Hate-speech detection and classification

Sirash Phuyal, Sushen Kolakaleti, Sam Macy
Virginia Tech
phuyals2@vt.edu, sushenk21@vt.edu, sammacy@vt.edu

https://github.com/Sam-Macy/ECE4424-proj

Abstract

The pervasive nature of hate speech on digital platforms has highlighted the need for more robust automated detection mechanisms. This project delves into the classification of hate speech in tweets, leveraging three distinct embedding techniques: BERT, GloVe, and word2vec. While one segment of our project will employ the DistilBERT model, capitalizing on its advanced transformer architecture, the others concurrently explore the potential of word2vec and GloVe embeddings. It also tries to address the unique challenge of multiple annotations per tweet using a majority voting approach. This threefold embedding exploration aims to offer a comparative analysis on the efficacy of each approach for hate speech detection. Future endeavors will harness cloud-based platforms to facilitate extensive model training and evaluation.

1 Introduction

Hate speech detection is a pressing challenge in the realm of natural language processing (NLP) and social media analytics. With the exponential growth of user-generated content on social media platforms, the prevalence of hate speech and offensive content has also seen a surge. Such content not only violates platform guidelines but also fosters a toxic online environment, contributing to serious real-world consequences.

Our project embarks on utilizing the capabilities of advanced natural language processing techniques, specifically focusing on three contrasting embeddings: BERT (Bidirectional Encoder Representations from Transformers), word2vec, and GloVe. BERT offers deep, context-aware word representations, while word2vec encodes each word with a vector and uses these vectors to compare with other words. GloVe fits in between these two using vectorization, but making connections with a pre-trained set of vocabulary. Our objective is twofold: to develop a dynamic hate speech detection system that evolves with changing language patterns, and to compare and contrast the performances of BERT, word2vec, and GloVe embeddings in this context. By juxtaposing these methodologies, we aim to offer a comprehensive perspective on the strengths and potential limitations of each embedding in addressing the challenge of hate speech detection.

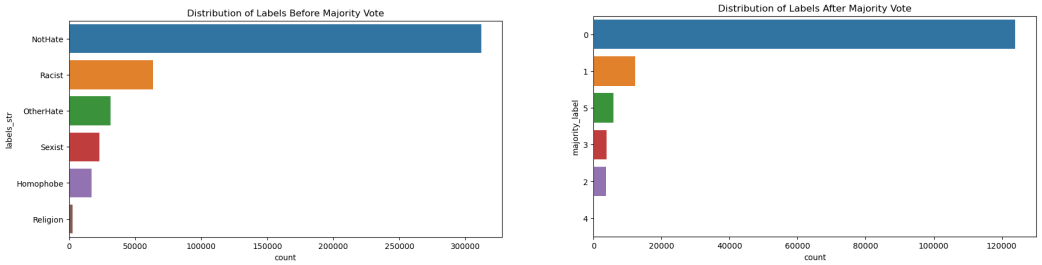
2 Dataset and Pre-processing

2.1 Dataset Description

The dataset utilized in this project is the MMHS150K dataset, which is multimodal (i.e., contains both text and images) and freely available on Kaggle. Directly sourced from Twitter, it comprises 150,000 tweets, where each entry is annotated by multiple independent human annotators, ensuring a robust label assignment. The labels encompass various categories of hate such as racism, sexism, and religious bigotry.

2.2 Pre-processing

1. **Loading and Conversion**: The raw dataset, stored in a JSON format, was initially loaded into the Python environment. To make it easier for manipulation and analysis, it was converted into a pandas DataFrame.
2. **Majority Voting for Labels**: Given there are multiple annotations for each tweet, a majority voting system was employed based on some research as an appropriate approach to tackle this issue. In cases where all annotators provided distinct labels, the label was chosen at random. This approach ensures a single, decisive label for each tweet. We realize this may not be the best approach, but for now we decided to stick with it. Here's the result before and after implementing the majority voting approach:



(a) (b)

Figure 1: Before vs after majority voting for labels

The substantial decrease in label counts after applying the majority vote is expected because now, multiple annotations for each tweet are consolidated into a single label. For example, this means that for each tweet with three labels, instead of counting all 3 (one from each annotator), you're now counting just one label.

3. **Text Cleaning**:
   - *URLs and Mentions Removal*: Tweets often contained URLs and user mentions, which don't contribute to the sentiment or content. Hence, these were stripped from the text.
   - *Punctuation and Numbers Removal*: To maintain uniformity and focus on words, all punctuations and numbers were removed.
   - *Lowercasing*: The entire text was converted to lowercase to ensure word uniformity.
   - *Tokenization and lemmatization*: For the GloVe based implementation,each tweet was tokenized, and the tokens were subsequently lemmatized using the WordNet Lemmatizer from the NLTK library.Lemmatization is a process that converts words to their root form. For example, 'running' becomes 'run', and 'better' becomes 'good'. This helps in reducing the word variations in the text, making it easier for the model to understand and process. This was also initially done for the BERT-based implementation, but upon further research, it was realized that removing stop words and lemmatizing the input text actually causes context-rich embedding models like BERT to perform worse. Hence, it was subsequently removed from the final implementation.
   - *Stopwords Removal*: Common English stopwords were removed to retain only the meaningful tokens in the tweets.

   Here's a sample output after implementing the above cleaning steps (Offensive words have been censored from the image):

	tweet_text	\
1114679353714016256	@FriskDontMiss N[redacted]a https://t.co/cAsaLWEpue	
1063020048816660480	My horses are retarded https://t.co/HYhqc6d5WN	
1108927368075374593	"N[redacted]A ON MA MOMMA YOUNGBOY BE SPITTING REAL S...	
1114558534635618305	RT xxSuGVNGxx: I ran into this HOLY N[redacted]A TODA...	
1035252480215592966	"EVERYbody calling you N[redacted]r now!" https://t....	

	cleaned_tweets
1114679353714016256	n[redacted]a
1063020048816660480	horse retarded
1108927368075374593	n[redacted]a momma youngboy spitting real shit n[redacted]a
1114558534635618305	rt xxsugvngxx ran holy n[redacted]a today
1035252480215592966	everybody calling n[redacted]r

Figure 2: Tweet data before and after cleaning

4. **Dataset Splitting**:
   - *BERT*: For the BERT model, the dataset was divided into training, validation, and test sets using predefined IDs included with the dataset. This ensured a consistent split, allowing for reproducible results.

   - *word2vec&GloVe*: For the word2vec and GloVe embedding methods, sklearn's 'train_test_split' method was used to automatically perform the splits for the training and testing data, this method does a random split of the data and therefore has a small variance in accuracy of roughly a quarter of a percentage point.

3 Methodology

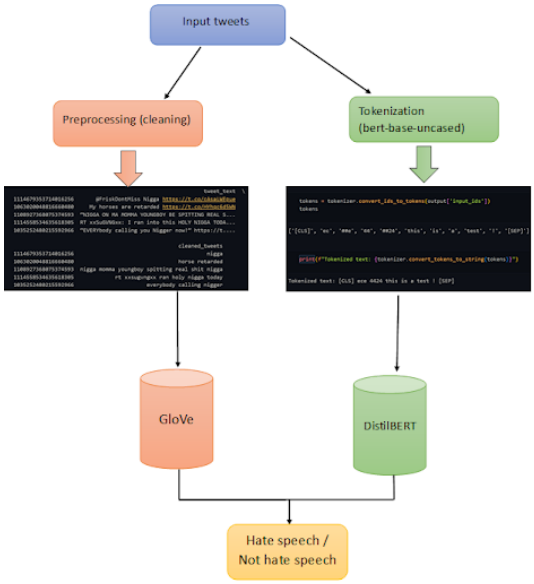


Figure 3: Process of training different embedding models

For our models to understand the text data, we needed to convert our tweets into a format that they can process. This is where embeddings come in:

Embedding Preparation:

- *BERT*: For the BERT-based model, we utilized the BERT tokenizer, specifically the 'bert-base-uncased' version, which not only breaks words into tokens, but also converts these tokens into unique IDs that correspond to the BERT vocabulary. BERT has a maximum sequence length of 512 tokens. If a tweet exceeded this length after tokenization, it was truncated to fit within this limit (But since tweets are usually short, this didn't pose any significant issues).To ensure consistent input sizes, we padded shorter sequences to a length of 512 tokens using zeroes. In order to differentiate between the actual content of the tweet

and the padding, a mask was added where a value of '1' indicates a real token, while a value of '0' indicates a padding token. At this point, these formatted inputs are ready to be fed into a model for training, which will be done in the next phase of the project.

- *word2vec*: The word2vec model is a simple model that takes the text and puts them into a vector of the texts. Explained above, the word2vec model uses some basic cleaning before creating the embedding and the model. The embedding takes advantage of the sklearn python library that has extensive machine learning tools. Specifically, CountVectorizer which simply takes a text document and converts it into a matrix of token counts. This matrix is the embedding, which we can then use for a model of choice. All things simplified, the word2vec model essentially turns the raw word data input into a series of vectors that is the embedding matrix. The embedding matrix can be simply plugged into a machine learning model of choice.

- *GloVe*: The GloVe model is similar to the word2vec model in that it is a vectorization-based embedding method; however, it has one key difference to the word2vec model. The GloVe model uses a list of global word vectors, which is essentially a pre-set vocabulary or compilation of a set number of parameters. The GloVe model makes connections with its input data and the set of pre-trained or pre-set parameters, to make a vectorized list of embeddings for a machine learning algorithm to use. For this implementation of the GloVe model, we used a pre-set vocabulary of 6B or 6 billion parameters, which is among the smallest of global vectorization. The reason for this choice was because of our computational limitations. We ran these models on our local machines, and so we had to use a smaller set of parameters because a very large set of parameters would take too long with our computational barriers. The GloVe model implementation also takes use of the CountVectorizer from the sklearn python library to vectorze the data.

Model Training

- *BERT*: For our BERT implementation, we decided to use the lighweight version called DistilBERT. DistilBERT is essentially a traditional BERT algorithm of tokenization and contextualization; however, it adds an extra step known as distillation. Distillation, in this context, means that the algorithm was trained in a student-teacher way. This means that if, for example, a traditional larger BERT model (known as the teacher model) has 2n attention layers, the smaller student model would have n attention layers. The point of this is to have a smaller model that tries to mimic the behavior of the larger model, while still maintaining accuracy and computing performance.

  The process of the training is that we train a traditional large BERT model, and then after we train the smaller student BERT model (DistilBERT) to mimic the behavior of the first model. The benefit of doing this is that we get a much faster output when testing our models on thousands of test data samples, while maintaining the original accuracy and performance. Again, this helps us to make rapid changes to our embedding models and see the effects of those changes for the purpose of comparative analysis.

- *word2vec/GloVe*: For the word2vec and GloVe models, we also used a Multinomial Naive Bayes classifier, which is suitable for classifications with discrete features such as texts. We chose the Multinomial Naive Bayes classifier because it is a simple, yet effective classifier for natural language processing tasks. This model also ensures that we can make fast changes to our embedding models to better understand the impact of the embedding model itself. Consequently, this method also means that we are able to see a much clearer comparison between the accuracies of the embedding models, whereas an algorithm like a feedforward neural network has much more variance to its results. However, if we were looking to improve the accuracy of these embedding models in the future, a deep neural network would be a better approach because of its generally higher accuracy.

4 Results and Analysis

- *word2vec*: test accuracy of 68.92%
- *GloVe*: test accuracy of 70.24%
- *DistilBERT*: test accuracy of 85.48%

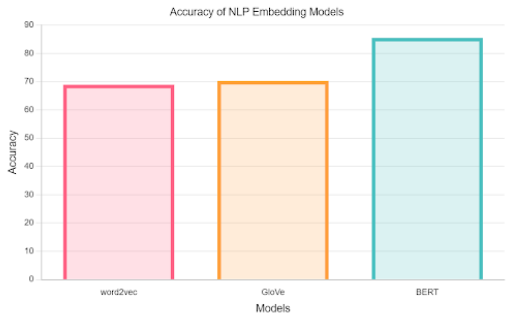


Figure 4: Chart visualization of embedding models results

We can see that the vectorized models have a lower accuracy, this is expected due to a few things, word2vec is a very simple model that simply attempts to make a connection between the text and the label. The GloVe model trains on a pre-set vocabulary and then makes the connections. Due to computational limitations (as expressed before), the smallest (6B word) vocab was used, which resulted in a slightly lower accuracy. A higher parameter global vectorization in the ballpark of 50B or 50 billion parameter could have yielded a much higher accuracy. The BERT model has the highest accuracy, since this model tokenizes the words within the text, it can make different connections based on where a word appears in a sentence. This makes the BERT model context-aware. Since a text may or may not be hate speech, depending on where a word appears in the text, this higher accuracy is to be expected.

Furthermore, a higher accuracy for all the models could have been achieved had we used a more complicated machine learning algorithm, as mentioned earlier. Something along the lines of a deep feedforward neural network would have made the embedding models more accurate, simply because of its greater prediction accuracy and size of the model. The two primary reasons that we did not choose to do this during this semester-long project. The first reason was that because a deep neural network takes too long to train, and we cannot make fast and frequent changes to the embedding models to see changes. The second reason was that we wanted to focus on a comparative analysis of different embedding models, and we felt that a complicated machine learning algorithm would distract from that primary goal. This was due to output variance and resource allocation towards the machine learning method itself.

5 Contribution

Sirash Phuyal: Implemented dataset cleaning and pre-processing, tokenization and trained the DistilBERT model.

Sushen Kolakaleti: Implemented vectorization and GloVe model.

Sam Macy: Implemented vectorization and word2vec model.

References

[1] https://www.tensorflow.org/text/tutorials/classify_text_with_bert.

[2] https://www.kaggle.com/datasets/victorcallejasf/multimodal-hate-speech

[3] https://www.kaggle.com/datasets/watts2/glove6b50dtxt/

[4] https://nlp.stanford.edu/projects/glove/