**ARTIFICIAL INTELLIGENCE**

**(Real-World Problem Solving Using Artificial Intelligence Approaches)**

**Project on:**

**“Your Project Title”**

**Submitted By**

Student Name (Student Number)

**Submitted On**

10th May 2024

**Table of Content**

[Abstract 3](#_Toc173189626)

[1. Introduction 4](#_Toc173189627)

[2. The Real-World Problem 8](#_Toc173189628)

[3. Project Aim and Objectives 9](#_Toc173189629)

[4. Adopted Artificial Intelligence Approach 10](#_Toc173189630)

[5. Evaluation, Results and Discussions 14](#_Toc173189631)

[6. Conclusion and Future Work 18](#_Toc173189632)

[References 20](#_Toc173189633)

[Appendix 21](#_Toc173189634)

# Abstract

Social media has revolutionized communication and community formation, but it has also become a platform for disseminating hate speech and offensive language. This project explores using artificial intelligence, specifically natural language processing, to develop an efficient system for detecting hate speech on social media, focusing on Twitter data. Employing state-of-the-art models like BERT, RoBERTa, and BERTweet, we address the challenges posed by the nuanced and evolving nature of online hate speech.

The project implemented two training strategies: one that froze the transformer layers and fine-tuned only a single-layer classifier, and another that fine-tuned the entire model. The evaluation demonstrated the superior performance of the fully fine-tuned models. RoBERTa, in particular, achieved the best results, effectively distinguishing between hate speech and offensive language.

Key achievements include model evaluation using accuracy, precision, recall, and F1 score; confusion matrix analysis to identify classification strengths and weaknesses; and a strategy comparison that highlighted the effectiveness of full model fine-tuning over partial. Future development can enhance the project through advanced models, larger datasets, longer training periods, multilingual support, real-world testing, and improved model interpretability. This project successfully lays the groundwork for future advancements in automated hate speech detection with more sophisticated AI and extensive data.

# 1. Introduction

Social media has transformed how people interact, share information, and form communities. However, it has also become a platform for the dissemination of hate speech and offensive language, which can harm individuals and societal groups. The vast scale and rapid pace of content generation make manual moderation efforts insufficient and highlight the urgent need for automated solutions. This project explores the application of artificial intelligence, specifically **natural language processing**[[1]](#footnote-1), to develop a reliable and efficient system for detecting hate speech on social media, focusing specifically on Twitter textual data. By employing state-of-the-art models such as BERT[[2]](#footnote-2), RoBERTa[[3]](#footnote-3), and BERTweet, we aim to address the challenges posed by the nuanced nature of language and the evolving ways in which hate speech is expressed online. Our goal is to create an AI-driven framework that not only improves the speed and accuracy of hate speech detection but also supports human moderators in maintaining safer online communities.

Hate speech on social media is a significant and complex issue that has garnered substantial research attention over the years. The phenomenon is characterized by offensive or harmful content directed at individuals or groups based on attributes such as race, ethnicity, gender, religion, and more [1].

A study conducted by Alkomah and Ma [1] provides a comprehensive review of textual hate speech detection systems, focusing on their primary datasets, textual features, and machine learning models. The study reveals a diversity of approaches to hate speech detection, including semantic analysis, fuzzy logic, and machine learning techniques. Many previous studies have focused on specific categories of hate speech or legal frameworks, often neglecting the broad spectrum of hate speech manifestations on platforms like Twitter [1].

The literature on hate speech detection encompasses a variety of machine learning techniques, each with unique strengths and limitations. Traditional approaches like **TF-IDF[[4]](#footnote-4)** transform text into numerical feature vectors, facilitating text classification when paired with machine learning algorithms. Neural network models such as **CNNs[[5]](#footnote-5)** and **RNNs**[[6]](#footnote-6)have also been widely used. CNNs excel at capturing local patterns in text, making them effective for identifying specific linguistic features associated with hate speech. RNNs, on the other hand, are adept at processing sequential data, enabling them to capture temporal dependencies crucial for understanding context. More recent advancements in the field have been driven by transformer models like BERT, which leverage bidirectional context understanding, making them highly effective for complex language tasks such as hate speech detection. Lexicon-based models, which utilize predefined lists of offensive terms, provide a straightforward approach but often fall short in handling contextual nuances. Hybrid models, which combine various techniques, have emerged as state-of-the-art solutions, leveraging the strengths of each method to improve detection accuracy and robustness. These advancements highlight the ongoing evolution of methodologies aimed at enhancing the precision and reliability of hate speech detection systems.

Despite advancements in the field, several challenges remain. The inconsistency in results across different hate speech categories poses a significant hurdle. Additionally, many existing datasets are either too small or unreliable, limiting their effectiveness in hate speech detection tasks [1]. This inconsistency is exacerbated by the complex nature of hate speech itself, which often overlaps with other forms of offensive and abusive language [1].

The decision to implement BERT in this project stems from its superior ability to handle the complex and nuanced nature of language used in hate speech. BERT is a transformer-based model that functions solely as an encoder, designed to process text bidirectionally [2]. This bidirectional processing allows BERT to capture the full context of a word based on both its preceding and succeeding words, a capability that is critical in accurately detecting hate speech, where meaning can be highly context-dependent.

BERT and its family of models, such as ALBERT, RoBERTa, and others, are trained on vast amounts of text data. These models excel in **natural language understanding[[7]](#footnote-7)** tasks due to their ability to learn complex language patterns and contextual relationships. For instance, RoBERTa improves on BERT by optimizing training strategies, which leads to enhanced performance in various NLU benchmarks [3].

The strength of BERT lies in its pre-trained models, which provide a robust foundation for fine-tuning on specific tasks like hate speech detection. This approach to transfer learning allows for improved performance with relatively less task-specific data compared to other models. This characteristic is particularly useful in domains where labeled data is scarce or where data needs to be processed efficiently.

RoBERTa and BERTweet are two advanced transformer-based models that have significantly contributed to the field of natural language processing. RoBERTa is an enhanced version of the original BERT model. It was developed by optimizing BERT's training process, including removing the next sentence prediction task, dynamically changing the masking patterns, and pre-training on larger datasets with longer sequences [3]. These improvements allow RoBERTa to achieve superior performance on various natural language understanding tasks. On the other hand, BERTweet is the first large-scale pre-trained language model specifically designed for English Tweets [4]. It shares the same architecture as BERT-base but utilizes the RoBERTa pre-training procedure to better capture the nuances of Twitter data [4].

In summary, BERT and its related models offer a powerful solution for understanding and moderating hate speech through their advanced language comprehension capabilities and their adaptability to complex and changing textual data environments.

# 2. The Real-World Problem

Hate speech is a pervasive problem on social media platforms, where it can promote discrimination, hostility, and violence. The challenge of moderating such content lies in the sheer volume of posts generated daily and the subtlety with which hate speech can be disguised. Current manual moderation techniques are not only labor-intensive but also often inconsistent, failing to keep up with the dynamic nature of online communication.

The distinction between hate speech and offensive language is critical in the context of social media. Hate speech is specifically targeted at disadvantaged social groups and is intended to harm, demean, or incite violence against these groups [5]. In contrast, offensive language can include derogatory terms or curse words that, while potentially vulgar or inappropriate, do not necessarily target specific groups or aim to cause harm [5].

This distinction is crucial because hate speech poses a greater threat to societal harmony by inciting violence and social disorder, prompting stricter social media policies compared to offensive language. Context is key in differentiating them [5]. Hate speech uses slurs with historical oppression context to harm or incite violence against groups, whereas offensive language may not target specific groups and lacks harmful intent. Hate speech often has legal consequences, with laws and social media policies in place to curb it, while offensive language generally does not, unless threatening [5].

The real-world challenge in automating the detection of hate speech versus offensive language lies in accurately capturing the context and intent behind the words. Automated systems that rely solely on keyword detection risk misclassifying offensive language as hate speech. This can lead to over-censorship or inappropriate flagging of content, hindering freedom of expression and failing to address the core issues of targeted hate.

Therefore, the development of automated systems that can effectively differentiate between these types of language is essential. This project aims to leverage advanced natural language processing models, trained on comprehensive datasets like the HSOL[[8]](#footnote-8) dataset, to improve the precision and reliability of content moderation systems. By doing so, we can ensure that hate speech is promptly and accurately identified, protecting users and fostering healthier online environments.

# 3. Project Aim and Objectives

The aim of this project is to develop a robust AI-based system for the detection and classification of hate speech on social media platforms. To achieve this aim, the project is guided by several specific objectives:

* **Objective 1**: Design and implement advanced NLP techniques to preprocess and analyze textual data effectively. This involves handling raw text, cleaned text, and lemmatized text to ensure comprehensive feature extraction. The HSOL dataset's labeled data supports this by providing examples across a spectrum of language uses and abuses [5].
* **Objective 2**: Train a variety of machine learning models, including BERT, RoBERTa, and TweetBERT, to accurately classify social media posts as hate speech, offensive language, or neutral content. The dataset's diversity in examples allows us to evaluate which models perform best in terms of accuracy and efficiency, offering insights into the contextual understanding of these models.
* **Objective 3**: Rigorously evaluate the performance of different model training strategies, such as training from scratch, fine-tuning specific layers, and etc, using the HSOL dataset.

By focusing on these objectives and utilizing the HSOL dataset, this project aims to contribute significantly to the field of automated hate speech detection and provide practical tools for social media platforms.

# 4. Adopted Artificial Intelligence Approach

Initially, exploratory data analysis[[9]](#footnote-9) was conducted to gain insights into the dataset, which consists of 24,783 entries and six columns, namely count, hate\_speech, offensive\_language, neither, class, and tweet. Each column is devoid of null values, indicating a complete dataset with no missing entries.

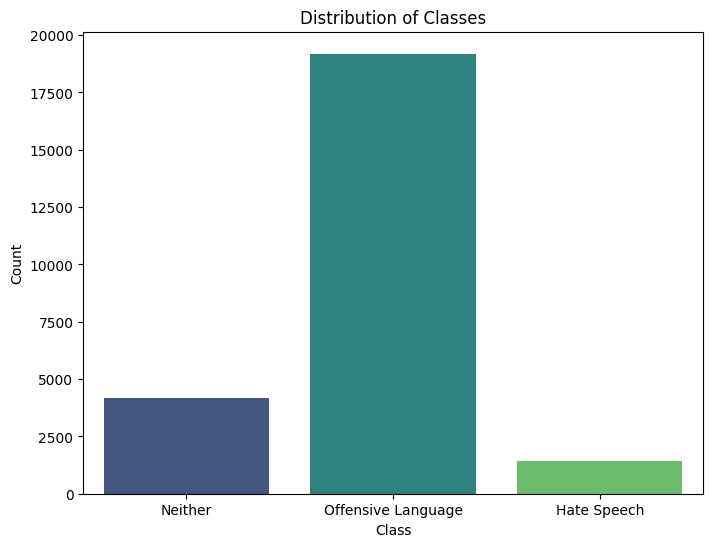


Figure : The class distribution of dataset.

I verified the dataset's integrity by checking for duplicates and missing values, finding none. The class distribution (as shown if figure 1) is notably imbalanced, with 77.43% of entries labeled as Offensive Language, 16.80% as Neither, and 5.77% as Hate Speech. This imbalance highlights the challenge of training a model to accurately classify minority classes.

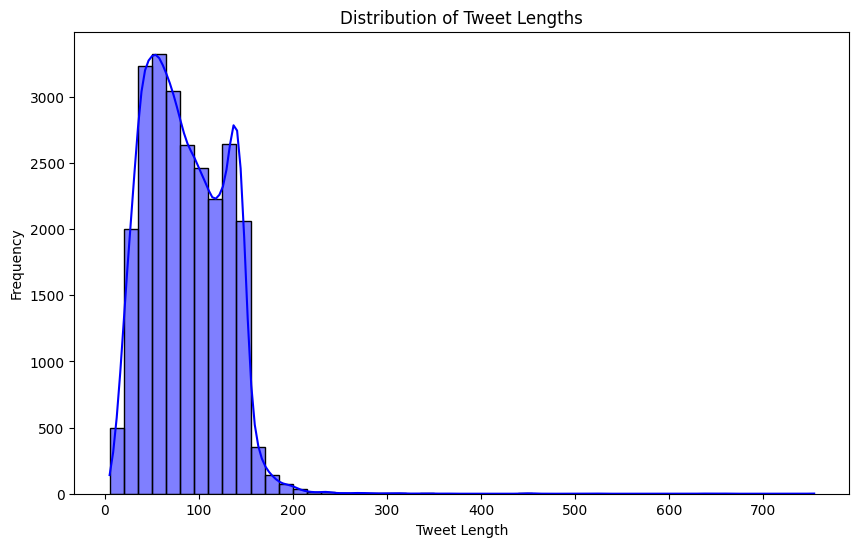


Figure : histogram of lengths of tweets.

Most tweets are succinct, with the median and mean length of tweets being about 100 characters. To explore the distribution of tweet lengths, I introduced a new column, tweet\_length, to capture the length of each tweet. Figure 2 shows a histogram of these lengths that reveals a concentrated distribution, demonstrating the typical shortness of tweets. Based on the range of tweet lengths (characters), the 128 **token size** for inputs in machine learning models, surely is a good choice.

Additionally, I identified the most frequently occurring words, including terms such as 'bitch', 'hoe', and 'nigga'. Figure 3 ,as a word cloud, visualizes these words, illustrating their prevalence in the dataset.

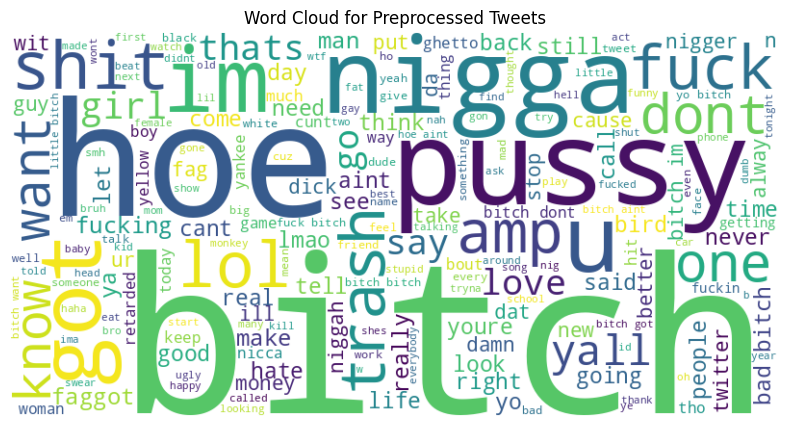


Figure : Most frequent words as word cloud.

Following EDA, preprocessing was conducted to prepare the dataset for model training. Preprocessing plays a crucial role in enhancing the quality of short texts, which is essential for improving the performance of various text classification tasks, including hate speech detection on social media platforms like Twitter. The study by Naseem et al. provides a comprehensive analysis of multiple preprocessing techniques and their impact on the classification of short texts, particularly tweets​ [6].

The preprocessed\_tweet column, containing sanitized, lemmatized, and stopword-free text, was created as the input for the machine learning models. The preprocessing steps included removing retweet indicators, mentions, URLs, special characters, and numbers; converting HTML entities and text to lowercase; expanding contractions; tokenizing text; removing stopwords; and lemmatizing the tokens.

To ensure effective model training and evaluation, the data was split into training, validation, and test sets with a 70-15-15 ratio. This strategy ensures that the majority of the data is utilized for training while maintaining balanced distributions across validation and test sets. The dataset splits were saved to separate CSV files for future use: train.csv, val.csv, and test.csv.

The thoughtful approach to data exploration, cleaning, and splitting lays a solid foundation for the subsequent application of machine learning models. To classify tweets into categories such as hate speech, offensive language, and neither, I employed three distinct transformer models: BERT-base, RoBERTa-base, and BERTweet. Each model was trained using fully preoprocessed version of the tweet data, namely preprocessed\_tweet, to evaluate their effectiveness in handling various levels of data processing.

In this project, two distinct strategies were employed for training and fine-tuning transformer models to classify hate speech, offensive language, and neutral content on social media platforms. The first strategy involved freezing the pre-trained model's encoding layers and only fine-tuning the classification layer. This approach reduces computational load and mitigates the risk of overfitting, making it especially useful when computational resources are limited. By focusing on the classification layer, the model retains the robust language understanding capabilities of the pre-trained encoder while adapting specifically to the task at hand.

The second strategy involved full fine-tuning of the model. This strategy adjusted both the encoding and classification layers, allowing the model to fully adapt to the dataset's specific nuances. Full fine-tuning leverages the model's strengths in understanding the informal and often noisy language characteristic of tweets. Although this approach typically results in higher performance metrics, it requires more computational power and is more susceptible to overfitting due to the larger number of trainable parameters.

# 5. Evaluation, Results and Discussions

The evaluation of the AI-based hate speech detection system involved rigorous testing using a variety of metrics to ensure its effectiveness and reliability. The primary models used—BERT-base, RoBERTa-large, and BERTweet—were assessed based on their accuracy, precision, recall, and F1 score. The performance metrics were calculated on the validation and test datasets split from the HSOL dataset.

**Table 1: Performance Metrics on Test Set**

| **Model** | **Training Time (s)** | **Accuracy** | **Precision** | **Recall** | **F1 Score** |
| --- | --- | --- | --- | --- | --- |
| BERT-base (Strategy 1) | 810 | 0.808 | 0.749 | 0.808 | 0.764 |
| BERTweet (Strategy 1) | 795 | 0.835 | 0.782 | 0.835 | 0.803 |
| RoBERTa-base (Strategy 1) | 807 | 0.846 | 0.829 | 0.846 | 0.816 |
| BERT-base (Strategy 2) | 1217 | 0.910 | 0.900 | 0.910 | 0.910 |
| BERTweet (Strategy 2) | 1293 | 0.910 | 0.910 | 0.910 | 0.910 |
| RoBERTa-base (Strategy 2) | 1296 | 0.910 | 0.910 | 0.910 | 0.910 |

**Table 2: Confusion Matrices**

| **Model** | **Confusion Matrix** |
| --- | --- |
| BERT-base (Strategy 1) | [**0**,182,25], [1,**2799**,80], [1,424,**205**] |
| BERTweet (Strategy 1) | [**0**,182,25], [0,**2777**,103], [0,302,**328**] |
| RoBERTa-base (Strategy 1) | [**5**,179,23], [3,**2792**,85], [0,283,**347**] |
| BERT-base (Strategy 2) | [**80**,117,10], [65,**2759**,56], [17,71,**543**] |
| BERTweet (Strategy 2) | [**94**,104,9], [84,**2737**,59], [14,62,**555**] |
| RoBERTa-base (Strategy 2) | **[96**,99,12], [83,**2734**,63], [17,58,**556**] |

* **Training Time**: BERT-base (Strategy 1) was the fastest to train, completing in 810 seconds, followed closely by BERTweet and RoBERTa-base. Strategy 2 models took more time to train, with RoBERTa-base and BERTweet taking the longest at 1296 and 1293 seconds, respectively. However, these models consistently achieved better performance metrics.
* All models in Strategy 2 achieved a high accuracy of 0.91, significantly higher than the models in Strategy 1.
* The models in Strategy 2 also showed perfect precision and recall values, indicating a balanced and effective detection capability. Consistently high F1 scores across Strategy 2 models indicate that they provide a good balance between precision and recall. RoBERTa-base in Strategy 1 achieved the highest F1 score among the models in that strategy, reflecting its superior performance.
* The confusion matrices reveal that Strategy 1 struggled the most with detecting class 0 (hate language). On the other hand, Strategy 2 (especially RoBERTa-base) showed better performance in correctly classifying instances of hate language.

**Table: Performance on Class 0 (Hate Language)**

| **Model** | **Precision (Class 0)** | **Recall (Class 0)** | **F1 Score (Class 0)** | **True Positives (TP)** | **False Positives (FP)** | **False Negatives (FN)** |
| --- | --- | --- | --- | --- | --- | --- |
| BERT-base (Strategy 1) | 0.00 | 0.00 | 0.00 | 0 | 0 | 182 |
| BERTweet (Strategy 1) | 0.00 | 0.00 | 0.00 | 0 | 0 | 182 |
| RoBERTa-base (Strategy 1) | 0.05 | 0.02 | 0.03 | 5 | 179 | 23 |
| BERT-base (Strategy 2) | 0.48 | 0.32 | 0.38 | 80 | 117 | 10 |
| BERTweet (Strategy 2) | 0.53 | 0.34 | 0.42 | 94 | 104 | 9 |
| RoBERTa-base (Strategy 2) | 0.54 | 0.33 | 0.41 | 96 | 99 | 12 |

* Strategy 2 models, particularly RoBERTa-base and BERTweet, showed significantly higher precision, recall and F1 in detecting hate language. BERT-base (Strategy 2) also performed reasonably well. In contrast, Strategy 1 models failed to detect any instances of class 0 correctly.
* Strategy 1 models had a high number of false negatives, missing all instances of class 0, which is critical in hate speech detection tasks. Strategy 2 models significantly reduced the number of false negatives, with BERTweet and RoBERTa-base achieving the lowest counts.
* BERTweet and RoBERTa-base from Strategy 2 had the highest true positives and relatively fewer false positives, indicating a more accurate detection capability.

The significant difference in performance between Strategy 1 and Strategy 2 can be attributed to the distinct approaches in training the transformer models. Strategy 1 involved freezing the layers of the transformer model and only training a single-layer classifier with three neurons. This approach resulted in poor performance because the pre-training of these transformer models was not specifically focused on hate speech detection or offensive language.

In contrast, Strategy 2 trained the entire network, allowing the transformer model to learn the linguistic features pertinent to this specific task. Consequently, the transformer model was able to incorporate these features into its embeddings and encodings, resulting in a better differentiation between hate speech and offensive tweets.

As expected, BERT performed adequately among the three models. However, BERTweet, which had been pre-trained on tweets, showed better performance for this task. Ultimately, RoBERTa delivered the best performance in detecting offensive tweets, aligning with its reputation as a stronger model.

# 6. Conclusion and Future Work

Conducting this project has been an enriching experience, providing insights into hate speech detection using transformer models. The process included data collection, model selection, training, evaluation, and analysis. Key challenges involved selecting representative datasets and implementing BERT, BERTweet, and RoBERTa models.

Two training strategies were tested: Strategy 1 froze the transformer layers, while Strategy 2 fine-tuned the entire model. Evaluating these strategies showed the importance of fine-tuning for better performance in detecting hate speech.

The project's primary objective was to develop a reliable hate speech detection system. This was achieved by:

* **Model Evaluation:** Assessing performance using accuracy, precision, recall, and F1 score.
* **Confusion Matrix Analysis:** Understanding classification strengths and weaknesses.
* **Strategy Comparison:** Demonstrating the effectiveness of fine-tuning (Strategy 2) over freezing layers (Strategy 1).

Future work could enhance the project by:

* **Advanced Models:** Using newer transformer models for improved accuracy.
* **Larger Datasets:** Expanding training datasets for better generalizability.
* **Longer Training:** Training over more epochs for refined detection.
* **Multilingual Support:** Adding support for multiple languages.
* **Real-World Testing:** Applying models in real-world scenarios for practical insights.
* **Model Interpretability:** Increasing transparency in model decision-making.

In conclusion, this project successfully developed and evaluated effective hate speech detection models, laying a foundation for future advancements with more sophisticated AI and larger datasets.

# References

|  |  |
| --- | --- |
| 1. | Alkomah, F. and Ma, X., 2022. A literature review of textual hate speech detection methods and datasets. Information, 13(6), p.273. |
| 2. | Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805. |
| 3. | Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. and Stoyanov, V., 2019. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692. |
| 4. | Nguyen, D.Q., Vu, T. and Nguyen, A.T., 2020. BERTweet: A pre-trained language model for English Tweets. arXiv preprint arXiv:2005.10200. |
| 5. | Davidson, T., Warmsley, D., Macy, M. and Weber, I., 2017, May. Automated hate speech detection and the problem of offensive language. In Proceedings of the international AAAI conference on web and social media (Vol. 11, No. 1, pp. 512-515). |
| 6. | Naseem, U., Razzak, I. and Eklund, P.W., 2021. A survey of pre-processing techniques to improve short-text quality: a case study on hate speech detection on twitter. Multimedia Tools and Applications, 80, pp.35239-35266. |

# Appendix

**Data Source and Description**

The dataset used in this study was obtained from a public collection of tweets, which were pre-labeled into three categories: hate speech, offensive language, and neither. The data consisted of 24,783 entries, each containing the following attributes:

* **Count:** An identifier for each entry.
* **Hate Speech:** A binary label indicating whether the tweet was classified as hate speech.
* **Offensive Language:** A binary label indicating whether the tweet was classified as offensive language.
* **Neither:** A binary label indicating whether the tweet was classified as neither.
* **Class:** An integer representing the overall classification of the tweet into one of the three categories.
* **Tweet:** The actual text content of the tweet.
* **Tweet Length:** The length of each tweet.
* **Cleaned Tweet:** The text after removing special characters and other noise.
* **Preprocessed Tweet:** The text after additional lemmatization and stop word removal.

**Data Preprocessing Steps**

This step included creating new columns for cleaned\_tweet and preprocessed\_tweet. The preprocessing steps involved:

1. **Cleaning Tweets:**
   * Removal of retweet indicators ("RT"), mentions, URLs, and special characters.
   * Conversion of text to lowercase and HTML entities.
   * Expansion of contractions using a predefined dictionary.
2. **Tokenization and Stopword Removal:**
   * Tokenization of text into individual words.
   * Elimination of stopwords to reduce noise.
3. **Lemmatization:**
   * Application of lemmatization to reduce words to their base forms.

The preprocessing of tweets involved several detailed steps to ensure the text was clean and suitable for machine learning model input. First, retweet indicators ("RT") were removed, followed by the elimination of user mentions, URLs, and special characters to declutter the text. HTML entities were converted to their corresponding characters, and the text was transformed to lowercase to maintain uniformity.

Next, contractions within the text were expanded using a predefined dictionary, converting phrases like "can't" to "cannot." The text was then tokenized, breaking it down into individual words. Stopwords, which are common words like "and" or "the" that typically do not carry significant meaning, were removed to reduce noise.

Finally, lemmatization was applied, which involves reducing words to their base or root form, ensuring consistency in the text data. For example, words like "running" and "ran" would be lemmatized to "run." This comprehensive preprocessing pipeline utilized the Natural Language Toolkit (NLTK) for efficient text processing and resulted in the creation of two new columns: cleaned\_tweet, containing the sanitized version of the original text, and preprocessed\_tweet, which included the lemmatized and stopword-free version of the text. The preprocessed\_tweet column was then used as the input for the machine learning models.

**Model Selection and Configuration**

* **BERT-base:** A balanced transformer model suitable for general text classification.
* **RoBERTa-large:** An enhanced version of BERT designed for improved performance on various tasks.
* **BERTweet:** Specifically fine-tuned for Twitter data, capturing nuances specific to tweets.

**Sample Data**

Below are sample entries from the dataset, showcasing the original and processed forms:

| **Count** | **Hate Speech** | **Offensive Language** | **Neither** | **Class** | **Tweet (Original)** | **Tweet Length** | **Cleaned Tweet** | **Preprocessed Tweet** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 0 | 0 | 3 | Neither | !!! RT @mayasolovely: As a woman you shouldn't... | 140 | as a woman you shouldnt complain about cleanin... | woman shouldnt complain cleaning house amp man... |
| 3 | 0 | 3 | 0 | Offensive Language | !!!!! RT @mleew17: boy dats cold...tyga dwn ba... | 85 | boy dats coldtyga dwn bad for cuffin dat hoe i... | boy dat coldtyga dwn bad cuffin dat hoe st place |

**Meta Information**

* **Class Distribution:** The dataset was imbalanced, with a majority of tweets classified as offensive language. This was addressed using class weights during training.
* **Tokenization Parameters:** A maximum token length of 128 was chosen to capture essential tweet content without truncation.
* **Training Resources:** Different batch sizes, such as 32, 16, and 4, were tried. While smaller batch sizes showed slight performance improvements, the larger batch size of 64 was chosen due to the limitations of the Google Colab GPU.

These details provide a comprehensive overview of the dataset, preprocessing, model selection, and experimental setup used in the study. This additional information supports the narrative of the report, offering clarity on the methods and rationale behind the AI solution implementation.

**Strategy 1: Freezing Encoding Layers and Fine-Tuning Classification Layer**

This strategy involved freezing the pre-trained transformer model's encoding layers and only fine-tuning the classification layer. This approach reduces computational load and mitigates the risk of overfitting, especially useful when computational resources are limited.

**Implementation Details:**

1. **Constants and Tokenizer Initialization:**

MAX\_LENGTH = 128

BATCH\_SIZE = 64

EPOCHS = 4

NUM\_LABELS = 3

tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')

1. **Tokenization Function:**

def tokenize\_data(texts, tokenizer, max\_length):

    return tokenizer(

        texts.tolist(),

        max\_length=max\_length,

        padding='max\_length',

        truncation=True,

        return\_tensors='tf'

    )

1. **Tokenizing the Data:**

train\_encodings = tokenize\_data(train\_df['preprocessed\_tweet'], tokenizer, MAX\_LENGTH)

val\_encodings = tokenize\_data(val\_df['preprocessed\_tweet'], tokenizer, MAX\_LENGTH)

test\_encodings = tokenize\_data(test\_df['preprocessed\_tweet'], tokenizer, MAX\_LENGTH)

1. **Preparing Datasets:**

train\_labels = train\_df['label'].values

val\_labels = val\_df['label'].values

test\_labels = test\_df['label'].values

train\_dataset = tf.data.Dataset.from\_tensor\_slices((dict(train\_encodings), train\_labels)).shuffle(len(train\_df)).batch(BATCH\_SIZE)

val\_dataset = tf.data.Dataset.from\_tensor\_slices((dict(val\_encodings), val\_labels)).batch(BATCH\_SIZE)

test\_dataset = tf.data.Dataset.from\_tensor\_slices((dict(test\_encodings), test\_labels)).batch(BATCH\_SIZE)

1. **Model Initialization and Freezing Layers:**

model = TFBertForSequenceClassification.from\_pretrained('bert-base-uncased', num\_labels=NUM\_LABELS)

for layer in model.layers[:-1]:

    layer.trainable = False

1. **Model Compilation and Callbacks:**

model.compile(

    optimizer='adam',

    loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

    metrics=['accuracy']

)

callbacks = [

    EarlyStopping(monitor='val\_loss', patience=2, restore\_best\_weights=True),

    ModelCheckpoint('best\_model', save\_best\_only=True, save\_weights\_only=True, save\_format='tf')

]

1. **Model Training:**

history = model.fit(

    train\_dataset,

    validation\_data=val\_dataset,

    epochs=EPOCHS,

    callbacks=callbacks

)

**Strategy 2: Full Fine-Tuning with BERTweet**

The second strategy involved using the BERTweet model, which is pre-trained specifically on Twitter data. This strategy entailed full fine-tuning of the model, adjusting both the encoding and classification layers to optimize performance for the specific task of hate speech detection.

**Implementation Details:**

1. **Preprocessing and Tokenization:**

def strip\_all\_entities(x):

    return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ", x).split())

train\_df['preprocessed\_tweet'] = train\_df['preprocessed\_tweet'].apply(strip\_all\_entities)

val\_df['preprocessed\_tweet'] = val\_df['preprocessed\_tweet'].apply(strip\_all\_entities)

test\_df['preprocessed\_tweet'] = test\_df['preprocessed\_tweet'].apply(strip\_all\_entities)

trn\_sentences = train\_df['preprocessed\_tweet'].values

train\_labels = train\_df['label'].values

val\_sentences = val\_df['preprocessed\_tweet'].values

validation\_labels = val\_df['label'].values

tst\_sentences = test\_df['preprocessed\_tweet'].values

test\_labels = test\_df['label'].values

tokenizer = AutoTokenizer.from\_pretrained('vinai/bertweet-base', use\_fast=False)

def bertweet\_encode(data, max\_len):

    input\_ids = []

    attention\_masks = []

    for i in range(len(data)):

        encoded = tokenizer.encode\_plus(data[i],

                                        add\_special\_tokens=True,

                                        max\_length=max\_len,

                                        padding='max\_length',

                                        truncation=True,

                                        return\_attention\_mask=True)

        input\_ids.append(encoded['input\_ids'])

        attention\_masks.append(encoded['attention\_mask'])

    return np.array(input\_ids), np.array(attention\_masks)

MAX\_LEN = 128

train\_inputs, train\_masks = bertweet\_encode(trn\_sentences, MAX\_LEN)

validation\_inputs, validation\_masks = bertweet\_encode(val\_sentences, MAX\_LEN)

test\_inputs, test\_masks = bertweet\_encode(tst\_sentences, MAX\_LEN)

train\_inputs = torch.tensor(train\_inputs)

validation\_inputs = torch.tensor(validation\_inputs)

test\_inputs = torch.tensor(test\_inputs)

train\_labels = torch.tensor(train\_labels)

validation\_labels = torch.tensor(validation\_labels)

test\_labels = torch.tensor(test\_labels)

train\_masks = torch.tensor(train\_masks)

validation\_masks = torch.tensor(validation\_masks)

test\_masks = torch.tensor(test\_masks)

1. **Creating DataLoader:**

batch\_size = 64

train\_data = TensorDataset(train\_inputs, train\_masks, train\_labels)

train\_sampler = RandomSampler(train\_data)

train\_dataloader = DataLoader(train\_data, sampler=train\_sampler, batch\_size=batch\_size)

validation\_data = TensorDataset(validation\_inputs, validation\_masks, validation\_labels)

validation\_sampler = SequentialSampler(validation\_data)

validation\_dataloader = DataLoader(validation\_data, sampler=validation\_sampler, batch\_size=batch\_size)

test\_data = TensorDataset(test\_inputs, test\_masks, test\_labels)

test\_sampler = SequentialSampler(test\_data)

test\_dataloader = DataLoader(test\_data, sampler=test\_sampler, batch\_size=batch\_size)

1. **Model Initialization and Optimizer Setup:**

model = AutoModelForSequenceClassification.from\_pretrained(

    "vinai/bertweet-base",

    num\_labels=3,

    output\_attentions=False,

    output\_hidden\_states=False,

)

model.cuda()

optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5, eps=1e-8)

epochs = 4

total\_steps = len(train\_dataloader) \* epochs

scheduler = get\_linear\_schedule\_with\_warmup(optimizer, num\_warmup\_steps=0, num\_training\_steps=total\_steps)

1. **Training Loop:**

seed\_val = 42

random.seed(seed\_val)

np.random.seed(seed\_val)

torch.manual\_seed(seed\_val)

torch.cuda.manual\_seed\_all(seed\_val)

training\_stats = []

for epoch\_i in range(0, epochs):

    print("")

    print('======== Epoch {:} / {:} ========'.format(epoch\_i + 1, epochs))

    print('Training...')

    t0 = time.time()

    total\_loss = 0

    model.train()

    for step, batch in enumerate(tqdm(train\_dataloader, desc="Training")):

        b\_input\_ids = batch[0].to(device)

        b\_input\_mask = batch[1].to(device)

        b\_labels = batch[2].to(device)

        model.zero\_grad()

        outputs = model(b\_input\_ids, attention\_mask=b\_input\_mask, labels=b\_labels)

        loss = outputs.loss

        total\_loss += loss.item()

        loss.backward()

        torch.nn.utils.clip\_grad\_norm\_(model.parameters(), 1.0)

        optimizer.step()

        scheduler.step()

    avg\_train\_loss = total\_loss / len(train\_dataloader)

    print("")

    print("  Average training loss: {0:.2f}".format(avg\_train\_loss))

    print("  Training epoch took: {:}".format(format\_time(time.time() - t0)))

    print("")

    print("Running Validation...")

    t0 = time.time()

    model.eval()

    eval\_accuracy = 0

    nb\_eval\_steps = 0

    total\_eval\_loss = 0

    for batch in tqdm(validation\_dataloader, desc="Validating"):

        batch = tuple(t.to(device) for t in batch)

        b\_input\_ids, b\_input\_mask, b\_labels = batch

        with torch.no\_grad():

            outputs = model(b\_input\_ids, attention\_mask=b\_input\_mask, labels=b\_labels)

        loss = outputs.loss

        total\_eval\_loss += loss.item()

        logits = outputs.logits

        logits = logits.detach().cpu().numpy()

        label\_ids = b\_labels.to('cpu').numpy()

        tmp\_eval\_accuracy = flat\_accuracy(logits, label\_ids)

        eval\_accuracy += tmp\_eval\_accuracy

        nb\_eval\_steps += 1

    avg\_val\_accuracy = eval\_accuracy / nb\_eval\_steps

    avg\_val\_loss = total\_eval\_loss / len(validation\_dataloader)

    print("  Accuracy: {0:.2f}".format(avg\_val\_accuracy))

    print("  Validation took: {:}".format(format\_time(time.time() - t0)))

    training\_stats.append(

        {

            'epoch': epoch\_i + 1,

            'Training Loss': avg\_train\_loss,

            'Valid. Loss': avg\_val\_loss,

            'Valid. Accur.': avg\_val\_accuracy,

            'Training Time': format\_time(time.time() - t0)

        }

    )

print("")

print("Training complete!")

1. NLP [↑](#footnote-ref-1)
2. Bidirectional Encoder Representations from Transformers [↑](#footnote-ref-2)
3. Robustly Optimized BERT Approach [↑](#footnote-ref-3)
4. Term Frequency-Inverse Document Frequency [↑](#footnote-ref-4)
5. Convolutional Neural Networks [↑](#footnote-ref-5)
6. Recurrent Neural Networks [↑](#footnote-ref-6)
7. NLU [↑](#footnote-ref-7)
8. Hate Speech and Offensive Language [↑](#footnote-ref-8)
9. EDA [↑](#footnote-ref-9)