Project Report: Hate Speech Detection Using Transformers (Deep Learning)

Name: Rakshit Grover
University: University of Southern California
Specialization: NLP
Batch Code: LISUM35
Email: rakshitg@usc.edu
Country: United States
Date: SEPTEMBER 29, 2024
Submitted to: Data Glacier

Table of Contents:

6. EDA

7.Model Building

7.1 Transformer Model Overview

7.2 Machine learning Models Overview

8. Model Evaluation

8.1 Transformer Model Evaluation

8.2 Machine learning Models Evaluation

9.Conclusion & Future Work

Project Plan

Week 1 (August 18, 2024):

- Deliverables: Problem Statement, Data Collection, Data Report

Week 2 (August 25, 2024):

- Deliverables: Data Preprocessing

Week 3 (September 1, 2024):

- Deliverables: Feature Extraction

Week 4 (September 8, 2024):

- Deliverables: Model Building

Week 5 (September 15, 2024):

- Deliverables: Model Evaluation

Week 6 (September 22, 2024):

- Deliverables: Flask Development, Heroku Deployment

Week 7 (September 30, 2024):

- Deliverables: Final Submission (Report, Code, Presentation)

2. Problem Statement

Hate speech refers to any form of verbal, written, or behavioral communication that attacks or uses derogatory or discriminatory language against individuals or groups based on factors such as religion, ethnicity, nationality, race, gender, or other identity markers. This project aims to build a machine learning model that can detect hate speech in text using Python.

The task of hate speech detection is often treated as a sentiment classification problem. For this project, we will focus on classifying hate speech using Twitter tweets as our data source. By training the model on data typically used for sentiment classification, we can create an effective tool to identify tweets containing hate speech.

3. Business Understanding

Hate speech detection is essential for online platforms to maintain a safe and inclusive environment.
1. Market Need: With the rise of social media, hate speech has become more prevalent, requiring effective detection systems to foster positive engagement.

2. Brand Impact: Companies that fail to address hate speech risk damaging their reputation and losing users. Proactive measures enhance corporate responsibility and build community trust.

3. Regulatory Compliance: As regulations against hate speech tighten, effective detection helps organizations avoid legal issues and fines.

4. Operational Efficiency: Automating detection reduces the workload on human moderators, enabling faster responses to incidents.

5. Data Insights: A machine learning model provides valuable insights into user behavior, guiding policy adjustments and interventions.

This project addresses a critical societal issue while positioning businesses for success in a digital landscape.

4.Data Collection

The dataset for this project, sourced from Kaggle, consists of Twitter data used for research in hate speech detection. The dataset contains 31,962 observations and three features. The text is categorized into three classes: hate speech, offensive language, and neither. It is important to

note that the dataset includes content that may be considered offensive, including racist, sexist, or homophobic language.

5.1 Text Cleaning:

- **5.1.1 Convert to Lowercase:**
  All text was converted to lowercase to ensure uniformity. For example, words like "Racism" and "racism" should be treated as the same in a vector space model, preventing the creation of unnecessary dimensions.
- **5.1.2 Remove Punctuation:**
  Punctuation marks do not contribute meaningful information to the model, so they were removed using regular expressions.
- **5.1.3 Remove URLs:**
  Since URLs are not relevant to the hate speech detection model, they were removed from the text to ensure only the core content is processed.
- **5.1.4 Remove @tags:**
  User mentions, signified by @tags, were removed because they are irrelevant to the model's task of detecting hate speech.
- **5.1.5 Remove Special Characters:**
  Special characters such as [!"#$%&'()*+,-./:;<=>?@[]^_{|}~] were eliminated as they hold no significance for the model. This was accomplished using Python's isalnum()` function.

5.2 Preprocessing Operations

In this section, we apply various preprocessing operations to prepare the text data for modeling.

5.2.1 Tokenization
Tokenization is the process of breaking down raw text into smaller, meaningful units called tokens. Since our data is composed of paragraphs, we use the `nltk.word_tokenize` library to convert the text into individual words (tokens). This step is essential for Natural Language Processing (NLP), as it helps the model interpret the context of the text by analyzing the sequence of words.

5.2.2 Removing Stopwords
Stopwords are common words like 'a,' 'is,' 'the,' and 'are,' which do not contribute significant meaning in NLP tasks like hate speech detection. To remove these irrelevant words, we first tokenize the text as mentioned above, and then exclude any tokens that match the stopwords provided by the `nltk.corpus.stopwords` collection. This ensures that the model focuses on the more meaningful words in the text.

5.2.3 Lemmatization

Lemmatization involves grouping together different inflected forms of a word, treating them as a single item. Unlike stemming, lemmatization takes the context of the word into account, ensuring that similar words are linked to their base form. For example, words like "intelligently" and "intelligence" are converted to their root form, "intelligent." This process enhances the model's ability to understand the true meaning of the text.

5.2.4 WordCloud

A WordCloud is a visual representation of text data where the most important words appear larger or in different colors. This tool helps to visually distinguish between keywords in datasets. Below, we provide WordClouds for both hate speech and free speech, highlighting the most frequent terms in each category.

5.3 Feature Extraction

5.3.1 TF-IDF Model

After preprocessing the text, we apply the Term Frequency-Inverse Document Frequency (TF-IDF) model to extract relevant features. The model calculates the importance of each word in the dataset by assigning a score based on its frequency across documents. From this, we select the top 2,000 most frequent words for both hate speech and free speech categories. The resulting word count vectors represent the frequency of these words in the entire dataset, and these vectors serve as input for training our model.

5.4 Splitting the Data into Train and Test Sets

For model development, we divide the dataset into two parts: 80% for training and 20% for testing. This data split allows the model to learn patterns from the training data, while the testing set is used to evaluate its performance. Proper data splitting is crucial for creating accurate and reliable machine learning models, as it ensures that the model generalizes well to unseen data.

6. Exploratory Data Analysis (EDA)

EDA was conducted to gain initial insights into the dataset and better understand the distribution of classes as well as the frequency of key terms.

- **Class Distribution**:
    - The dataset consists of 2 classes: hate speech and not offensive. A bar plot was generated to visualize the distribution, showing that the majority of tweets fall into the "not offensive" categoryThis imbalanced class distribution suggests that special techniques like oversampling or undersampling might be needed.
- **Word Frequency Distribution**:

    - To better understand the textual data, we generated a **WordCloud** for each class. It was observed that offensive language often contains profanity, while neutral tweets include common conversational words.
    - **Most Common Words**: A frequency plot highlighted the most common terms across all categories, providing useful insight into key discriminatory features between classes.
- **Sentence Length Distribution**:
    - Tweets were analyzed based on the number of words they contain. Hate speech tweets tend to be slightly longer on average than neutral ones, possibly due to the usage of multiple slurs and offensive remarks in the same tweet. This insight will later help inform decisions during the model-building phase.

7. Model Building

7.1 Transformer Model Overview

For the hate speech detection task, we implemented a Transformer-based model due to its superior performance in NLP tasks. Specifically, we utilized the **Bidirectional Encoder Representations from Transformers (BERT)** model, pre-trained on a large corpus of general English text. BERT's ability to understand the context of a word in relation to all of its surrounding words makes it an ideal choice for hate speech detection, where nuanced language understanding is crucial.

Model Architecture:

1. **BERT Tokenizer**:
    - Text data was preprocessed using BERT's tokenizer, which tokenizes the tweets and maps them to the corresponding BERT vocabulary.
    - Special tokens, such as [CLS] (indicating the start of a sentence) and [SEP] (marking the end of a sentence), were added to each tweet.
2. **BERT Encoder**:
    - The BERT model's encoder processes the tokenized data. Since BERT is bi-directional, it analyzes text both forwards and backward, allowing it to capture the context of words more accurately.
3. **Output Layer**:
    - A fully connected layer was added on top of the BERT model to perform classification into three classes (hate speech, offensive language, neutral). The output layer used **Softmax** activation to output probabilities for each class.
4. **Fine-tuning BERT**:
    - Since BERT is pre-trained, we fine-tuned it on our dataset. This involved unfreezing the weights of the BERT layers and training the model further on our task-specific dataset.
5. **Loss Function and Optimizer**:

- We used **Cross-Entropy Loss** for multi-class classification. The **AdamW optimizer** was chosen for training due to its efficiency in handling large NLP models like BERT.

7.2 Machine Learning model Overview

Created a machine learning pipeline for text classification, integrating SMOTE (Synthetic Minority Over-sampling Technique) to handle imbalanced datasets. Here's an overview:

1. Data Splitting: The dataset, is split into training and validation sets using `train_test_split`.

2. Pipeline Setup: An imbalanced-learn `Pipeline` is created with three steps:

- TF-IDF Vectorizer: Converts the raw text data into numerical features.

- SMOTE: Balances the dataset by oversampling the minority class.

- Classifier: A placeholder for different classifiers like `LogisticRegression`, `RandomForestClassifier`, and `GradientBoostingClassifier`.

3. Grid Search: A `GridSearchCV` is used to perform a hyperparameter search over the classifiers and their parameters. The parameter grid includes options for each classifier's hyperparameters (e.g., regularization strength for logistic regression, number of estimators for random forests, and learning rate for gradient boosting).

4. Model Selection: The grid search finds the best classifier and hyperparameters by training on the training set with 5-fold cross-validation.

5. Evaluation: The best model is used to make predictions on the validation set, and a `classification_report` is generated to assess model performance in terms of precision, recall, and F1-score.

8. Model Evaluation

8.1 Transformer Model Evaluation

•The evaluation results indicate that the model has a high accuracy of 94.5% on the validation set, but the recall is relatively low at 36.8%, suggesting it misses many positive instances. The precision is 73.0%, and the F1-score, which balances precision and recall, is 48.9%, indicating a moderate overall performance.

8.2 Machine learning Models Evaluation

•The model achieves an overall accuracy of 96%, performing well on the majority class (0) with high precision (97%) and recall (99%). However, it struggles with the minority class (1), showing lower recall (59%) and a moderate F1-score (68%), indicating it misses many positive instances despite decent precision (81%). I utilized Logistic Regression, Random Forest, and Gradient

Boosting, and after performing a grid search and comparing models based on accuracy, Random Forest emerged as the best performing model with these metrics.

9. Conclusion & Future Work

Summary:

Hate speech detection models were developed using machine learning techniques, applying oversampling methods like SMOTE to tackle class imbalance and leveraging TF-IDF for text feature extraction.

Limitations:

Although the models achieve reasonable accuracy, some misclassifications, particularly false positives and negatives, are expected. This can be due to the model's inability to fully capture linguistic subtleties like sarcasm, slang, and implicit hate speech, as well as contextual variations across different platforms and cultures.

Future Enhancements:

To enhance the model, advanced techniques like Temporal Convolutional Networks (TCN) could be employed to capture long-term dependencies in sequential text data. Additionally, Random Multimodal Deep Learning (RMDL) could combine models trained on multiple data modalities, improving robustness and accuracy for detecting hate speech across diverse content types, including text, images, and audio.