Detection of Hate Speech using BERT

*Note: Sub-titles are not captured in Xplore and should not be used

1st Abigail Yohannes
CDSE
North Carolina Agricultural and Technical State University
Greensboro, US
ayohannes@aggies.ncat.edu

***Abstract*—The continued rise of online interaction has required platforms to monitor for hate content. Efforts to use Natural Language Processing (NLP) to detect and mitigate the use of hate speech has become essential in reducing the impact of harmful content on social media users. New ways of masking hate speech have grown through the use of coded words, abbreviations, and specific spelling mistakes, making detection of these words much more difficult. Distinguishing between offensive or toxic language can be difficult from hate speech. Very often it's specific slurs relating to race, religion, skin color, sexual identity, gender identity, ethnicity, disability, or national origin that differentiates text from hate-speech to offensive language. However in many contexts offensive language contains similar characteristics including strong profanity, explicit content, at toxic connotations. In this sense it can be difficult to distinguish what type of language needs regulation on online platforms. Furthermore it can be difficult to distinguish among the various linguistic stlyes and speech patterns of cultural groups in which language that may seem offensive to some may not to others. Within this paper we investigate a multi-class classification problem using a pre-trained BERT Model to determine text as hate speech versus offensive and neither classified text. With the proliferation of Large Language Models (LLM) developed on large corpuses of text data, the ability to detect specific word patterns expands as these types of models are trained on billions of words. The pre-trained BERT model was used on the Hate Speech and Offensive Language Dataset sourced from Kaggle and used in the "Automated Hate Speech Detection and the Problem of Offensive Language" publication. Containing tweets of class hate, offensive language, and neither the model was used to predict tweets of each class. With minimal data labeled as hate speech the results showed inconclusive prediction of hate speech with an f1-score of 43 percent, however predicted offensive language with an f1-score of 95 percent and text classified as neither with an f1-score of 89 percent. Potential areas of concern for detecting hate speech come from imbalanced data and model complexity.**

***Index Terms*—Hate Speech, BERT, Offensive Language, bias, fine-tuning**

I. Introduction

Highlighting the extensive use of social media, there is a large amount of unregulated content making its way to these platforms in the way of racism, sexism, discrimination based on sexuality, religion, and other personal identifiers. Online platforms can unintentionally act as a common hub of destructive content for users. As more and more people engage on social media, the likelihood of encountering harmful or threatening content will continue to rise. Within the context of the article, approximately 60 percent of the world population uses some form of social media to communicate [1]. According to [2] 53 percent of Americans have experienced some form of harassment or hate speech online while [3] determined 21 percent of students commonly come across hate speech on social media platforms. As this problem continues to grow, social media companies, engineers, Trust and Safety personnel, and more work on the issue of detecting and removing this type of content from the platforms. However most organizations develop and define their own criteria for what constitutes hate speech, creating a structure where it can be difficult to determine hate speech from offensive language. Additionally, bias can be introduced to the data labeling process as various groups of people have a different linguistic pattern which can be perceived in various ways by the annotator. For example, it's very common that those speaking in African American Vernacular English (AAVE) have language misclassified as hate speech or offensive when using words pertaining to their culture such as n**ga. Among many in this group it is not interpreted as offense, however this word coming from another demographic could be classified as hate speech or offensive. Understanding not only the text but who is saying it and how it is being used is important in accurately detecting patterns of hate speech and offensive language.

According to [4] hate speech is defined as "as language that is used to expresses hatred towards a targeted group or is intended to be derogatory, to humiliate, or to insult the members of the group. In extreme cases this may also be language that threatens or incites violence..". A significant amount of nuance exists in what can be considered hateful or offensive from the reader's persepective. It is important to understand the difference of this classification as hate comments should not be tolerated by media platforms, however offensive language may need its own form of regulation. As there remains a lot of individual solutions for various forms of hate speech detection, there is yet to be a standard set for detecting these form of speech. Proliferating the use of LLM's in hate speech detection can potentially bridge the gap to creating a standard detection mechanism for various platforms [5]. The work of "Detection of Hate Speech using Bert and Hate Speech Word Embedding with Deep Model" is to help find solutions to create a safer

Identify applicable funding agency here. If none, delete this.

online environment. They aim to determine the best method for detecting hate speech using NLP while focusing on the nuances that may not always come across as hate initially. These come in the form of code words that are difficult to detect through deep learning (DL) and NLP. In response, to previous studies showing deep learning methods have worked well for multiple NLP use cases, the study conducted by [5] put forward the concept of using domain specific word embeddings. Within this study, the word embeddings were built on a hate speech corpus of over a million sentences. These domain specific word embedding results are then compared to domain-agnostic Google Word2Vec and GloVe. Furthermore, these are compared to pre-trained Bert Models which are known to achieve good results.

II. Related Works

As this problem continues to grow, social media companies, engineers, Trust and Safety personnel and more work on the issue of detecting and removing this type of content from the platform. The work of "De-tection of Hate Speech using Bert and Hate Speech Word Embedding with Deep Model" is to help find solutions to create a safer online environment. They aim to determine the best method for detecting hate speech using NLP while focusing on the nuances that may not always come across as hate initially. These come in the form of code words that are difficult to detect through deep learning (DL) and NLP. In response, to previous showing deep learning methods have worked well for multiple NLP use cases, the study conducted by [5] put forward the concept of using domain specific word embeddings. Within this study, the word embeddings were built on a hate speech corpus of over a million sentences. These domain specific word embedding results are then compared to domain-agnostic Google Word2Vec and GloVe. Furthermore, these are compared to pre-trained Bert Models which are known to achieve good results. Additionally, the work of "Automated Hate Speech Detection and the Problem of Offensive Language" provides the data used for this study collected from a twitter API and classified by Crowdflower (CF) employees [4]. Focusing on addressing bias in hate speech detection, using trained classifiers as an approach through the usage of a bias alleviation mechanisms and regularization techniques have been used to reduce racial bias within hate speech datasets [6].

III. Data

As reflected previously, the dataset used for came from the Automated Hate Speech Detection and the Problem of Offensive Language" [4]. They developed this dataset by identifying words as hate speech using Hatebase.org. To pull text data they accessed the Twitter API and assembled a random sample of a little less than 25,000 tweets. The team tasked with classifying the data into hate, offensive, and neither used the researchers definition of what constitutes hate vs offensive data. In addition, they were asked to consider the context of the tweet, not simply the words used as it was important to maintain that although a word or sentence may be offensive,

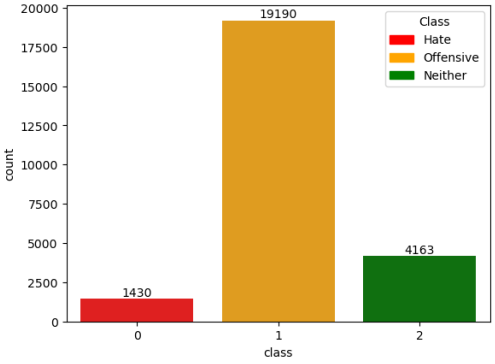

Fig. 1. Data Distribution

it may not actually connote hate speech. A group of three people from the team decided whether a certain tweet was hate speech, offensive or neither. Which ever label received 2 or more votes is what the tweets were classified as. A quick breakdown of the dataset shows where there are 1,410 hate tweets accounting for 5.8 percent of the dataset, 19,190 offensive tweets which accounts for 77.4 percent of the dataset, and 4,163 classified as neither which accounts for 16.8 percent of the dataset. As most of the data is classified as offensive it raises concern for training the model to detect hate tweets specifically due to the imbalance. This causes some issues in the prediction of hate tweets. Other researchers who used this data combined the hate and offensive tweets as one class which created a generalized detection of tweets with harmful language. However, it was important in this study to maintain a distinction between the two a offensive language can be considered as offensive or not by the user, but hate speech is undeniable.

Wordcloud representations were generated, showcasing the most common words that appear in tweets labeled as hate speech and offensive vs neither. You can see a lot of overlap between the hate speech and offensive language words as both maintain high levels of profanity. The Hate Speech wordcloud contains a few more racial and sexual orientation slurs. However, a lot of the verbiage is the same and you can still see some language such as n**ga or f***ot in both wordclouds, yet it appears at a higher rate in the hate labeled tweets. Although the neither category does have some words that could have been classified as offensive, the overall context of the tweet may have left data labeler's to believe that it wasn't hate speech or offensive language. Overall, the differentiating factor between the labeling of hate speech versus offensive data came down to the contextual interpretation of harsh language by data labeler's, therefore introducing potential bias into the dataset.

IV. Data and Text Preprocessing

To prep the data for training, some data preprocessing and text preprocessing is performed to get the text ready for tokenization. Here I remove usernames, urls, numbers, and punctuation to reduce noise in the data. The output as you

Fig. 2. Hate WordCloud Representations

Fig. 3. Offensive WordCloud Representations

can see contains tweets with just the words, other characters have been removed. To then tokenize the data for model understanding, Bert.Tokenizer uncased was used to create embeddings and vectorize the text. The token embedding vector is created by adding the token, segment, and position embeddings together. For sentence classification, BERT uses [CLS] short for classification, which is a unique token placed at the beginning of the sentence tokens to indicate the starting position of the classification task; in other words, the starting position of the fully connected layer to the last encoder layer. Token embeddings (or pre trained embeddings) containing up to word pieces of vocabulary up to 30,000 tokens are used. Segment embeddings, will be the sentence number encoded into a vector, and lastly the position embeddings are the positions of the words in the sentence encoded into the vector. Adding these three vectors together provided an embedding vector that would be input to BERT. For this model the max length of tokens for each input was set to 150. This was set this as the largest amount of tokens per tweet for standardizing the length of all inputs. The CLS token (101), has special

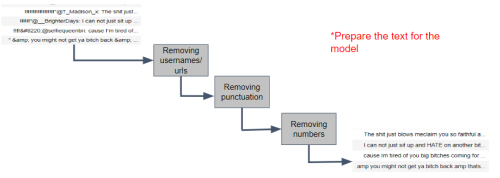

Fig. 4. Text Preprocessing

significance as BERT consists of 12 Transformer layers where each transformer takes in a list of token embeddings, and produces the same number of embeddings on the output, On the output of the final transformer, only the first embedding (corresponding to the [CLS] token) is used by the classifier. The function bert_encode was designed to tokenize and pad the list of tweets using a BERT tokenizer. Utilizing the transformers library, specifically the tokenizer.batch_encode_plus method, the function processes the input tweets in batches. It adds the special tokens and truncates sequences longer than a specified maximum length (max_len), and pads shorter sequences to match the max length. By returning tokenized input sequences converted into token IDs alongside attention masks, that distinguish between actual tokens and padding tokens, the function prepares the input data for BERT model training.

In order to get the train, validation and test inputs prepared for the model and evaluation process, each set of data is assigned their corresponding input sequences and attention masks for all. Subsequently, the arrays are converted into PyTorch tensors using the torch.tensor function. This allows seamless integration with PyTorch and enables efficient utilization of the data within for training and result evaluation. Lastly, data loaders are set up for both the training and validation datasets, facilitating the efficient processing of data during the training and evaluation phases. For the training and validation dataset, a TensorDataset is created with the there respective tokenized input sequences. For the train data a RandomSampler is applied to to shuffle the data before sampling to try and preventing the model from learning any order-related biases. A DataLoader is created for the training dataset, developed with the random sampler and a batch size of 32 to iterate over batches of training data during model training. The validation data goes under a similar process except a SequentialSampler is used to sample data in order. These data loaders are used to enhance efficiency and performance of the model development. The same is also done for test data [7].

V. Model

In this study I propose the usage of BERT or Bi-directional Encoder Representations from Transformers (BERT) for the detection of hate speech within the hate speech lexicon dataset proposed by (Davidson and Warmsly 2017). BERT is a type of deep learning Large Language Model (LLM) and Natural Language Processing (NLP) model developed by Google researchers in 2018, created to pre-train bi-dirctional representations from unlabeled text data reading from both left to right and right to left, improving individual word connections for text classification, sentiment determination, Neural Machine Translation, Q and A among other uses. BERT was pre-trained using text from Wikipedia and the Toronto BookCorpus containing over 3 billion words total. LLM's such as BERT have been known to display bias in areas such as gender and race. This model will be analyzed as part of my study on Language Bias Detection. To discuss the

architecture of the BERT model it is important to first understand transformers as this is the architecture BERT is based on. Transformers are a deep learning architecture which uses both attention mechanisms and encoder-decoder processing. Encoders will process the input sequences while decoders will create an output based on representations from the encoder. Additionally, it's essential to note that transformers process data in any order which is important for the BERT model as data is read bi-directionally.

Specifically in BERT, self attention is used to create relationships between words in the model. Self attention is important for BERT because it allows for the understanding of complex patterns, assigns weights based on relevance within a model for contextual understanding, and can compute multiple elements within in an input sequence at the same time [8]. When going through the self attention stage, input sequences are converted into query, key, and value vectors. This can be obtained through linear transformations of the input sequence. To determine the similarity between the query and key vectors the Cosine Similarity method can be used where the value 1 represents the highest similarity and -1 represents no similarity. The input embedding layer is where input sequences can be converted to a specific length vector based on so each word can be represented by a fixed value. Since transformers intake input in parallel allowing it to run faster than other models which take in data sequentially, position encoding is necessary to determine the location of each word embedding. The position embedding's go into the query layer where they are copied and go into the key and value layers. Before the positional embedding's are passed into the linear layer of the query key and values they are transposed. The values that come from the query and key layers then undergo matrix multiplication and the values produce an attention filter which is then trained and become accurate representations of the words given as a score. The scores are then reduced to values between 0 and 1 by using the softmax function which creates an attention filter. This is mulitplied by the value matrix and what then happens is relevant words remain and unneeded words are removed [9].

For the purpose of this study BERTBase with 12 transformers is used to The Embedding Layer includes word embeddings, position embeddings, token type embeddings, and layer normalization parameters. Transformer layers consist of parameters for self-attention mechanisms, intermediate dense layers, output dense layers, and layer normalization parameters. The Output Layer includes parameters for the final classification layer, consisting of weights and biases.

VI. Hyperperamter and Finetuning

To optimize Bert for the specific task of sequence classification the following criteria were implemented. A batch size of 32 which determines the number of samples processed in each iteration of training An AdamW optimizer is being initialized with certain hyperparameters: a Learning rate (lr) of 3e5, determines the size of the steps taken during optimization. The Adam optimizer adapts the learning rate for each parameter during training based on estimates of the first and second moments of the gradients and reduces the loss proportionally to 3e-5. An average learning rate was chose to help improve the speed o convergence without overshooting the minimum. Additionally, an epsilon parameter (eps) of 1e-8 was used to act as form of regularization that prevent numerical instability. The epsilon parameter also acts as a form of weight decay which should penalize large weights in the model to prevent overfitting by encouraging the model to use smaller parameter values. A learning rate scheduler is defined using LambdaLR. This scheduler adjusts the learning rate during training according to a given function lr_lambda. For this studty the learning rate is reduced linearly from the initial value to 0 over the training period. It gradually reduces the learning rate as training progresses. The number of epochs is then set to 4 which will be the number of times the model will see the entire dataset during training. This number was selected to enable the model to learn from the data but not memorize and cause overfitting[7].

VII. Model Deployment

To split the dataset for running the model a train_test_split was performed (80 percent going to training, 10 percent to validation and 10 percent). Firstly, for running the train and validation model the random seeds for reproducibility are set to 42 ensuring consistent results across runs. Lists are initialized to store the training loss, accuracy, and number of evaluation steps. The training loop iterates over each epoch, where the model is set to the training mode. Within each epoch, the training data is iterated over in batches. For each batch, the gradients are cleared, and the model's forward pass is computed with the input IDs, attention mask, and labels. The loss is then calculated and backpropagated to update the model's parameters. The gradients are clipped to prevent them from becoming too large. It's cliped the norm of the gradients of the parameters in the bert_model to a maximum value of 1.0. To compute the norm of all gradients together and scales them down if the norm exceeds the specified threshold of 1.0. By limiting the gradient norm, this operation helps to stabilize the training process and prevent gradients from growing excessively, preventing the risk of gradient explosion. The learning rate is adjusted using a scheduler defined as lambda_Lr Specifically, the learning rate multiplier decreases linearly from 1.0 to 0.0 as the training progresses. After each epoch, the model's performance is evaluated on the validation set to calculate accuracy. The accuracy and number of evaluation steps are stored for analysis. Finally, the accuracy results are saved to a file using pickle.

To evaluate on the test dataset, predictions and true labels are being tracked for each batch of data. For each batch of data in the prediction_dataloader, which contains tokenized input sequences, attention masks, and labels for the test dataset, the model's forward pass is executed without gradient calculation using torch.no_grad(). The logits produced are obtained from the model's output and converted to a NumPy array, along with the corresponding label IDs. These predictions and true

Fig. 5. Training Loss Curve

labels are stored in separate lists as predictions and tru_labels to be tracked across all batches. After processing all batches, the predictions and true labels are linked along the batch dimension and saved to separate files using NumPy's np.save(). The use of pickle and np.save() was an attempt to try and save all training and test results so the model would not have to be reloaded. Unfortunately, running this model did generate some issues in terms of deployment times with full deployment taking anywhere from 2 to 3 days on CPU. This factor is important when analyzing results [7].

VIII. Results

For training and validation I reran my model as I would often lose progress or my kernel would die before getting to the next step. Regrettably this is some of the hardship incorporated with running a model as large as BERT on CPU. The training loss decreases from .30 to .11 by epoch 4. With a consistent validation accuracy of .91. indicating that the model should generalize well to unseen data and continues to perform well throughout training. I should have also monitored the validation loss so I could check to see if it was decreasing as the model validation accuracy remained consistent to make sure that accurate predictions were happening through the training process. Because I didn't do this there may be some overfitting that I have not been able to detect.

With the offensive labeled tweets having a f1-score of 95 percent a precision of 93 percent and a recall of 96 percent. The model was able to predict instances of offensive speech relatively well, although with a higher recall it's possible that there are some false positives. It appears that the BERT model performed well for this class.

Additionally with an f1-score of 89 percent, the overall performance between class prediction on text not classified as hate or offensive performs similarly to previous literature

Where my model begins to see issues is in the classification of hate text with a precision of 50 percent a recall of 37 percent and an overall performance of 43 percent the model has not learned how to classify unseen hate data from the test set. This is likely due to the imbalanced data, as there were only a little over 1000 tweets for the hate class while other classes had significantly more data.

The following confusion matrix showing the predicted vs the actual values on the test set. This reveals that the offensive language had a pretty high number of tweets accurately predicted as offensive with 1828 computed correctly. More

	precision	recall	f1-score	support
0	0.47	0.37	0.42	134
1	0.94	0.96	0.95	1904
2	0.91	0.88	0.89	440
accuracy			0.91	2478
macro avg	0.77	0.74	0.75	2478
weighted avg	0.91	0.91	0.91	2478

Fig. 6. Test Predictions Confusion Matrix

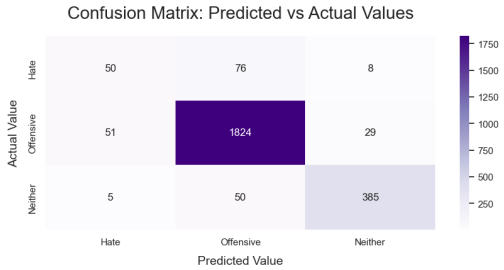

Fig. 7. Confusion Matrix: Actual vs Predicted

hate text is predicted as offensive than as hate speech, showing there is a lot of overlap in the sentiment of hate and offensive language and it was difficult for the model to distinguish between the two.

IX. Conclusion

From previous works it's been noted building large pre-trained models from rich domains specific content in current social media platforms can be effective for classifying hate data. BERT combines the benefits of domain-agnostic, domain-specific word embedding and domain-specific data (fine-tuning) due to the large amount of data it was trained [5] In trying to keep the definition of hate speech and offensive language different on the model evaluation my hate speech detection results suffered significantly. Overfitting due to small size of test data for hate speech was highly likely. In the future it would be important to find a more balanced dataset or at least have more hate speech data for the BERT model to train on for Hate Speech due to complexity. Additionally, having the oppurtunity to make more adjustments in the finetuning of hyperparameters can improve results in the future. Deploying this model on GPU instead of CPU could also contribute to improved results. Although the BERT model did not perform well in detecting hate speech for this study, building on the aforementioned critiques can significantly improve prediction of hate speech, as the offensive and neither predictions happened to perform as expected signifying there is potential for improvements to be made. It still remains important to differentiate hate speech vs offensive data as they are not the same and shouldn't be treated as such by organizations Furthermore, investigation on how text in hate datasets are classified is essential to reducing bias. Future exploration includes a deep dive into Racial Bias in existing Hate Speech Detection Models and the potential of building LLM's specifically with hate speech signifiers/ classified words to understand context,

sarcasm, bias against cultural and linguistic speech patterns for the future safety of online platforms.

REFERENCES

[1] Ltd, We Are Social. 2020. Digital 2020. Accessed November 2008.

[2] League, A.D. Online hate and harassment: The American experience, Pew Research Center: Internet, Science and Tech.2019

[3] Clement, J. U.S. teens hate speech social media by type 2018 l Statistic, October.2019

[4] Davidson, T., D. Warmsley, M. Macy, and I. Weber. "Automated hate speech detection and the problem of offensive language." In Eleventh international aaai conference on web and social media, Canada. 2017.

[5] H. Saleh, A. Alhothali, and K. Moria, "Detection of Hate Speech using BERT and Hate Speech Word Embedding with Deep Model," vol. 37, no. 1. Informa UK Limited, Feb. 02, 2023, doi: 10.1080/08839514.2023.2166719.

[6] M. Mozafari, R. Farahbakhsh, and N. Crespi, "Hate speech detection and racial bias mitigation in social media based on BERT model," vol. 15, no. 8. Public Library of Science (PLoS), Aug. 27, 2020, doi: 10.1371/journal.pone.0237861.

[7] Chris Nick and Mccormick, "BERT Fine-Tuning Tutorial with PyTorch ," 2019, [Online].

[8] "Self-attention. What is Self-attention?" (n.d.). https://h2o.ai/wiki/self-attention

[9] Kalra, G. (2022, June 6). "Attention networks: A simple way to understand self attention." Medium.

[10] Waseem, Z. 2016. Are you a racist or am i seeing things? annotator influence on hate speech detection on twitter. In Proceedings of the first workshop on NLP and computational social science, Austin, 138–42.

[11] Gupta, S., and Z. Waseem. 2017. A comparative study of embeddings methods for hate speech detection from tweets. Association for Computing Machinery.

[12] Efimov, V. (2023, September 16). Large language models: Bert - bidirectional encoder representations from Transformer. Medium.