F1 scores obtained per class. This arithmetic mean is unweighted and hence each class has an equal contribution towards the Macro F1 score.

4.1 Macro F1 Results

In the first experiment we used BERT's CLS tokens and added its output to a single dense layer. BERT used a loss of 1e-5 whereas the dense layer used a loss of 1e-3. This model gave us a base score of 0.756 Macro F1.

In the next setup, instead of using the CLS embeddings, we pooled the outputs of all the token embeddings of BERT's final transformer layer. We used the same loss parameters for loss. The results improved by about 0.12. The outputted Macro F1 score was 0.767.

The model which gave us our best accuracy is a finetuned version of the same concept. Instead of a

single dense layer connecting BERT's embeddings to 3 dimensions, we add another dense layer to improve the scores. We achieve the best results with this model, with a Macro F1 score of about 0.78 (an improvement of about 0.13).

The rest of the experiments were based on BERT based transformer models, incorporated with neural layers at the end. We changed various parameters and tried to add convolutional layers instead of linear layers to the model. However, we were not able to beat the accuracy of the previous model. A lot of these experiments resulted in a Macro F1 score of about 0.75-0.76.

In some experiments, we used data augmentation on the hate and normal speech classes of the dataset, since the data was biased towards offensive speech. This approach also showed some promise. We tried this approach with our base model, but were unable to achieve better results. We may incorporate augmentation for results in the final model.

We also miscellaneously tried some models that were not based on transformers, like Decision Tree classification and Logistic Regression. But these models scored much poorly as compared to BERT based models. We will not explore this direction further.

Finally, we tried ensemble modelling with 3 BERT models, each trained on a binary classifier (one-vs-rest architecture). These results were concatenated and pooled with 2 dense layers to obtain results. The current accuracies are unsatisfactory. However, we believe this method has a better scope to achieve higher Macro F1 scores and hence we shall be exploring this further.

Classifier	Macro F1
SBERT + Logistic Regr.	0.6687
Decision Tree	0.4312
BERT	0.756
BERTPool	0.767
BERTPoolLin	0.78
BERTPoolLin (Aug)	0.72
BERTEnsemble	0.6876

5 Analysis

We tried multiple architectures, starting from traditional machine learning method (Logistic Regression and Decsion Tree Classifier) to advanced transformer based models (BERT finetuning). We observe that we achieve better results by keeping BERT as the base architecture. However, using only the special [CLS] token did not yield good results, it is better to consider the entire sequence length embeddings and then train the classifier. The Ensembling methods are showing promise but we are still working to get better results. We did data augmentation since classes were imbalanced but the results are average since we are generating nearly then thousand samples from just one thousand samples. Post augmentation, we got comparable or sightly worse results.

6 Contribution

All the members contributed equally to the project. We distributed and took responsibilities of different sections among ourselves but worked collectively in all the sections.

- Related works: Diksha, Mudit, Niranjan, Vibhu
- Data preprocessing: Diksha, Vibhu
- Data augmentation: Diksha, Mudit
- SentenceBert embedding: Mudit
- Traditional ML models: Diksha, Niranjan, Vibhu
- Bert Classifier: Diksha, Mudit, Niranjan, Vibhu
- Modified Bert Classifiers: Vibhu, Niranjan
- Report + PPT : Diksha, Mudit, Niranjan, Vibhu

7 References

Davidson, T., Warmsley, D., Macy, M. and Weber, I. (n.d.). *Automated Hate Speech Detection and the Problem of Offensive Language* *. [online] Available at: https://arxiv.org/pdf/1703.04009v1.pdf.

Lu, Z. and Nie, J.-Y. (n.d.). *VGCN-BERT: Augmenting BERT with Graph Embedding for Text Classification*. [online] doi:10.1007/978-3-030-45439-5.

Mozafari, M., Farahbakhsh, R. and Crespi, N. (n.d.). *A BERT-Based Transfer Learning Approach for Hate Speech Detection in Online Social Media*. [online] Available at: https://arxiv.org/pdf/1910.12574v1.pdf [Accessed 3 Dec. 2022].

Djuric, N.; Zhou, J.; Morris, R.; Grbovic, M.; Radosavljevic, V.; and Bhamidipati, N. 2015. *Hate speech detection with comment embeddings*. In WWW, 29–30.

Kwok, I., and Wang, Y. 2013. *Locate the hate: Detecting tweets against blacks.* In AAAI.

Guterres, A. (2019). *UNITED NATIONS STRATEGY AND PLAN OF ACTION ON HATE SPEECH.* [online] Available at: https://www.un.org/en/genocideprevention/documents/UN