Detecting and Classifying Hate Speech Using NLP Techniques

Sonali Palit, Suryam Gupta

The code used to train the models and generate the results can be found here:
https://github.com/suryamgupta25/Hate-Speech-Classification

Abstract

The increasing use of hate speech and offensive language on social media poses a significant societal challenge. This study investigates the effectiveness of various natural language processing (NLP) techniques and machine learning models for detecting and classifying hate speech. The research focuses on classifying tweets into three categories: *Hate Speech*, *Offensive Language* and *Neither*. Using the publicly available Hate Speech and Offensive Language Dataset, we experiment with traditional methods like logistic regression, support vector machines (SVM) and advanced techniques including transformer-based BERT. Our approach involved advanced data preprocessing, feature engineering with Bag-of-Words (BOW) and Word2Vec and addressing class imbalance in the dataset. The results show that logistic regression and SVM perform well with BOW features, while BERT achieves the highest accuracy (82.2%) but is computationally expensive. These findings highlight the balance between computational efficiency and model performance in hate speech detection.

Introduction

Hate speech and offensive language have become prevalent on platforms like Twitter and contribute to online toxicity. These forms of communication can harm individuals, escalate conflicts and perpetuate discrimination. The ability to automatically classify hate speech is crucial for content moderation and fostering a safer online environment. However, distinguishing hate speech and offensive language from non-hateful language remains a challenge.

Our project aimed to address these challenges by exploring the effectiveness of Natural Language Processing (NLP) techniques in detecting and classifying hate speech. The research questions we addressed were:

1. Could we effectively classify hate speech and offensive language from non-offensive language using pre-existing NLP techniques?
2. What were the key linguistic or semantic features distinguishing hate speech from offensive but non-hateful language?
3. How did different machine learning models (e.g., logistic regression, support vector machines, neural networks) perform on this classification task, particularly in identifying hate speech?

Our approach involved building a classifier capable of detecting hate speech and offensive language. We experimented with various NLP techniques and evaluated the performance of different machine learning models. The results indicated that careful feature engineering and model selection could lead to significant improvements in classification accuracy.

Related Work

Two prior research papers form the foundation for our work:

1. https://arxiv.org/pdf/2211.00243v1 - "Why Is It Hate Speech? Masked Rationale Prediction for Explainable Hate Speech Detection" by Jiyun Kim, Byounghan Lee, and Kyung-Ah Sohn

This paper discusses the flaws of hate speech identification, centered around how they don't properly use human context to make accurate judgements, and then proposes a Masked Rationale Prediction (MRP) model to solve these. Even if "hateful" words are present in the sentence, it doesn't imply that the sentence conveys a hateful tone. However, the presence of such words can easily bias a model that doesn't pay attention to human rationale. To remove potential biases and narrow down what causes hate speech to be classified a certain way, Kim et al. propose a new model that masks the human rationale annotations of certain words and then fine-tunes a BERT model to determine if those annotations are really what makes the sentence hate speech. Through experiments and comparison with previous work (ex: HateXPlain), they determine that their new MRP model doesn't lose accuracy compared to previous models, and it can better track what annotations and words create hateful tone in a sentence.

2. https://aclanthology.org/W17-1101.pdf - "A Survey on Hate Speech Detection using Natural Language Processing"

This paper provides a comprehensive review of tools, techniques and challenges associated with identifying hate speech. Some tools that the paper addresses are n-grams, bag of words (BOW), and synthetic structures. It classifies techniques into two categories: traditional machine learning methods (logistic regression, SVM, decision trees, etc.) and more modern neural network approaches, such as CNNs, RNNs, and BERT. Challenges that the authors point out are the lack of actual "hate" speech data, because most conversations online aren't hate speech, and the vague definition of "hate speech" - what exactly determines that hate speech should be labeled as "hate speech"?

Data

Dataset

We used the Hate Speech and Offensive Language Dataset sourced from Kaggle.

https://www.kaggle.com/datasets/mrmorj/hate-speech-and-offensive-language-dataset/data

It is publicly available, pre-annotated and well-documented, making it suitable for our classification task. The average tweet length was around 12.5 words. The dataset consisted of 24,783 labeled tweets

categorized into three classes: hate speech, offensive language and neither and had the following class distributions:

- Hate Speech (Class 0): 1,430 instances
- Offensive Language (Class 1): 19,190 instances
- Neither (Class 2): 4,163 instances

One key observation was the significant class imbalance, with over 75% of the data classified as offensive language. This imbalance posed a challenge, as models trained on such data risked biasing predictions toward the majority class. To address the imbalance, we created a balanced dataset by limiting each class to 1,000 randomly sampled instances from the original dataset. This approach ensured an even distribution and allowed us to observe the model's performance on underrepresented classes, specifically 'hate speech.'

Data Preprocessing

To prepare the dataset for analysis, we implemented several preprocessing steps to clean and standardize the text data:

- Removed @username and &username mentions from the text
- Removed hyperlinks to eliminate irrelevant information
- Converted all text to lowercase for consistency
- Removed stopwords to focus on meaningful words
- Converted emojis into descriptive text to preserve sentiment and context
- Lemmatized the text to reduce words to their base forms

After preprocessing, the average tweet length was about 7.5 words.

We split the dataset into training, validation and testing subsets, using 70% for training, 10% for validation and 20% for testing across all models. This approach ensured robust model evaluation and facilitated hyperparameter tuning during the validation phase.

Methods

To understand how hate speech and offensive language can be detected and understood by a computer, several models were trained on the preprocessed dataset. First, a baseline logistic regression model was trained using a bag of words (BOW) model. The BOW implementation estimated the probability of the word occurring given the class as its TF-IDF: the number of times the word appeared in the class’s tweets divided by the total number of words in the class’s tweets. The TF-IDF vectorizer was used to implement this component of the model. One notable feature of the vectorizer is its feature extraction capability: post training, the words and weights could be extracted to determine the words that had the most impact in the text classification process.

Then, more complex models built on the initial implementation. Another logistic regression model was created, but instead of using BOW to get features, GloVe was used to get the tweet embeddings. The problem with using semantic-rich embeddings such as Word2Vec here is Word2Vec generates embeddings for individual words, but feature generation in this scenario requires entire document embeddings. Multiple approaches were considered when deciding on how to generate these. One was to use another library - Doc2Vec. While there were open source Doc2Vec models online, there weren't enough resources available to bring the models, train them on personal devices using the extremely large corpora, then fine tune the models for the dataset being used in this project. Another possibility was training a new Doc2Vec model from scratch. However, the dataset chosen does not have enough data points to both generate accurate embeddings on the trained data, then predict embeddings for new text observed.

With word embeddings, there were multiple open source pretrained models that could be downloaded in relatively quick time. The Gensim GloVe-twitter-200 pretrained model was chosen, as it was pretrained on tweets and generated a substantial number of features for each word. However, this model is no different from Word2Vec in that features are generated on a word-by-word basis. To generate the document embedding, word embeddings have to be somehow combined without a significant loss of latent and semantic information. Two approaches were considered: taking the average of all word embeddings to generate the document embedding, and aligning the word embeddings side by side. The first method considers all words in the text equally, but loss of sequential information is present when the words' embeddings are simply averaged. The second method considers sequencing of data by aligning the embeddings instead of aggregating them. However, this results in different length embeddings for different tweets. To train models such as logistic regression, the feature matrix size must be fixed. Either the document embedding would need to be restricted, meaning some of the text would have to be cut off, resulting in loss of key information, or the embeddings are padded with extra zeros that can negatively impact training.

Despite inherent problems with both approaches in generating features, the second method of appending word embeddings sequentially was chosen. Sequences were limited to a length of 5000 (equivalent to the first 25 words, since each word has an embedding of length 200). The rest of the empty embedding data was padded with zeros to each the target size. As the average tweet length was about 12.5 words prior to preprocessing and about 7.5 words post preprocessing, limiting sequences to the first 25 words seemed substantial enough to cover the majority of tweets. However, a lot of the embedding data ended up consisting of many zeros due to the low text length and the high upper bound of sequence length.

In addition to the two logistic regression models, two versions of SVM were also trained. One SVM model used the same BOW implementation (TF-IDF) as the baseline logistic regression model, and the other SVM implementation used the same GloVe feature generation algorithm as the other logistic regression model. To implement validation and hyperparameter tuning in logistic regression and SVM, GridSearchCV was used for stratified cross validation with cv = 8 folds to represent a 70 to 10 percent split in the total training and validation data.

Finally, to analyze text classification with a neural model, a fine tuned BERT model was also created. The open source HuggingFace pretrained BERT model 'bert-base-uncased' was used, and it was fine tuned on

the balanced, preprocessed dataset. Instead of the 70 percent training, 10 percent validation, 20 percent testing used for the rest of the models, the dataset was split into 70 percent training, 15 percent validation, and 15 percent testing for BERT. A custom dataset class, ‘HateSpeechDataset’, was created to tokenize tweets using the ‘AutoTokenizer’ for BERT, and the three datasets (training, validation, and test) were prepared using this class. Texts were tokenized with padding and truncation to a maximum length of 128 tokens - tweets longer than this were truncated, and shorter ones were padded. The new model was trained using the AdamW optimizer with weight decay to stabilize training and cross-entropy loss for multiclass classification. After extensive hyperparameter tuning, the most optimal settings for fine tuning the BERT model were a learning rate of 2e-5, batch sizes of 16 for training and 64 for evaluation, and three epochs for training. The process of hyperparameter tuning revealed that lower learning rates were better for this task, due to the nature of BERT having already been pre-trained on hundreds of millions of parameters, and performance plateaued outside of the ideal three epochs.

Results

Baseline Logistic Regression Model (TF-IDF feature generation)

Time for training: 0.25 seconds
Accuracy: 0.8233

Class	Precision	Recall	F1
Hate Speech (class ‘0’)	0.84	0.69	0.75
Offensive Language (class ‘1’)	0.81	0.83	0.82
Neither (class ‘2’)	0.82	0.95	0.88

Most Important Words Used to Distinguish Each Class (separated by class, sorted by descending weight):

Class → Words ↓	Hate Speech (class ‘0’)	Offensive Language (class ‘1’)	Neither (class ‘2’)
First Ranked Word (weight)	'Faggot' (4.1173)	'Bitch' (6.4923)	'Bird' (3.0252)
Second Ranked Word (weight)	‘Nigger' (3.8765)	'Pussy' (3.2473)	'Yankee' (2.4419)
Third Ranked Word (weight)	‘White' (3.3357)	'Hoe' (3.1333)	'Charlie' (2.2481)
Fourth Ranked Word (weight)	‘Nigga' (3.3079)	'Shit’ (1.5390)	'Yellow' (2.0804)

Fifth Ranked Word (weight)	'Fag' (2.6798)	'Aint' (1.4679)	'Oreo' (1.6405)

Logistic Regression Model (GloVe feature generation)

Time for training: 2.64 minutes (CPU)
Accuracy: 0.7667

Class	Precision	Recall	F1
Hate Speech (class ‘0’)	0.74	0.66	0.70
Offensive Language (class ‘1’)	0.76	0.73	0.74
Neither (class ‘2’)	0.79	0.92	0.85

SVM (TF-IDF feature generation)

Time for training: 4.33 seconds
Accuracy: 0.8133

Class	Precision	Recall	F1
Hate Speech (class ‘0’)	0.85	0.64	0.73
Offensive Language (class ‘1’)	0.79	0.84	0.82
Neither (class ‘2’)	0.81	0.96	0.86

Most Important Words Used to Distinguish Each Class (separated by class, sorted by descending weight):

Class → Words ↓	Hate Speech (class ‘0’)	Offensive Language (class ‘1’)	Neither (class ‘2’)
First Ranked Word (weight)	'Faggot' (2.2456)	'Bitch' (3.2655)	'Bird' (1.6244)
Second Ranked Word (weight)	‘Nigger' (2.1001)	'Pussy' (1.6780)	'Yankee' (1.3224)

Third Ranked Word (weight)	‘White' (1.8235)	'Hoe' (1.6070)	'Charlie' (1.2237)
Fourth Ranked Word (weight)	‘Nigga' (1.6261)	'Aint’ (0.8090)	'Yellow' (1.1319)
Fifth Ranked Word (weight)	'Fag' (1.4221)	'Shit' (0.6998)	'Oreo' (0.8747)

SVM (GloVe feature generation)

Time for training: 77.9 minutes (CPU)
Accuracy: 0.7550

Class	Precision	Recall	F1
Hate Speech (class ‘0’)	0.73	0.62	0.67
Offensive Language (class ‘1’)	0.74	0.75	0.74
Neither (class ‘2’)	0.79	0.90	0.84

Fine Tuned BERT

Time for training: 2 minutes (T4 GPU) - approximately equivalent to between 10 and 40 minutes on a CPU
Accuracy: 0.8200

Class	Precision	Recall	F1
Hate Speech (class ‘0’)	0.77	0.76	0.76
Offensive Language (class ‘1’)	0.83	0.76	0.79
Neither (class ‘2’)	0.86	0.93	0.89

Discussion and Future Work

Analysis of Presented Results of all Models and Future Work

The presented results indicate that the logistic regression and SVM models performed significantly better with a simpler TF-IDF feature representation than the GloVe representation, in terms of both accuracy and F1 scores across all three classes. This difference was present the most when classifying “offensive language”, where F1 scores differed by 0.08 between feature representations, in both models. In terms of which of the two models performed better, based on F1 scores, logistic regression had a slight advantage over SVM when classifying texts as “hate speech” or “neither”, and both models performed similarly when classifying “offensive language”. This observation was independent of the chosen feature representation technique. Results also showed that BERT, while being the most advanced model implemented, did not outperform the logistic regression and SVM models that used TF-IDF features. BERT did slightly outperform both models in terms of classifying “hate speech”, according to F1 scores - while its precision was lower, its recall was much higher, suggesting that BERT was more thorough in finding examples of “hate speech”, although not as effective in classifying its identified “hate speech” correctly. BERT’s F1 score was higher for “neither” compared to that of logistic regression and SVM, but lower for “offensive language”, showing that the fine tuned model may have been more biased towards classifying text as “neither”, or more general, instead of “offensive language”.

While the results presented as is indicate that TF-IDF feature generation is sufficient for accurately classifying tweets as “Hate Speech”, “Offensive Language”, or “Neither”, in most applications and in theory, more advanced semantic features are required for text classification to be accurate. Specifically in the context of this application, which is primarily to distinguish between “hate speech” and “offensive language”, semantic data seems necessary to differentiate between when swears are just used in regular speech versus when swears appear to be targeted at someone or a group of people, which would cause the text to be offensive. However, it is difficult to use word embedding models, such as Word2Vec and GloVe, to comprehensively generate an embedding for a document, because the combination step can result in loss of important data. As described in the ‘Methods’ section, the two most straightforward approaches (averaging the individual word embeddings or appending the embeddings in sequence) both have shortcomings. Averaging individual word embeddings assumes that every word, regardless of meaning or its position in the text, has equal importance in predicting the text’s class, which is untrue. In this application, swears or targeted groups of people should take precedence over other words. It also results in loss of sequence data, which is vital because English text is meant to be read and analyzed from left to right, and the ordering of words in the sentence can greatly change its meaning.

If sequential data is deemed necessary, instead of averaging the embeddings, the embeddings could be attached in sequence to form a long feature vector for each document. While this preserves sequential data, it still does not mitigate the assumption that every word has equal importance in the text’s classification, regardless of what context it is present in. This is because embeddings for each word are generated independently, so even though positional data is preserved, word dependence in the data is not captured, so none of the embeddings had any additional meaning across different contexts.

In hindsight, although Doc2Vec is not trivial to implement and consumes more resources and time than Word2Vec and GloVe, Doc2Vec is specifically meant to generate document embeddings, which is notably more relevant to the application than combining individual word embeddings. It will be able to capture both sequential and contextual dependencies between words. Future work could entail fine tuning Doc2Vec to the original dataset (class data isn’t needed to generate document embeddings, so the more

important part of the dataset is its size of about 25,000 tweets), or training a completely new Doc2Vec model on a larger corpus of tweets, which could be combining tweets from multiple data sources and websites.

In terms of BERT performance, there are multiple factors that could be contributing to its lack of benefit over simpler logistic regression and SVM models. One is the lack of data in the balanced, preprocessed dataset. With only 3000 tweets total to fine tune a BERT model that has millions of parameters, the weights are unlikely to significantly change towards the intended application, so there is not much of a benefit of the fine tuned model relative to the base BERT model and simpler models. Another limitation of fine tuning BERT is the specific application and the dataset chosen for this project. BERT is trained on well-formatted, structured language from various sources and domains on the Internet. However, tweets are obtained from a select few websites (in this case, Twitter) and are informal, containing a lot of slang and word choice that BERT isn't accustomed to. In addition to lack of balanced data to fine tune with, such a text structure may cause BERT to not learn as much towards an application of classifying informal text. Future work to improve the BERT model would definitely be gathering more data, possibly from either datasets or websites, while ensuring that the dataset is still balanced so BERT doesn't bias towards a particular class.

Feature Extraction for Analyzing Model Performance and More Future Work

Unfortunately, because the TF-IDF vectorizer was not used in the models that generated GloVe or other embeddings instead of TF-IDF, individual word and weight data that was most significant in determining text classification weren't able to be retrieved. However, both the logistic regression and SVM models were able to identify the same top five words that aided in classification, in almost the exact same order. The words make sense in this application as well - the most significant hate speech identified were either swears that are targeted towards specific groups of people ("nigga", "fag", etc.), or the groups of people themselves ("white"). The most significant offensive language were commonly used swears that are often deemed offensive, but don't necessarily imply hate towards someone (for example, "bitch" and "pussy"), with the exception of 'ain't'. Other text, whose subspace of significant words is much larger than that of specifically hate speech or offensive language identification, would more likely depend on how often the individual words appear in the dataset (due to the TF-IDF feature generation), and are more likely to appear randomly selected instead from the English vocabulary instead of specifically being chosen for a specific application. The weights of all of the words were higher in logistic regression than SVM, but this is most likely due to different optimization goals of logistic regression versus SVM, rather than differences in model performance.

However, such feature extraction is not very informative when it comes to determining where each model struggled the most. To aid in future analysis of where exactly all of the models struggled with, it would be beneficial to extract tweets that had the highest probability of belonging to a certain class, but that class prediction ended up being incorrect. This way, potential weaknesses in the model can be identified and models can be fine tuned to address them.