Emotion-Aware Hate Speech Detection in User-Generated Content Using Natural Language Processing

By

Tang Yu Tong

TP069018

APU3F2408CS(DA)

A report submitted in partial fulfillment of the requirements for the degree of

B.Sc. (Hons) Computer Science Specialism in Data Analytics

at Asia Pacific University of Technology and Innovation.

Supervised by Mr. Nicholas Teh Sek Kit

2nd Marker: Mr. Raheem Mafas

2025

DECLARATION OF THESIS CONFIDENTIALITY

Author's full name: **TANG YU TONG**

IC No./Passport No.: **031030-01-1504**

Thesis/Project title: **Emotion-Aware Hate Speech Detection in User-Generated Content Using Natural Language Processing**

I declare that this thesis is classified as:

☐ CONFIDENTIAL

☐ RESTRICTED

☑ OPEN ACCESS

I acknowledged that Asia Pacific University of Technology & Innovation (APU) reserves the right as follows:

1. The thesis is the property of Asia Pacific University of Technology & Innovation (APU).
2. The Library of Asia Pacific University of Technology & Innovation (APU) has the right to make copies for the purpose of research only.
3. The library has the right to make copies of the thesis for academic exchange.

Author's Signature: yutong

Date: 15 April 2025

Supervisor's Name: **MR. NICHOLAS TEH SEK KIT**

Date: 21 April 2025

Signature:

LIBRARY FORM

First Name: Tang
Middle Name (only if applicable): Yu
Last Name: Tong
Title of the Final Year Project / Dissertation / Thesis: Emotion-Aware Hate Speech Detection in User-Generated Content Using Natural Language Processing
Abstract: With the rapid development of online platforms, there is an increasing demand for powerful hate speech detection technology to ensure the safety of the online environment. However, existing models often ignore implicit hate speech. This study will detect hate speech by incorporating emotion features into the natural language processing (NLP) model. The project follows CRISP-DM methodology. Text preprocessing techniques include label mapping, text cleaning and text normalization. By comparing the traditional machine (ML) learning model, deep learning (DL) model and integrate emotion feature model, to test whether the integration of emotion features can help the model improve the performance of detecting hate speech. This project is in line with the 16th United Nations Sustainable Development Goal (SDG), which advocates peace, justice and strong institutions.
A few keywords associated with the work: Hate Speech Detection, Natural Language Processing (NLP), CRISP-DM, Text Pre-Processing, Deep Learning Model, Focal Loss Function.
General Subject: Computer Science Data Analytics
Date of Submission: 30 April 2025

ACKNOWLEDGEMENT

I want to thank Mr. Nicholas Teh Sek Kit and Mr. Raheem Mafas, my supervisor, for his constant assistance and direction during my final year project. His constructive feedback and insightful suggestions have been instrumental in shaping the quality and direction of this project. The time and effort he dedicated to reviewing the progress and providing valuable insights are greatly appreciated.

My appreciation also goes to Dr. Dewi Octaviani, the project manager of the final year projects. Her briefings clear instructions and well-curated reference materials have significantly contributed to the successful completion of this investigation report.

Lastly, thanks to my family and friends for their constant support, which has been my rock during this journey. Their encouragement and understanding have motivated me to persevere during challenging times and to achieve this milestone. This project would not have been possible without their belief in my capabilities.

ABSTRACT

With the rapid development of online platforms, there is an increasing demand for powerful hate speech detection technology to ensure the safety of the online environment. However, existing models often ignore implicit hate speech. This study will detect hate speech by incorporating emotion features into the natural language processing (NLP) model. The project follows CRISP-DM methodology. Text preprocessing techniques include label mapping, text cleaning and text normalization. By comparing the traditional machine (ML) learning model, deep learning (DL) model and integrate emotion feature model, to test whether the integration of emotion features can help the model improve the performance of detecting hate speech. This project is in line with the 16th United Nations Sustainable Development Goal (SDG), which advocates peace, justice and strong institutions.

Keywords **– Hate Speech Detection, Natural Language Processing (NLP), CRISP-DM, Text Pre-Processing, Deep Learning Model, Focal Loss Function.**

SDG Goal 16: Promote just, peaceful and inclusive societies. Promoting peaceful and inclusive societies, providing access to justice for all and building effective, accountable and inclusive institutions at all levels.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Introduction

New social media platforms like Reddit and Twitter allow people to freely, quickly, and broadly connect with others in our daily lives. Social media may also be used to spread misleading information. This misinformation can be aggressive, radical, or even phony. This is because social media is somewhat anonymous. The so-called "hate speech," which communicates hate feeling on social media, is one of the most typical patterns among the different negative information patterns. Hate speech is a complicated phenomenon that is mostly exclusive to connections between groups, in contrast to generic emotions.

Hate speech has been shown to have profound psychological and emotional impacts, often leading to trauma-like symptoms in its victims. Studies using the crisis reaction model reveal that hate speech triggers a cycle of emotional, cognitive, and behavioural responses similar to those caused by other traumatic events. Post-traumatic stress disorder (PTSD) symptoms, including pain, dread, anxiety, nightmares, and recurring intrusive ideas of dehumanization and intimidation, are frequently experienced by victims. Even individuals who are not direct targets of hate speech, particularly in environments like college campuses, may experience stress, lowered self-esteem, and social isolation. Prejudice, discrimination, and intolerance create a hostile atmosphere, affecting the mental well-being and social integration of minority groups. Hate speech, therefore, poses not only a threat to personal safety but also contributes to long-term psychological harm and societal division. (Saha et al., 2019)

Numerous online forums, including Facebook, YouTube, and Twitter, have standards in place to delete anything that is considered hate speech. The research of automatic hate speech identification is highly motivated by societal concerns and the growing amount of hate speech on the internet. Hateful content may be stopped from spreading by automating its identification, which contributes to the development of a safer online community. However, identifying hate speech is a difficult process. There are differences of opinion over the definition of hate speech. This suggests that different people may view different information as hate speech depending on how each word is employed. (MacAvaney et al., 2019)

Due to the significant social issues caused by the spread of hate speech, this research paper proposes a deep learning approach incorporating emotional features to improve detection accuracy. Strong negative emotions including fear, contempt, rage, and despair are frequently the root causes of hate speech. By leveraging emotion classification, which automatically categorizes text into various emotional categories, this study aims to enhance the detection of hate speech. Integrating emotional awareness into the model offers a more nuanced understanding of the content, ultimately contributing to more effective hate speech moderation and fostering a more peaceful and just society SDG 16 which is peace, justice, and strong institutions.

1.2 Problem Background

The rise of online hate speech has emerged as a pressing issue, significantly exacerbated by the proliferation of social media and other internet platforms. High-profile extremist attacks, such as the 2015 shooting at Emanuel African Methodist Episcopal Church, the 2019 Walmart attack in El Paso, and the 2022 shooting at a nightclub in Colorado Springs, starkly illustrate the severe and often tragic consequences of unchecked online hate speech. This pervasive problem manifests in numerous forms, including derogatory comments targeting individuals based on race, nationality, ethnicity, gender, gender identity, religion, disability, and sexual orientation. Research indicates that up to one-third of internet users have encountered such harmful content, with the prevalence even higher in online gaming communities, where approximately 50% of users report experiencing hate speech (Office, 2024).

1. Data Analytics Challenges in Imbalanced Distribution of Hate Speech Dataset

Challenges in detecting hateful content arise from imbalanced distribution in the hate speech data (Kovács et al., 2021) (MacAvaney et al., 2019). While harmful speech is a serious problem, hate speech accounts for only a small fraction of social media interactions. This data imbalance is also reflected in hate speech corpora, such as the HASOC 2019 training set, where less than 40% of tweets are classified as hateful or offensive. As a result, the scarcity of hate speech instances poses difficulties for machine learning models, which struggle to accurately detect minority groups. While standard resampling methods exist to address this issue, effectively mitigating their effects remains a key challenge to ensure accurate and fair detection of hate speech (Khullar et al., 2023).

2. Challenges of Keyword-Based Approaches in Identifying Hate Speech

Basic text analysis methods often fail to capture the nuances of hate speech. Non-profane hate speech targets a specific group with the intention of harming them but does not use explicitly offensive language, which is a huge challenge for traditional text analysis models because many current models are based on keyword-based detection methods (MacAvaney et al., 2019). On the other hand, profane language is not always a sign of malicious intent. It can be used in a variety of contexts, such as joking or enthusiastically, which can lead to false positives. For example, comments such as "he's fking a genius" and "he's fking an idiot" may be incorrectly labelled as hate speech, especially when such language is used casually (Kovács et al., 2021).

3. Complexity of Hate Speech in Text Analysis

To evade detection methods based on hate speech keywords, users often use obfuscation techniques, such as replacing letters with similar numbers, or using creative spellings to bypass automatic review systems, such as replacing "E" with 3, "l" with 1, and so on (De la Peña Sarracén & Rosso, 2023). In addition, hate speech content may be embedded in seemingly harmless statements, making it difficult to identify. For example, a comment such as "1 re@lly x like b1@ck ppl" may hide some underlying hostility that is not easy to detect, due to the word are being replaced by using number or a symbol (Kovács et al., 2021).

Conclusion of the problem statement

In conclusion, the relatively low frequency of hate speech within the overall data presents a challenge for detection models, which struggle with imbalanced data. Additionally, the complexity of hate speech in text analysis further complicates detection, as users often evade keyword-based methods through character substitution and subtle language, reducing the effectiveness of these approaches. These problems highlight the need for more advanced techniques to identify and address online hate speech accurately.

1.3 Project Aim

This project aims to develop a hate speech detection model that integrates emotional features using natural language processing techniques to improve accuracy and context understanding in user-generated content.

1.4 Objectives

1. To investigate and obtain comprehensive datasets for emotion feature extraction and hate speech detection to ensure effective model training.
2. To develop, train, and deploy an advanced NLP model capable of detecting hate speech by incorporating emotion analysis, overcoming the limitations of traditional keyword-based and basic text analysis methods, and implementing it in a local web page for practical use.
3. To evaluate the performance of the developed model, with the goal of achieving and sustaining an accuracy rate above 80% in detecting hate speech, ensuring reliability and effectiveness in diverse real-world applications.

1.5 Scope

1.5.1 Deliverable

1. Data Understanding

   A comprehensive report detailing the data collection and exploration processes for both hate speech and sentiment datasets, highlighting the sources, criteria for selection, and initial insights gained from the data analysis.

2. Data Preprocessing Pipeline

   A documented preprocessing pipeline outlining the steps taken, including data cleaning (removal of symbols and stop words), normalization (lowercasing), stemming, and tokenization to prepare the datasets for analysis.

3. Text Analysis and Natural Language Processing Techniques

   A summary of the natural language processing (NLP) techniques and text analysis methods employed, including sentiment analysis and emotion classification frameworks, tailored for detecting hate speech.

4. Trained Emotion-Aware Deep Learning Model

   A fully trained and fine-tuned deep learning model capable of emotion-aware hate speech detection, which incorporates emotion to enhance detection performance. The model will undergo rigorous testing and validation.

5. Performance Evaluation Report

   A detailed report evaluating the performance of the emotion-aware hate speech detection model, employing metrics such as accuracy, precision, recall, F1 score, and confusion matrix analysis to provide a thorough assessment of the model's capabilities.

6. Model Deployment and Visualization Tools

   A deployed version of the trained model along with interactive data visualizations, including charts and graphs that illustrate key findings, such as the correlation between emotional states and hate speech, trends in emotional language across user-generated content, and model performance metrics.

1.5.2 Constraint

1. Data Privacy Compliance

   This project will exclusively utilize publicly available datasets to ensure compliance with privacy regulations. Sources will include platforms such as GitHub and Kaggle. By sourcing data from platforms like GitHub and Kaggle, which provide curated and accessible datasets for research, the project mitigates potential legal and ethical concerns surrounding data privacy.

2. Resource Accessibility

   Access to certain journal articles may be restricted due to user profile limitations, which could affect the availability of relevant research.

3. Use of Existing Models

   The project will leverage established open-source machine learning models and algorithms, negating the necessity to develop these components from the ground up. Using proven, well-documented models helps streamline the development process, ensure reliability, and allows the project to focus on customization and optimization rather than foundational model development.

4. Language Limitation

The detection models will be designed to process content exclusively in the English language. Focusing on a single language enhances the model's accuracy and effectiveness, as it allows for a deeper understanding of English-specific linguistic nuances, slang, and cultural context. While expanding to multiple languages could be valuable, it would introduce challenges related to data availability and increase model complexity, potentially impacting overall accuracy. Additionally, English is the predominant language used on the Internet and serves as a common global language, making it a practical and impactful choice for this project.

5. Accuracy Prioritization

   Prioritizing accuracy over speed allows the model to process each input more thoroughly, reducing the risk of false positives and negatives, which is critical for sensitive content like hate speech. Real-time processing capabilities may be considered in future iterations once accuracy benchmarks are met.

1.5.3 Project Boundaries

Will be done in this project:

1. The dataset used for this project will be gathered from publicly available hate speech and emotion-related datasets on GitHub and Kaggle.
2. Cleaning and preprocessing the collected data to remove noise and irrelevant content.
3. Developing a detection model to identify hate speech in English content, with a focus on general human-generated data and incorporating emotional analysis for better context understanding.
4. Conducting training and testing to evaluate the model's accuracy, precision, recall, and F1 score.
5. Creating a simple local webpage where users can input text and directly test the performance of the detection models.

Will not be done in this project:

1. The system will only support hate speech detection for English language content and achieving complete detection accuracy is not feasible. The project will only aim to minimize false positives due to the complexity of human language and the continuous evolution of new words.

2. This project will not include the development of a mobile application for detection, limiting its usage to the local web interface.
3. The system will not include real-time monitoring and detection as this project focuses solely on building and training a model to identify hate speech. The deployment aspect is limited to demonstrating the trained model's performance.
4. The project will concentrate exclusively on identifying hate speech without delving into a detailed emotional analysis and detection. For instance, it will not explore the reasons behind specific emotions but will instead emphasize detecting the emotional features integrated into hate speech. A comprehensive emotional analysis would demand significant time, resources, specialized datasets, and advanced modeling techniques. By prioritizing hate speech detection, the project remains focused and ensures the achievement of primary objectives within the given time.

1.6 Potential Benefits

1.6.1 Tangible benefits

For Platform Providers

1. Enhanced Safety: By detecting and labelling instances of hate speech, platforms can take prompt action, such as removing harmful content or issuing warnings to users, to help prevent sensationalism caused by inappropriate speech.
2. Resource Efficiency: Automated detection of hate speech can significantly reduce the need for manual content reviews, leading to time savings and decreased labour costs for platforms.
3. Legal compliance: Implementing hate speech detection can help platforms comply with legal requirements regarding harmful content, reducing the risk of penalties.

For Researchers

1. Insights from Data Analysis: Analysing hate speech patterns through the model can provide valuable insights, which can be used to understand the underlying issues and trends related to hate speech.

For Users

1. Increased User Retention: Actively addressing hate speech can attract users to spend more time on the platform.

1.6.2 Intangible benefits

For Platform Providers

1. Positive reputation: Effectively managing hate speech contributes to a positive public image, attracting more users and advertisers who prioritize a safe online community.

For Users

1. User Mental Health: Reducing hate speech can protect the mental health of all users by preventing feelings of being targeted or excluded, thereby fostering a more inclusive online environment.
2. Improved User Experience: Platforms that actively reduce hate speech can create a more pleasant and engaging environment for users, increasing user satisfaction and loyalty.

1.6.3 Target User

1. Researchers (Individual)

   Researchers can use the system to analyze patterns of hate speech, advancing their work in natural language processing and contributing to developing more robust detection models, which help gain deeper insights into online hate speech trends and improve strategies.
2. Software Engineers (Developers)

   Software engineers can integrate the model into platforms to enhance automated hate speech detection. This reduces the need for manual moderation and contributes to safer online communities, improving platform efficiency and user satisfaction.

1.7 Overview of the FYP Documentation

This Final Year Project (FYP) report is organized into four chapters, each addressing a distinct aspect of the research, ranging from the project background to the research results. In Semester 2, the focus will be on building, training, testing, and evaluating the model.

Chapter 1, Project Introduction. This chapter describes the reasons for choosing this FYP topic, points out the issues that need further exploration in the current research, and clarifies the goals and scope of the project. In addition, this chapter discusses the target users and potential benefits of the project and finally gives a detailed schedule for the entire project.

Chapter 2, Literature Review, conducts in-depth research on relevant literature and studies the algorithms needed for the project, laying the foundation for the project. And provide a section evaluates the similar work, highlighting those research paper strengths and limitations. Next, the technology study section outlines potential system requirements and technology requirements.

Chapter 3, Methodology, this project uses the CRISP-DM Methodology, which includes business understanding, data understanding, data preparation, modelling, evaluation and deployment. This chapter also compares the KDD methodology and SEMMA methodology and explains why this project is suitable for the CRISP-DM Methodology.

Chapter 4, Text Preprocessing and Model Building, explains the source of the data and the steps of text data preprocessing. It further demonstrates the methods and algorithms used in building machine learning, deep learning, and multi-task learning models. Each selected algorithm is explained, along with a detailed justification of their hyperparameters and model tuning techniques.

Chapter 5, Discussion of Results and Model Deployment, evaluates the performance of each model through various metrics and charts. In-depth comparisons are made of the basic performance, hyperparameter tuning results, and fine-tuning results of each model. Special focus on models that integrate emotional features and multi-task capabilities. In addition, this chapter introduces model deployment and a dashboard of model performance and hate speech dataset, providing users with a simple input prediction hate speech user interface.

Chapter 6 summarizes the contributions of the project, highlighting project successfully achieved stated objectives and aligned with the goals of Sustainable Development Goal (SDG) 16, Peace, Justice, and Strong Institutions. This chapter also reflects on the limitations and challenges encountered in the project and provides several recommendations for future improvement.

1.8 Project Plan

Semester 1

Name of Tasks	Duration	Start Date	End Date	Status
Project Proposal Form	**20 Days**	**18-Sep-24**	**7-Oct-24**	**Done**
Fast-Track Form	**2 Days**	**11-Oct-24**	**12-Oct-24**	**Done**
Chapter 1: Introduction	**4 Days**	**19-Oct-24**	**22-Oct-24**	**Done**
1.1 Introduction	< 1 day	19-Oct-24	19-Oct-24	Done
1.2 Problem Background	< 1 day	19-Oct-24	19-Oct-24	Done
1.3 Project Aim	< 1 day	20-Oct-24	20-Oct-24	Done
1.4 Objective	< 1 day	20-Oct-24	20-Oct-24	Done
1.5 Scope	< 1 day	21-Oct-24	21-Oct-24	Done
1.5.1 Deliverable	< 1 day	21-Oct-24	21-Oct-24	Done
1.5.2 Constraint	< 1 day	21-Oct-24	21-Oct-24	Done
1.5.3 Project Boundaries	< 1 day	21-Oct-24	21-Oct-24	Done
1.6 Potential Benefit	< 1 day	22-Oct-24	22-Oct-24	Done
1.6.1 Tangible Benefit	< 1 day	22-Oct-24	22-Oct-24	Done
1.6.2 Intangible Benefit	< 1 day	22-Oct-24	22-Oct-24	Done
1.6.3 Target User	< 1 day	22-Oct-24	22-Oct-24	Done
1.7 Overview of the IR	< 1 day	22-Oct-24	22-Oct-24	Done
1.8 Project Plan	< 1 day	22-Oct-24	22-Oct-24	Done
Chapter 2: Literature Review	**9 Days**			
2.1 Introduction	< 1 day	23-Oct-24	23-Oct-24	Done
2.2. Domain Research	7 days	23-Oct-24	29-Oct-24	Done
2.3 Similar Systems/ Works	2 days	30-Oct-24	31-Oct-24	Done
2.4 Technical Research	3 days	1-Nov-24	3-Nov-24	Done
2.5 Summary	< 1 day	3-Nov-24	3-Nov-24	Done
Chapter 3: Methodology	**14 Days**			
3.1 Introduction	< 1 day	4-Nov-24	4-Nov-24	Done
3.2 Methodology	< 1 day	4-Nov-24	4-Nov-24	Done
3.2.1 Introduction of Methodology	< 1 day	4-Nov-24	4-Nov-24	Done
3.2.2 Methodology of choice and justifications	< 1 day	4-Nov-24	4-Nov-24	Done
3.2.3 Phases of Methodology	3 days	5-Nov-24	7-Nov-24	Done
3.3 Data Collection	2 days	8-Nov-24	9-Nov-24	Done
3.4 Initial Data Pre-processing and Data Understanding				
3.4.1 Data Pre-processing	6 days	10-Nov-24	15-Nov-24	Done
3.4.2 Data Understanding	3 days	15-Nov-24	17-Nov-24	Done
3.5 Summary	< 1 day	17-Nov-24	17-Nov-24	Done
Chapter 4: Conclusion	**2 Days**			
4.1 Achievements of the Project	< 1 day	21-Nov-24	21-Nov-24	Done
4.2 Research Investigation Justification	< 1 day	21-Nov-24	21-Nov-24	Done
4.3 Research and Design Gaps	< 1 day	21-Nov-24	21-Nov-24	Done
Acknowledgement Page	**< 1 day**	**18-Nov-24**	**18-Nov-24**	**Done**
Abstract Page	**< 1 day**	**21-Nov-24**	**21-Nov-24**	**Done**
References	**< 1 day**	**21-Nov-24**	**21-Nov-24**	**Done**
Appendies	**< 1 day**	**21-Nov-24**	**21-Nov-24**	**Done**

Figure 1: Project Plan Semester 1

Semester 2

Name of Tasks	Duration	Start Date	End Date	Status
Project Proposal Form	20 Days	18-Sep-24	7-Oct-24	Done
Fast-Track Form	2 Days	11-Oct-24	12-Oct-24	Done
Acknowledgement Page	< 1 day	18-Nov-24	18-Nov-24	Done
Abstract Page	< 1 day	19-Apr-25	19-Apr-25	Done
Chapter 1: Introduction				
1.1 Introduction	< 1 day	19-Oct-24	19-Oct-24	Done
1.2 Problem Background	< 1 day	19-Oct-24	19-Oct-24	Done
1.3 Project Aim	< 1 day	20-Oct-24	20-Oct-24	Done
1.4 Objective	< 1 day	20-Oct-24	20-Oct-24	Done
1.5 Scope	< 1 day	21-Oct-24	21-Oct-24	Done
1.5.1 Deliverable	< 1 day	21-Oct-24	21-Oct-24	Done
1.5.2 Constraint	< 1 day	21-Oct-24	21-Oct-24	Done
1.5.3 Project Boundaries	< 1 day	21-Oct-24	21-Oct-24	Done
1.6 Potential Benefit	< 1 day	22-Oct-24	22-Oct-24	Done
1.6.1 Tangible Benefit	< 1 day	22-Oct-24	22-Oct-24	Done
1.6.2 Intangible Benefit	< 1 day	22-Oct-24	22-Oct-24	Done
1.6.3 Target User	< 1 day	22-Oct-24	22-Oct-24	Done
1.7 Overview of the IR	< 1 day	22-Oct-24	22-Oct-24	Done
1.8 Project Plan	< 1 day	22-Oct-24	22-Oct-24	Done
Chapter 2: Literature Review				
2.1 Introduction	< 1 day	23-Oct-24	23-Oct-24	Done
2.2. Domain Research	7 days	23-Oct-24	29-Oct-24	Done
2.3 Similar Systems/ Works	2 days	30-Oct-24	31-Oct-24	Done
2.4 Technical Research	3 days	1-Nov-24	3-Nov-24	Done
2.5 Summary	< 1 day	3-Nov-24	3-Nov-24	Done
Chapter 3: Methodology				
3.1 Introduction	< 1 day	4-Nov-24	4-Nov-24	Done
3.2 Methodology	< 1 day	4-Nov-24	4-Nov-24	Done
3.2.1 Introduction of Methodology	< 1 day	4-Nov-24	4-Nov-24	Done
3.2.2 Methodology of choice and justifications	< 1 day	4-Nov-24	4-Nov-24	Done
3.2.3 Phases of Methodology	3 days	5-Nov-24	7-Nov-24	Done
3.3 Summary	< 1 day	10-Nov-24	10-Nov-24	Done
Chapter 4: Conclusion				
4.1 Introduction	< 1 day	8-Nov-24	8-Nov-24	Done
4.2 Data Collection	2 days	8-Nov-24	9-Nov-24	Done
4.3 Exploratory Data Analysis (EDA)	3 days	10-Nov-24	12-Nov-24	Done
4.4 Data Preprocessing	8 days	12-Nov-24	19-Nov-24	Done
4.5 Data Understanding	< 1 day	20-Nov-24	20-Nov-24	Done
4.6 Model Building - Hate Speech	30 days	22-Feb-25	23-Mar-25	Done
4.7 Model Building - Hate Speech model Intergrate Emotion Feature	9 days	23-Mar-25	31-Mar-25	Done
4.8 Summary	< 1 day	14-Apr-25	14-Apr-25	Done
Chapter 5: Result and Discussion				
5.1 introduction	< 1 day	23-Mar-25	23-Mar-25	Done
5.2 Model Evaluation and Discussion	13 days	23-Mar-25	4-Apr-25	Done
5.3 Model Deplovemnt	5 days	11-Apr-25	15-Apr-25	Done
5.4 Summary	< 1 day	16-Apr-25	16-Apr-25	Done
Chapter 6: Conclusion				
6.1 Critical Evaluation	< 1 day	17-Apr-25	17-Apr-25	Done
6.2 Limitation of this Project	< 1 day	18-Apr-25	18-Apr-25	Done
6.3 Recommendation for the Future Work	< 1 day	18-Apr-25	18-Apr-25	Done
References	< 1 day	18-Apr-25	18-Apr-25	Done
Appendies	< 1 day	18-Apr-25	18-Apr-25	Done

Figure 2: Project Plan Semester 2

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

Chapter 2, the Literature Review, provides a comprehensive exploration of research paper related to hate speech detection with integrated emotion analysis. The chapter begins with an overview of domain research, delving into the complexities and definitions of hate speech, especially in the context of unstructured data. Key topics include feature extraction techniques like TF-IDF and advanced embeddings such as Word2Vec, FastText and BERT, which are fundamental in capturing the nuanced nature of hate speech. The chapter also covers various text classification techniques, including traditional classifiers and deep learning models, such as XGBoost, Random Forest and LSTM, and what are the challenges and contribute in hate speech detection and why key-based models are not suitable in detecting subtle hate speech.

Additionally, this chapter examines ensemble learning and multi-task learning approaches, which integrate multiple datasets to improve model robustness and provides eight similar works with list out each research paper's limitations, challenges, gaps and contributions in hate speech detection. The technical research section further outlines the specific tools and technologies chosen for this project, including the programming language, IDE, libraries, operating systems, and web browsers.

2.2. Domain Research

2.2.1 Definition of Hate Speech

The meaning of hate speech is still up for controversy and lacks a uniform or widely recognized definition. The absence of agreement makes it more difficult to identify and control hate speech, particularly in legal and online settings. There are several ways to distinguish this type of speech from free expression. This study of the literature critically looks at several definitions of hate speech, emphasizes how difficult it is to define the term, and describes the technological difficulties in identifying hate speech, especially on digital platforms.

Based on research from (MacAvaney et al., 2019), the Encyclopedia of the American Constitution defines hate speech as speech that attacks individuals based on specific attributes like race or religion. In contrast, Facebook and Twitter provide more specific definitions,

focusing on hate speech as direct attacks on groups or individuals based on protected characteristics such as race, gender identity, and religious affiliation.

Hietanen and Eddebo (2022) argue that defining hate speech solely from an emotional perspective risk limiting it from political discourse. If any speech that evokes negative emotions is categorized as hate speech, the definition may become too broad, thereby limiting political discourse. Therefore, a more cautious approach is advocated, warning that over-reliance on emotional responses may inadvertently stifle speech that is both valid and controversial.

In conclusion, the subjective character of hate speech and the absence of a widely recognized definition make it an elusive idea. It is challenging to draw distinct lines between hate speech and free expression because of the ambiguity surrounding its definition. From a critical point of view, finding the right balance between regulating hate speech and protecting free speech is a complex process. Any definition of hate speech must consider factors like intent, who the speech is directed at, and its impact on society. If we regulate too much or have overly broad definitions, we risk limiting free speech. But if we regulate too little, harmful and discriminatory language might spread. This shows the importance of definitions and detection methods that take context into account the difference between hate speech and strong but legitimate opinions.

2.2.2 Unstructured Data Pre-Processing

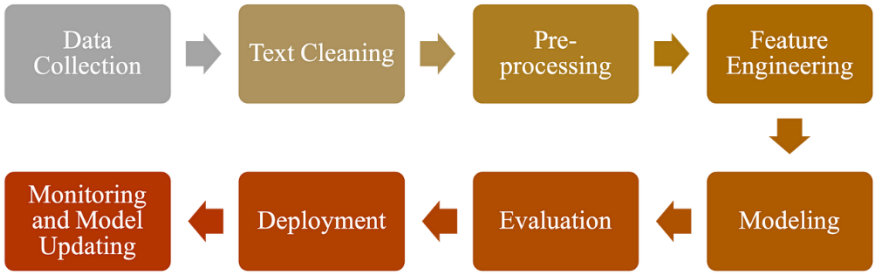


Figure 3: Natural Language Processing Pipeline (Dube, 2023)

Text preprocessing includes lowercasing all text for consistency, removing HTML tags and URLs, eliminating punctuation, processing chatter or slang, spelling correction, removing stop words (such as a, the, of, are, etc.), removing emoticons, tokenizing text into meaningful units, and stemming word forms to their root form (Arslan, 2024).

2.2.3 Method to Handle Imbalanced Dataset

Resampling Method

Some common techniques involved in balancing data distribution include oversampling and undersampling, according to (Chen et al., 2024). In oversampling, synthesized samples of the minority class are generated to balance its representation within the dataset. Undersampling decreases the number of samples that exist in the class of the majority. The research paper (Yu & Zhou, 2021) used different SMOTE methods for oversampling, including SMOTE, AdaSyn, SMOTE-ENN and SMOTE-Tomek. All methods produced similar performance results, especially in terms of AUC scores. The AUC score of the logistic regression model improved from 85.08% to 87% and the linear SVC model improved from 83.33% to 86%.

Loss Modification

Algorithm-level methods focus on modifying algorithms to make them more suited for imbalanced data without altering the dataset. For example, include adjusting classifier decision thresholds.

Focal Loss is a loss function that specifically addresses data imbalance. It is different from traditional cross entropy loss, which treats all misclassifications equally. Focal Loss introduces an adjustment factor that reduces the loss weight of majority class samples while focusing on minority classes and misclassified data (Yesi Novaria Kunang et al., 2021). For example, when training a classification model, if 90% of the data is labelled “non-hate” and only few 10% is “hate”, the ordinary loss function may directly predict all results as “non-hate” because the overall error rate is the lowest, but the model learns nothing. Focal loss uses alpha and gamma to adjust the loss weight of prediction errors and minority classes. Focal Loss optimizes the model during the training phase, and threshold adjustment is the decision during the inference phase.

The paper (Jiang et al., 2024) mentioned that adjusting and finding the optimal threshold is time-consuming. Since the threshold cannot be adjusted dynamically during the training phase, many repeated experiments are required to determine the threshold. Adjusting the threshold means that if the original prediction value is greater than 0.5 is hate speech, but to make the model pay more attention to a few categories, the threshold can be lowered to 0.3. Contrariwise, while a high threshold has the effect of reducing false positives, a small one can make it easier

for the model to classify samples into a certain category but will lead to more false positives. A static asymmetric sample mining strategy (ASMS) and its dynamic version, adaptive tolerance ASMS (AT-ASMS), are designed by (Jiang et al., 2024) specifically for sample mining methods. ASMS solves the problem of too few positive samples and too many redundant negative samples caused by filtering samples with a single threshold by using differentiated thresholds.

Class Weights

Weighting the classes of different categories in cost-sensitive learning allows one to make the model pay more attention to fewer categories during the training process, especially those with a small number but at high error cost. This weight set is to make the model more sensitive to minority class samples on hate speech in unbalanced data and reduce the misclassification of these samples.

In summary, whether it is undersampling, oversampling, class weight, or changing the threshold, decide to use which one is based on different models and dataset quality. If the imbalance of some dataset is very serious or the fluctuation is large, can choose to use a hybrid method that combines both technical and algorithm advantages. For example, it can generate synthetic samples by oversampling the minority class and train it with a cost-sensitive classifier (Chen et al., 2024).

2.2.4 Feature Extraction

Feature extraction is an essential step in converting text into numerical data representations. There are some methods for feature extraction, including some traditional methods like N-grams and TF-IDF and advanced word embedding techniques such as Word2Vec, Glove, FastText and BERT. These methods are commonly used to capture linguistic features.

2.2.4.1 TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF is a commonly used technique in NLP and information retrieval. It measures the importance of a word about a specific document and the entire document set.

Term Frequency (TF) to calculate a word that appears in a document relative to the total word count. It gives more weight to frequently used words within that document (Karabiber, n.d.).

$$TF = \frac{number\ of\ time\ the\ term\ appears\ in\ the\ document}{total\ number\ of\ term\ in\ the\ document}$$

Inverse Document Frequency (IDF) is calculated based on the entire corpus (all document collections) and reflects how common or rare a word is across all documents. Words that appear in only a few documents, such as specialized terms, are given more weight, while common words that appear everywhere, like "the" or "and," have lower importance (Karabiber, n.d.).

$$IDF = \log(\frac{number\ of\ the\ documents\ in\ the\ corpus}{number\ of\ documents\ in\ the\ corpus\ contain\ the\ term})$$

$$TF - IDF = TF * IDF$$

In each document, a word with a higher TF-IDF score is considered more essential. That is, the word appears more often in that document, but is less common in other documents, making it more discriminative. A lower TF-IDF value means that the word is less important in that document but may be common in many documents.

TF-IDF plays a valuable role in hate speech detection by helping to identify words that are more significant in distinguishing harmful language. Hate speech often contains specific terms or phrases that carry a targeted negative connotation, and TF-IDF can assign higher importance to such terms if they appear frequently in hate speech instances but are rare in neutral or non-harmful content.

2.2.4.2 Word Embedding Techniques

According to (Allen & Hospedales, 2019) research paper shows that the word embeddings techniques are widely used in deep learning models, especially in NLP. Word embedding represents words as numerical vectors, allowing computers to process words more efficiently. Word embedding methods can learn the semantic relationships between words and map the words into a low-dimensional, dense vector space. Compared to one-hot encoding, this method is more efficient in capturing word meanings and reduces computational and storage costs.

2.2.4.2.1 Word2Vec

Skip-gram and Continuous Bag of Words (CBOW) are the two primary models that Word2Vec uses. In Natural Language Processing (NLP), these models were developed to effectively capture word meanings and can be easily transferred to different downstream tasks.

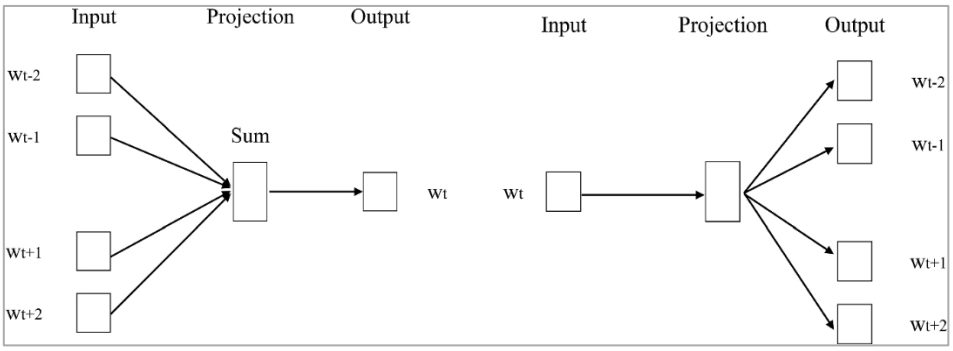


Figure 4: CBOW (left) and skip-gram (right) model (Wang et al., 2019)

The CBOW model tends to predict the target word given context, using the context of words to predict the meaning of the middle word. In contrast, the model called Skip Gram predicts the surrounding words given a target word. Both these models use a two-layer neural network applying SoftMax on top to map raw output scores to probabilities. This process helps allow models to learn word representations in a low-dimensional vector space such that semantics analysis can be efficiently done (Wang et al., 2019).

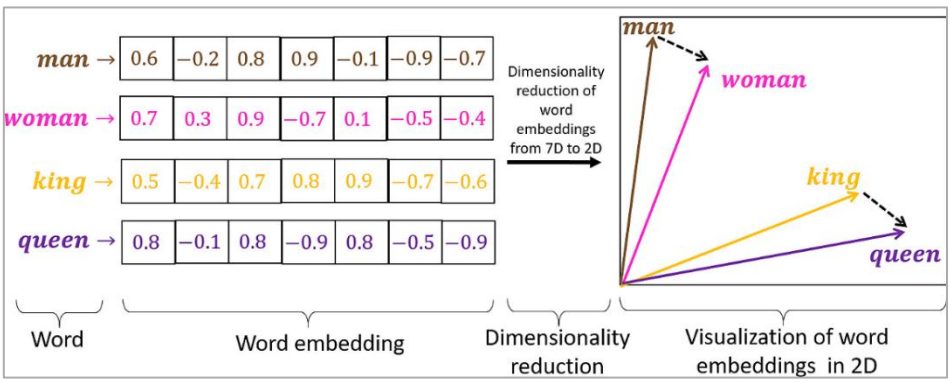


Figure 5: Diagram of Word Embedding (What Are Word & Sentence Embedding? 5 Applications, 2024)

Word2Vec captures the relationships between words by placing similar words close together in this vector space. A classic example given by (Allen & Hospedales, 2019) illustrating this capability is the analogy "King - Man + Woman = Queen." This equation showcases how linear relationships in word embeddings can be leveraged to uncover semantic similarities and relationships among words.

Word2Vec models word relationships, or context, therefore enhancing hate speech detection by identifying derogatory words showing underlying biases. By placing similar words close to each other within one vector space, algorithms can then identify whether speech falls within neutral or harmful contexts. Besides, linearity in the word embeddings is able to find analogy tasks, which really help uncover stereotypes and discriminatory language.

In fact, according to (Saleh et al., 2023), Word2Vec is quite efficient in the extraction of hate speech-related features, particularly when dealing with specific domains of hate speech. Constructing a domain-specific word embedding model known as HSW2V (Hate Speech Word2Vec) is a word embedding model specific to hate speech that is built on the framework of Word2Vec. HSW2V is trained on a corpus containing hate speech, enabling the model to identify and distinguish specific words and phrases in hate speech, improving the ability to detect these harmful languages.

2.2.4.2.2 FastText

FastText (FT) is a word embedding technology developed by Facebook. Different from Word2Vec and GloVe, FastText can effectively handle out-of-vocabulary (OOV) and rare words. The FT algorithm takes advantage of the fact that each word is an n-gram character to provide an effective vector representation for out-of-vocabulary (OOV). not only the vector of the entire word is learned, but also the word is split into n-gram, and then learn the vectors of these subwords. FastText contains multiple words and n-gram features to represent a single document (Pritom Mojumder et al., 2020).

According to (Khasanah, 2021) provide a comparison between GloVe with BiLSTM and FastText with BiLSTM models. The results showed that FastText as a word representation can slightly improve the performance of a single-layer Bi-LSTM model. In addition, FastText with Bi-GRU model has similar results as FastText with BiLSTM on the sentiment classification task, Bi-GRU has fewer hyperparameters to optimize more efficiently. The FastText with CNN model performs comparably to FastText+BiLSTM and FastText+BiGRU on the sentiment classification task, but FastText+CNN is faster to train due to CNN more efficient computation, sparse connections and parameter sharing. (Pritom Mojumder et al., 2020) used convolutional neural network (CNN), bidirectional LSTM (BLSTM), and convolutional bidirectional LSTM

(CBLSTM). Through experimental comparison found that the accuracy of FastText plus BiLSTM on the training set, test set, and validation set reached 91.49%, 87.87%, and 85.5%, respectively.

According to the research paper provided by (Badri et al., 2022), a hybrid approach combining FastText and GloVe word embeddings to detect offensive and hate speech using BiGRU-based deep learning models. Including RoBERTa, BiGRU with FastText, BiGRU with GloVe, BiLSTM, BiGRU FastText and SVM. The results show that the BiGRU Glove-FastText model achieved 84% accuracy, 87% precision, 93% recall, and 90% F1 score in the non-offensive category. In the offensive category, the model achieved 77% precision, 63% recall, and 69% F1 score, which had a better performance than RoBERTa. Furthermore, GloVe combined with FastText has better results than either alone. FastText performed well in terms of recall, making the model more effective in identifying offensive speech, while GloVe improved overall classification accuracy.

2.2.4.2.3 BERT

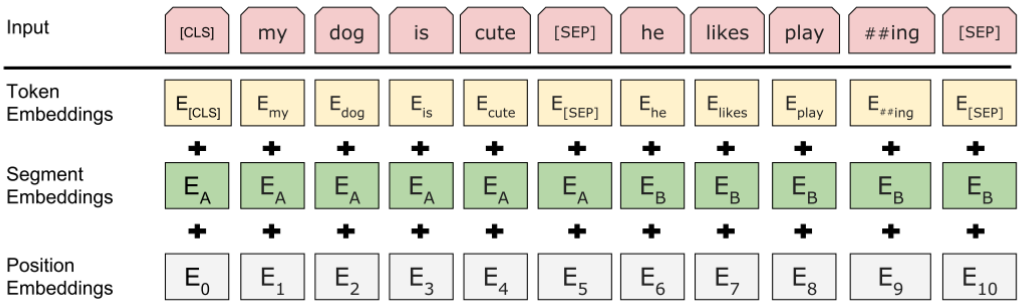


Figure 6: BERT Input Embedding (Devlin et al., 2019)

As understandable from the research paper by (Devlin et al., 2019), it is clear how BERT does feature extraction. First, the job of feature extraction is initiated by the BERT model, where the input sentence is tokenized into tokens. Corresponding to every token, there are three kinds of embeddings, token embedding, position embedding and segment embedding.

Token embeddings represent the semantic meaning of individual tokens, segment embeddings identify the sentence each token belongs to, and position embeddings encode the sequence of tokens within a sentence. These embeddings are summed to create the final embedding vector for each token. A unique token known as [CLS] is always the first token. In BERT, the [SEP]

token acts as a boundary marker between sentences or segments, assisting the model in managing tasks involving multiple textual elements (Politi, 2022). The final embeddings are then passed through multiple Transformer layers.

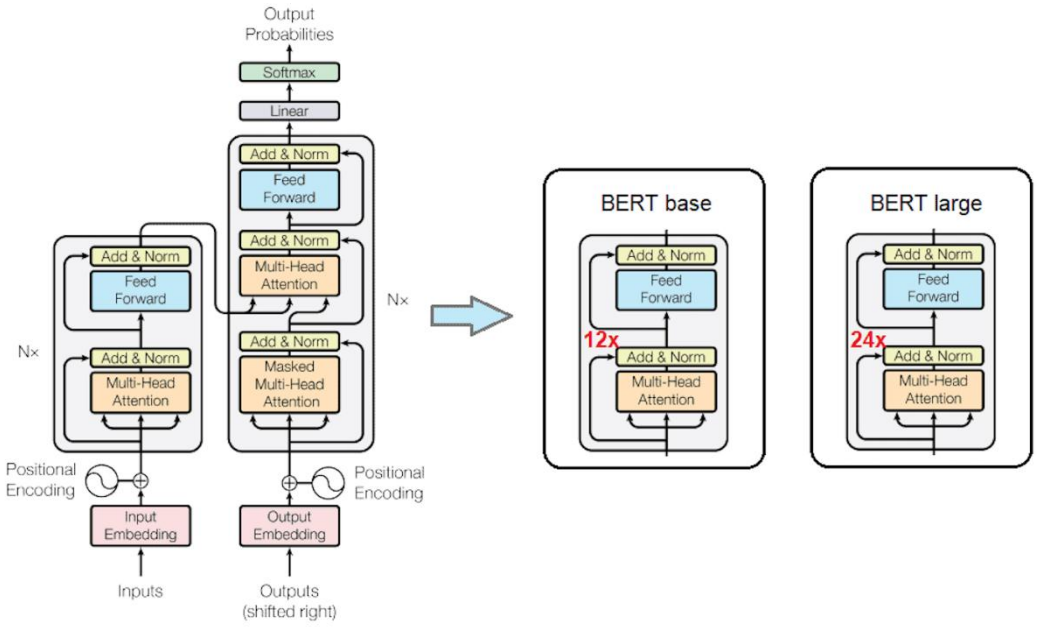


Figure 7: BERT's architecture with multiple Transformer encoders (Kumar, 2024)

The self-attention mechanism within these layers enables BERT to capture contextual relationships between tokens, ultimately producing context-aware representations. For tasks requiring sentence-level understanding, the output corresponding to the [CLS] token is typically used as the aggregated feature representation. This process allows BERT to effectively understand and represent the nuanced semantics of the input text.

2.2.4.2.4 Comparison of Word Embedding Techniques

The performance of the deep model based on a bidirectional LSTM integrated with the HSW2V (Hate Speech Word2Vec) model reached 93% on various datasets, while for the same tasks, the BERT model reached even higher performances with an F1 score of 96% (Saleh et al., 2023). Thus, while HSW2V captures features relating to hate speech quite effectively, BERT perhaps provides a more semantically advanced embedding. Similar to traditional word embeddings, such as Word2Vec or FastText, token embeddings in BERT also represent the meaning of words or sub-words. One point at which these approaches went further than previous approaches was adding more context by position and segment embeddings and incorporating Transformers to learn word representations contextualized by looking both at the left and right context of a

sentence. Such a deeper understanding of context therefore makes BERT excel in tasks related to the detection of hate speech.

2.2.4.3 Summary of Feature Extraction Method

In the hate speech detection process, TF-IDF is advantageous for its fast implementation, but TF-IDF cannot capture text relationships, and becomes inefficient for complex tasks such as text classification. Word2Vec addresses this limitation by incorporating semantic information, but the performance depends on the quality and quantity of training data. For example, human typing errors may cause Word2Vec fail to recognize the certain words. So, FastText techniques including character n-grams, which help it handle unknown words and spelling mistakes better. Building upon these approaches, BERT proves to be more suitable for complex tasks due to its ability to extract deep contextualized features. Nevertheless, its high computational cost remains a significant drawback (Maslej-Krešňáková et al., 2020).

2.2.5 Text Classification

Text classification is one of the basic activities in NLP. Such tasks play an important role in many applications that range from sentiment analysis, topic labelling, spam, and hate speech identification.

From the paper (Occhipinti et al., 2022), it can be realized that most of the older types of machine learning models include Naïve Bayes (NB), Support Vector Machines (SVM), and K-Nearest Neighbor (KNN). These older approaches rely on feature extraction mechanisms to represent the documents numerically through BoW, N-grams, and TF-IDF techniques designed hand in glove. These approaches have certain weaknesses. For example, those methods cannot capture the contextual relationships between words, which is highly desirable for extracting meaningful hate speech sentences from the text.

Deep learning introduced models such as CNNs and RNNs, among others, while more recently Transformer-based models such as BERT have been implemented for text classification. This model allows the automatic learning of meaningful features from raw text and can capture contextual and sequential dependencies that are important in tasks like sentiment detection and hate speech identification. Transformer models are especially viewed as the state-of-the-art

way to cope with long-distance dependencies within text and achieve state-of-the-art results on many text classification tasks.

Classifier	Time (s)	Precision	Recall	F-score
kNN	0.217	82%	81%	83%
MPNN	0.007	92%	91%	92%
Logistic Regression	**0.006**	89%	87%	89%
Random Forest	0.025	**94%**	**94%**	**94%**
XGBoost	0.017	**94%**	93%	**94%**
Multinomial NB	0.009	85%	85%	85%
Gaussian NB	0.011	87%	85%	87%
Bernoulli NB	0.008	91%	91%	91%
RBF SVM	3.661	77%	58%	71%
Linear SVM	0.212	89%	88%	89%
Poly SVM	0.675	25%	50%	67%
Sigmoid SVM	0.806	76%	56%	70%

Figure 8: Different Classifier Comparison (Occhipinti et al., 2022)

According to the comparison done by (Occhipinti et al., 2022), XGBoost and Random Forest yield the best performance in terms of recall and accuracy, along with 94% F-scores. Both have reasonable computational times of 0.017 seconds and 0.025 seconds.

2.2.5.1 XGBoost Classifier

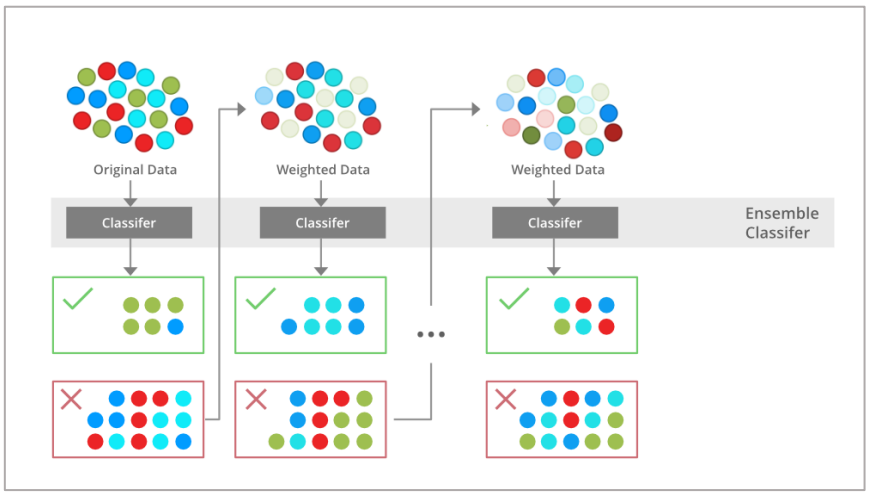


Figure 9: XGBoost Classifier (GeeksforGeeks, 2021)

XGBoost use the gradient-boosting framework that excels in text classification due to its scalability and efficiency in handling large datasets. XGBoost constructs an ensemble of decision trees, each correcting errors of the previously built tree. This recursive process optimizes classification performance, particularly in datasets including complex features, imbalanced classes or noise. A major strength of XGBoost in text classification is its ability to automatically handle missing values and prevent overfitting using regularization techniques such as Lasso (L1) and Ridge (L2). L1 yields sparse models with fewer features by pushing some feature weights to zero, effectively excluding those features. L2 produces a more evenly distributed set of feature weights without necessarily excluding any features from the model (Um, 2021).

Hate speech detection datasets are often highly imbalanced with hate speech instances being far fewer than neutral or non-harmful content. XGBoost ability to iteratively correct errors from previous trees makes it well-suited to handle these imbalances as noted by (Marzieh Babaeianjelodar et al., 2022). It can assign higher importance to misclassified hate speech samples, which improves the overall accuracy rate. Thus, XGBoost optimizes for better recall in identifying hate speech while maintaining precision in detecting non-harmful content.

2.2.5.2 Random Forest Classifier

Random Forest (RF) ability to provide information on the importance of the features used for classification. RF works by constructing multiple decision trees, each tree produces a classification result, and the forest returns the final classification outcome based on the majority vote from all the trees. The splitting strategy commonly used in Random Forest is based on the reduction of Gini impurity. Gini impurity measures the likelihood of misclassifying a randomly chosen element if its label is assigned according to the label distribution within the subset (Occhipinti et al., 2022).

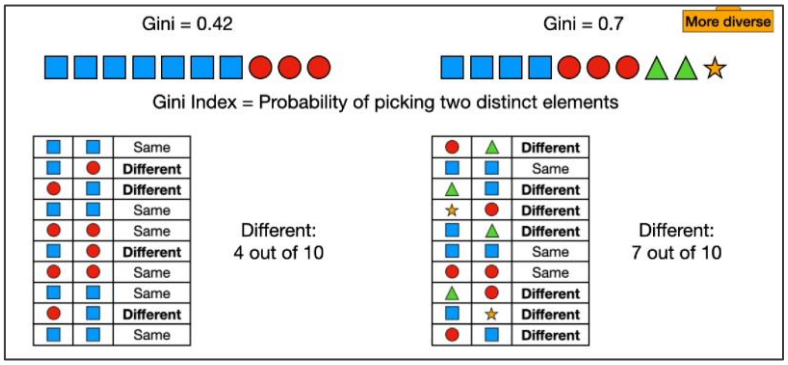


Figure 10: Explain in What is Gini Index

The formula for Gini Impurity is **$Gini=1-\sum(p_i^2)$.**

- 0 indicates a pure node, where all elements belong to a single class.
- 1 indicates maximum impurity, where elements are evenly distributed across multiple classes (Syed Muhammad Mubashir Rizvi, 2024).

2.2.5.3 Comparison between XGBoost and Random Forest in Text Classification

In hate speech detection, both XGBoost and Random Forest have advantages, but each effectiveness varies depending on the characteristics of the dataset. XGBoost is particularly well suited to handling the imbalanced nature of hate speech datasets because it iteratively focuses on misclassified samples, thereby improving recall and ensuring that more hate speech instances are detected. Regularization techniques prevent overfitting and make the model more stable in noisy and unstructured data environments. According to (Bradter et al., 2022) and (Marzieh Babaeianjelodar et al., 2022), the Random Forest model performs well in general classification tasks. However, when dealing with highly imbalanced datasets, the model's performance is inferior to XGBoost and may fail to accurately detect minority "hate" labels. Therefore, the XGBoost model is able to iteratively correct the errors in previous trees, which makes it particularly suitable for dealing with these imbalanced problems in the field of hate speech detection.

2.2.6 Ensemble Learning

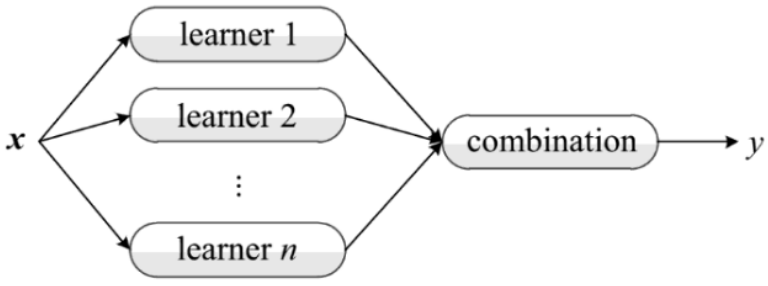


Figure 11: Ensemble Method

Ensemble techniques are effective methods developed to overcome the limitations of individual machine learning models. Since no single model is optimal, ensemble approaches combine multiple models to improve accuracy and performance. (Subramanian et al., 2023) show that integrating multiple instances of a base model such as random forest model using the bagging

method (IBM, 2021) or diverse algorithms such as stacking can reduce variance and enhance learning capacity.

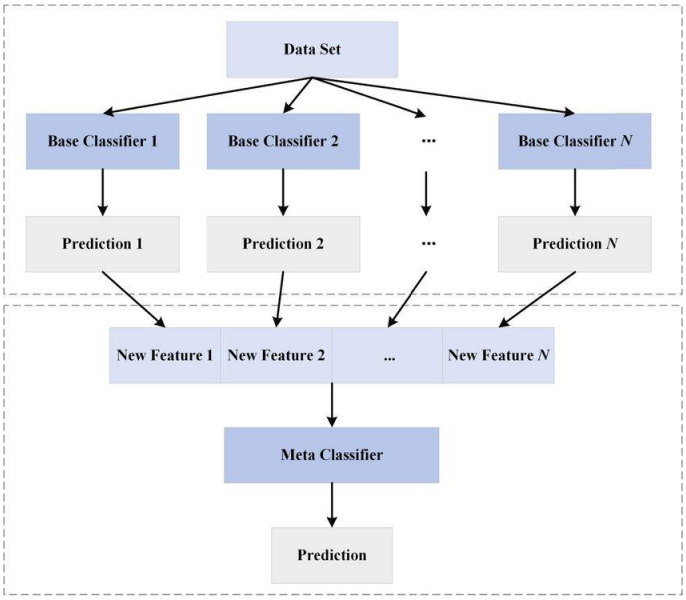


Figure 12: Stacking Framework (Feng et al., 2023)

Ensemble Stacking is a technique that combines several models by creating a meta-model, refining predictions based on the strengths of the base learners. This has seized much attention over many domains because it works in a way that reduces bias while increasing the variance of predictions (Proscura & Zaytsev, 2022). In the proposed model by (Kapil & Asif Ekbal, 2022), the ensemble stack is based on BERT and ALBERT, each independently trained on hate speech, sentiment, and emotion datasets. By putting together such models as a stacked combination, the method captures a wide range of speech patterns that are characteristic of hate speech.

Further, (Kapil & Asif Ekbal, 2022) supported the performance of their stacking-based ensemble model on eleven publicly available hate speech datasets and eight datasets for sentiment and emotion. For the weighted F1 score alongside accuracy, the stack of ensembles performed better when compared to the individual models and traditional baselines on all datasets. The stacking ensemble model in (Kapil & Asif Ekbal, 2022), in which BERT and ALBERT are combined into a multi-task learning framework, reached its best score of 93.63% weighted F1 score, while the lowest F1 score of 60.14% was scored on its best-performing dataset.

Studies have shown by (Mazari et al., 2023) that BERT's contextual understanding is enhanced when stacked with RNN-based models like Bi-LSTM and Bi-GRU. BERT's strength in recognizing toxic content pairs well with Bi-LSTM and Bi-GRU's capacity to handle sequential dependencies, creating a comprehensive view of text structure and sentiment. Stacking in this work provided a maximum F1 score on several hate speech categories and ROC-AUC score of 98.63%, thus improving the detection accuracy and reducing the limitation that may emanate from single-model performance. On the other hand, stacking did not considerably improve the recall and F1 scores for the minority classes in hate speech categories compared to a single-model setup, indicating that stacking improves the performance of majority classes but potentially calls for complementary strategies to balance the rare labels.

In summary, whether to use bagging depends on the stability of the base classifier. If the model is relatively stable, such as the SVM model, bagging will not improve the accuracy. Bagging is more suitable for unstable models such as random forests. To address this limitation (Jurek et al., 2013) proposed the "nice bagging" method to improve the model more stable and reliable. Research provided by (Sáez & Romero-Béjar, 2022) shows that bagging can better handle classification noise in the data, with increased accuracy by about 2-4%. Boosting focuses on difficult classified samples by continuously adjusting sample weights, but this method can easily lead to overfitting of the model (Jurek et al., 2013). To address this problem, MadaBoost and Local Boosting can be used for optimization (Zhang et al., 2023). Although the Stacking method has a strong model fusion capability but has a high risk of overfitting due to its multi-layer structure (Lim, 2022).

2.2.7 Multi-task Learning

According to a research paper (Yuan & Marian-Andrei Rizoiu, 2024), in the binary classification task of neutrality and hate/abuse, the annotators have high consistency in labeling the "neutral" category but low consistency in labeling "hate and abuse". Therefore, it is easier to identify harmless content in the data than to distinguish hateful or abusive content. This also shows that even if annotators generally believe that there are problems with tweets, the existing methods lack a clear distinction between "abuse" and "hate" when dealing with hate speech classification, which limits the performance of the model.

This research (Yuan & Marian-Andrei Rizoiu, 2024) uses multi-task learning (MTL) to improve the model's ability to detect hate speech using multiple datasets. MTL is used to jointly train the common features of multiple datasets, reducing the bias of a single dataset so that it can be better generalized to new fields. Use the Leave-One-Out method to test on the GAB dataset. The MTL macro F1 reaches 0.8738, while the single dataset model has a maximum of only 0.673, a relative improvement of 29.8%. (Kapil & Ekbal, 2024) also mentioned that even using a multi-task BERT model to combine all features, it is still difficult to detect sarcastic or metaphorical content.

Although the multi-task learning (MTL) method has shown strong generalization ability in cross-dataset hate speech detection but still has certain limitations. Experimental results show that the MTL model using eight datasets to train the model and test on the Waseem dataset macro F1 is only 0.3801, which is lower than the single dataset model (using PubFios-L data to train the model and testing it on the Waseem dataset) macro F1 is 0.6048 (Yuan & Marian-Andrei Rizoiu, 2024).

The purpose of MTL is to learn the common features of multiple datasets, but when faced with datasets with special annotation standards or significant data deviations, such as Waseem data, using the MTL model may weaken the model's ability to capture the unique patterns of a single dataset. In contrast, the single-task model trained specifically for the Waseem dataset can more attentively capture its unique language features and expression patterns, thereby achieving better results on this dataset. Therefore, when choosing to use the MTL model, it is necessary to weigh its generalization advantages across datasets and its adaptability to special datasets. If the application scenario involves multi-domain data with unified annotation standards, MTL model is an ideal choice. However, if for a single field with unique annotation specifications or significant deviations or targeted field terms, choosing to use a single-task model may be a more appropriate choice.

(Plaza-Del-Arco et al., 2021) pointed out in the research paper that the MTL model needs to process multiple tasks of hate speech and emotion classification at the same time, so the computational cost is higher than single-task learning (STL). The performance of MTL depends on the data quality of related tasks. Less of high-quality hate speech datasets, especially datasets other than English language. If the low data quality of auxiliary tasks of the MTL

model, such as sentiment analysis or emotion classification, harms the performance of the main task hate speech detection.

2.2.7.1 Feature-Level Fusion - Concatenation

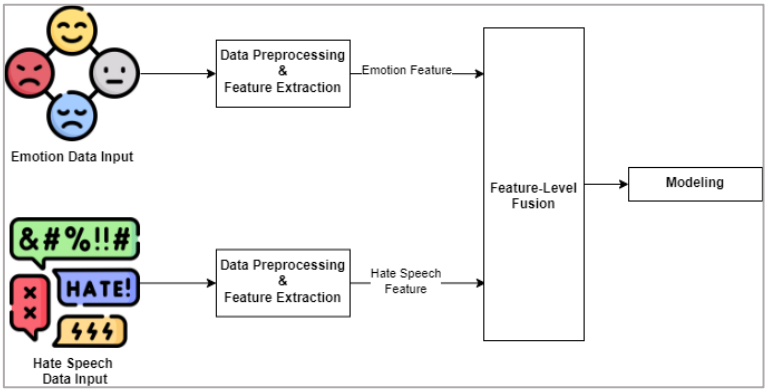


Figure 13: Basic multimodal feature-level fusion framework

Feature-level fusion belongs to early fusion (GeeksforGeeks, 2024), the process of combining many features from multiple types of data to better understand the content and thus improve model performance. Fusion can be achieved in a variety of ways, including feature stacking, weighted averaging, concatenation, or advanced methods such as support vector machines (SVMs) and neural networks (Chetana Kamlaskar & Abhyankar, 2020). Sum fusion and cascading are common fusion techniques that combine features from multiple models to create a unified representation.

The paper (Dwivedy & Roy, 2023) proposed a deep feature fusion (DFF) model to improve the accuracy of hate speech detection by concatenating text and image features. The text features (512 dimensions), the text features in the image (512 dimensions), and the image visual features (512 dimensions) are concatenated into a 1536-dimensional joint feature vector, which is input into the fully connected layer for classification. The concatenated multimodal model (69.04% accuracy) is better than the model using only text (68.80%) or only image (54.90%).

According to the research paper (Khan et al., 2023), five different fusion methods are used to fuse pre-trained language models to optimize hate speech detection. The five combination strategies include addition, concatenation, interleaving, multiplication, and random interleaving. The results show that concatenation and interleaving have similar performance. In low computing resource scenarios, the paper recommends using Addition instead of Concatenation because Addition does not increase the dimension and the both methods

performances are similar. The disadvantage of Concatenation is the high computational cost and limited performance advantage.

2.2.7.2 Joint Representation Technique

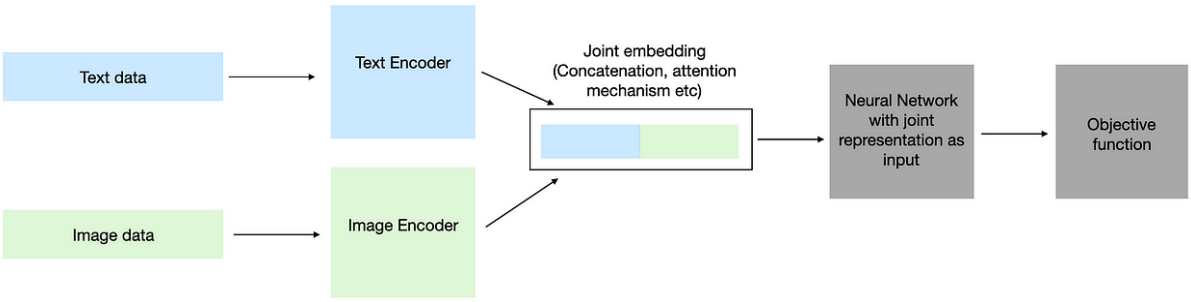


Figure 14: Joint Represent Learning Model (Parate, 2024

Joint Representation Learning is a method designed to integrate multiple data modalities into a single unified representation. By processing different types of inputs, like text and images, it generates embedding using specialized encoder models for each data source. For instance, text is transformed into embeddings through text encoders such as transformers, neural networks, or probabilistic models, while images are processed with image encoders like CNNs, ResNet, or Vision Transformers. These encoders create embeddings that represent each type of input, which are then combined to form a single, cohesive joint multimodal representation. This representation can either pass through multiple hidden layers or can be used directly for predictions. The model is trained end-to-end, enabling it to learn both the data representations and the specific task it needs to perform (Parate, 2024).

In hate speech detection, combining emotion-aware features with text data helps the model better understand the context. (Rana & Jha, 2022) proposed a method that integrates sentiment and semantic features using a multimodal framework. Their model combines text embeddings from a Transformer model with sentiment information extracted from audio, bringing these inputs together in a shared space. This approach allows the model to detect subtle links between language and emotional tone, which is especially useful for identifying hate speech in multimedia content. The study showed that adding emotion-aware features improved precision and recall compared to text-only models, proving the value of including sentiment context in hate speech detection.

2.2.7.3 Comparison Between Using Joint Representation Learning and Concatenation

Due to there being less information comparing Joint Representation Learning and Concatenation, cite this study by (Schlötterer et al., 2017) compared joint representation learning (TG-SPLIT) with feature concatenation for classifying academic papers. The results demonstrated that while joint learning aims to integrate network structure and text content during training, it did not significantly outperform the simpler concatenation approach, using Joint Representation Learning accuracy is 83.1%, while using the Concatenation method has 84.8%. However, concatenating features (text and network embeddings) still achieved better performance than using either modality alone.

2.2.8 Deep Learning

According to Kapil and Asif Ekbal (2022), traditional machine learning methods, including logistic regression and support vector machines, provided the initial foundation for classifying text data. These models are not good at handling the nuanced, complex language. Advances in deep learning have shifted the focus to neural network models, particularly Transformer-based architectures such as BERT and ALBERT. These models are good at capturing contextual and semantic meanings suitable for distinguishing hate speech.

Deep Neural Networks (DNNs) serve as virtual brains. More layers allow the model to learn more complex patterns. DNNs are suited for data with intricate structures, like text and images. Among DNN architectures, Convolutional Neural Networks (CNNs) are well-suited for image recognition. These networks function like a computer's eyes, scanning segments of an image to detect shapes, colours, and other details, which are then synthesized to understand the whole picture. Recurrent Neural Networks (RNNs) excel with sequential data such as text and speech, by retaining previous information to interpret current inputs, act like a computer memory. This structure allows RNNs to capture dependencies in sentences and time-based data more effectively (SPRH LABS, 2019). The paper (Chiril et al., 2021) mentioned that hate speech detection currently focuses on using neural models such as LSTM, bidirectional long short-term memory (Bi-LSTM), GRU and CNN, as well as word embedding models such as FastText, word2vec and ELMo.

2.2.8.1 Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs), including Bidirectional Recurrent Neural Networks (BiRNNs), Gated Recurrent Units (GRUs), and Long Short-Term Memory (LSTMs). RNNs are different from FNNs in that RNNs process sequential data by integrating hidden memory states, while Feedforward Neural Networks (FNNs) lack memory and process inputs in a unidirectional manner. But RNNs also have some disadvantages. First, RNNs require multiple iterations to adjust model parameters and reduce error rates. Second, RNNs are susceptible to the vanishing gradient problem (Sharkawy, 2020), which hinders model learning when the gradient approaches zero, resulting in underfitting. This means that the model is too simple and has not learned complex features. Third, RNNs require high computing power, memory and processing time (Pritom Mojumder et al., 2020) (AWS, n.d.).

According to (Sharkawy, 2020), due to the limited memory capacity of RNN, it is difficult for RNN to process very long sequences, and some early information will be forgotten after a certain length. When using tanh or ReLU activation functions, RNN is prone to gradient explosion and gradient vanishing, which hinders model convergence. In addition, RNN requires a large amount of high-quality data to prevent the model overfitting or underfitting. To address these limitations, the new model architectures LSTM and GRU alleviate the gradient problem through gating mechanisms. Those models can be enhanced with attention mechanism techniques for better context understanding to improve model interpretability.

Based on the explanation by (Van Houdt et al., 2020) and (Sakib Ashraf Zargar, 2021), LSTM networks represent an improved version of standard recurrent neural networks that effectively address the common problems of vanishing and exploding gradients. LSTM has a more advanced memory unit and two key features. LSTM has a specialized memory unit that can store and manage important information for a long time. Second, the LSTM architecture includes three basic control mechanisms. First, the forget gate is responsible for deciding which information needs to be deleted. Second, the input gate determines which new information to store. Third, the output gate controls which information to pass forward. By managing which information to keep, update, and discard, LSTM can effectively learn and remember patterns in long sequences.

2.2.8.2 Transformer Keyword-Based Model

According to (De la Peña Sarracén & Rosso, 2023) study focused on the performance of the model in key scenarios such as subtle hate and non-hate texts containing hate keywords, such as “oh shit” being misclassified as hate. The paper used the BERT and ROBERTA models and three datasets. The results show that less than 50% of the words the model focuses on are hate words, and it is difficult for the model to detect hate text that does not contain hate words, but it is easy to misjudge non-hate text that contains hate words.

Class	HateEval		W&H		Founta	
	N-HS	HS	N-HS	HS	N-HS	HS
HMRF	**48.41**	**70.28**	**48.30**	**76.67**	**25.36**	**56.95**
YAKE	36.75	45.74	56.80	60.92	24.56	18.74

Figure 15: Coverage of HMRF and YAKE keyword extraction methods on different datasets (De la Peña Sarracén & Rosso, 2023)

To address these challenges, (De la Peña Sarracén & Rosso, 2023) employed HMRF and YAKE for keyword extraction. HMRF penalizes words that appear frequently in non-hate texts, even if these words appear frequently in hate texts, and highlights unique words in hate contexts. YAKE is a method for extracting keywords suitable for multiple text types. In the experiment, the keywords extracted by HMRF overlapped more with the words the model focused on, and the words extracted by YAKE overlapped less with the words the model focused on. Among the tested models, BERT achieved the highest F1-score of 80.04% on the HateEval dataset, while RoBERTa obtained the highest F1-score of 72.36% on the W&H dataset.

Although HMRF keyword extraction is better than YAKE, HMRF relies on interpretation tools Captum and HMRF may ignore low-frequency hate words. According to (De la Peña Sarracén & Rosso, 2023), (Yuan & Marian-Andrei Rizoiu, 2024), (Chiril et al., 2021) and (Kapil & Ekbal, 2024) keyword-based detection is only effective for explicit hate speech, such as insults or discriminatory language, but is ineffective for subtle hate speech, such as sarcasm and metaphorical hate speech.

2.3 Similar Systems/ Works

Table 1: Similar works of hate speech detection

No	Research Topic	Author(s)	Description	Dataset Used	Analysis Techniques	Evaluation Technique	Evaluation Result	Findings/Challenges	Limitations/Gap
1	Challenges of Hate Speech Detection in Social Media	György Kovács, Pedro Alonso, Rajkumar Saini (13 February 2021)	This research focuses on the challenges of detecting hate speech in social media due to factors like data scarcity, varying definitions of hate speech, and context dependency. The paper also highlights the difficulties in creating a reliable labeled dataset due to the subjective nature of hate speech.	HASOC 2019 dataset	CNN-RNN, **Pretrained models:** BERT, RoBERTa	Macro, Weighted F1-scores, Cross-validation	CNN-LSTM (macro F1-score): 0.63% RoBERTa (macro F1): 0.7882 RoBERTa + FastText (Macro F1): 0.7953	**Findings:** 1. Transformer-based models (RoBERTa) outperform traditional deep learning (CNN-LSTM) and classical ML methods. 2. Adding external labelled data (OLID, HateBase) improves performance, but pre-trained embeddings (fastText, WikiNews) alone are less impactful. **Challenges** included potential overfitting, data imbalance, and lack of consensus on hate speech definition.	**Limitation:** 1. RoBERTa performed best, but computationally expensive and may not be practical for real-time applications. **Gap:** 1. The research primarily only focuses on a single dataset.
2	Profiling Hate Speech Spreaders on Twitter: SVM vs. Bi-LSTM	Inna Vogel, Meghana (Sep,2021)	Investigates hate speech spreaders on Twitter, comparing the performance of SVM and Bi-LSTM for hate speech classification. The SVM model text preprocessing removes URL/username placeholders, standardizes emoticons, and performs word form restoration. The Bi-LSTM model retains more original text information, such as stop words and short words.	Twitter hate speech dataset. The data is available in English (EN) and Spanish (ES).	SVM, Bi-LSTM (Sentence-BERT)	Accuracy, Precision, Recall, F1-score	SVM accuracy 64% (en), 75% (ES), average 69.5%. Bi-LS TM accuracy 59% (en), 79% (ES), average 69%. The SVM model performs slightly better on average and is more stable	**Finding:** 1. Model have a better performance on Spanish tasks. **Challenge:** 1. Spelling errors, abbreviations, hashtags, and mentions in tweets can make it more difficult for models to analyze.	**Limitations:** 1. Dataset have bias, Bi-LSTM requires high computational resources. 2. Simply relying on insults or profanity cannot effectively distinguish hate speech. **Gaps:** 1. Interpretable classification. 2. Analyzing the impact of hate words and context.

No	Research Topic	Author(s)	Description	Dataset Used	Analysis Techniques	Evaluation Technique	Evaluation Result	Findings/Challenges	Limitations/Gap
3	Emotionally Informed Hate Speech Detection: A Multi-target Perspective	Patricia Chiril, Endang Wahyu Pamungkas, Farah Benamara, Véronique Moriceau, Viviana Patti (28 June 2021)	Multi-target hate speech detection framework, using emotional features to improve hate speech detection results.	**Emotional resources**: SenticNet, EmoSenticNet, HurtLex **Hate Speech Dataset**: 1. Davidson 2. Founta 3. Waseem 4. Evalita 5. IberEval 6. HatEval	SVM, LSTM, CNN, BERT, ELMo	Precision, Recall, Accuracy, F1 score	1. CNN-FastText + Hurtlex, F1-score 0.845 2. BERTmulti-task + Hurtlex, F1-score 0.829 3. LSTMmulti-task, F1-score 0.818 4. LSTMmulti-task(FastText), F1-score 0.825 5. LSTMmulti-task + sentic, F1-score 0.823 6. LSTMmulti-task (FastText) + sentic, F1-score 0.836 7. ELMomulti-task + sentic, F1-score 0.769 8. BERTmulti-task + sentic, F1-score 0.786	Findings: 1. BERTmulti-task overall performs best in most tasks, with F1 score increased by 5-17%. 2. Combining the features of EmoSenticNet and HurtLex helps the model detect hate speech.	**Limitations:** 1. Existing datasets have sampling bias, and over-reliance on keywords affects model generalization. 2. Training on general topic datasets often fails to take into account topic-specific linguistic characteristics. **Gaps:** 1. Covert hate detection 2. Multimodal Fusion 3. Bias Mitigation
4	Generalizing Hate Speech Detection Using Multi-Task Learning: A Case Study of Political Public Figure	Lanqin Yuan, Marian-Andrei Rizoiu (17 July 2024)	A multi-task learning framework based on BERT, training multiple hate speech datasets simultaneously (8 public datasets + 1 newly constructed dataset). By sharing BERT model parameters, learn more general representations of hate speech and reduce the bias of a single dataset.	**Hate Speech Datasets:** 1. Davidson 2. Waseem 3. Reddit 4. Gab 5. Fox 6. Mandl 7. Stormfront 8. hateEval	MTL-NCH, MTL-MV, Eight BERT models based on different hate speech dataset	Use the leave-one-out method to evaluate. Macro-F1, Weighted-F1, Micro F1	The highest: MTL-NCH on Gab dataset, macro F1-score is 0.8738.	**Finding:** 1. After MTL training, BERT can better distinguish hate, insults, and neutral content, especially Islamophobia and sexism. **Challenges:** 1. There is no unified definition of hate speech, and the annotation standards of different datasets vary greatly. 2. Difficulty in generalizing models across domains.	**Limiation:** 1. Models are not very effective at detecting implicit (veiled, indirect) hate speech. **Gap:** 1. The current model only processes text, and in the future it can be combined with context such as conversation chain or media.

No	Research Topic	Author(s)	Description	Dataset Used	Analysis Techniques	Evaluation Technique	Evaluation Result	Findings/Challenges	Limitations/Gap
5	Interpretable and High-Performance Hate and Oensive Speech Detection	Marzieh Babaeianjelodar, Gurram Poorna Prudhvi, Stephen Lorenz, Keyu Chen, Sumona Mondal, Soumyabrata Dey, Navin Kumar (26 Jun 2022)	This paper mainly explains how to make hate and offensive speech detection models more explainable and understandable.	Twitter's public annotated dataset	**XGBoost,** feature extraction: 1. VADER 2. POS 3. NER 4. TF-IDF **LSTM, AutoGluon, ULMFiT**	F1-score	**Imbalanced Dataset Result (class 1: Hate F1-score):** 1. XGBoost: 0.75 2. LSTM: 0.38 3. AutoGluon: 0.37 4. ULMFiT: 0.38 **After Down Sampling (class 1: Hate F1-score):** 1. XGBoost: 0.79 2. LSTM: 0.69 3. AutoGluon: 0.77 4. ULMFiT: 0.66	**Findings:** 1. SHAP visualization clearly shows the contribution of key features to classification. 2. XGBoost+SHAP builds a high-performance and explainable hate speech detection model that outperforms deep learning models and identifies key classification features. **Challenges:** 1. Dataset Imbalance.	**Limitations:** 1. Model insufficient contextual understanding makes it difficult to detect implicit hate. 2. Does not take into account demographic characteristics, such as user gender, region and so on. **Gaps:** 1. The definitions of hate and offensive speech overlap, and existing models do not clearly distinguish between the two. 2. Propose an interpretable XGBoost model that combines multimodal features such as emotion, syntax and symbols.
6	Systematic keyword and bias analyses in hate speech detection	Gretel Liz De la Peña Sarracén , Paolo Rosso (17 June 2023)	The paper focuses on hate speech detection using transformer-based models, analyzing biases and the relationship between model attention and hateful keywords. Using keyword extraction method (HMRF and YAKE).	**Twitter hate speech datasets:** 1. HatEval 2. Waseem & Hovy 3.Founta	**Keyword Extraction Method:** 1. HMRF 2. YAKE **Model:** 1. BERT 2.ROBERTA	F1-score	**BERT Model** (the highest F1-score) On HateEval Datset: 80.04% **ROBERTa Model** (the highest F1-score) on W&H dataset: 72.36% **Keyword Coverage Persentage (HMRF):** **1.** On HateEval Dataset: 70.28% (hate speech), 48.41% (non-hate speech) **2.** On W&H Dataset: 76.67% (hate speech), 48.30% (non-hate speech) **Keyword Coverage Persentage (YAKE):** **1.** On HateEval Dataset: 45.74% (hate speech), 36.75% (non-hate speech) **2.** On W&H Dataset: 60.92% (hate speech), 76.67% (non-hate speech)	**Findings:** 1. HMRF outperformed YAKE in extracting hateful keywords. HMRF keywords had higher overlap with model-attended words than YAKE. 2. Keyword-based detection is only effective for explicit hate speech. 3. Experimental results show that when hate text that does not contain hate keywords is used to fine-tune the model, the model's bias towards hate keywords will be reduced, and the reduction in the model's bias will lead to an improvement in the F1 score.	**Limitations:** 1. HMRF might exclude hateful words frequent in non-hateful contexts. 2. Models difficult to predict subtle hate, without explicit keywords. **Gaps:** 1. Exploring cross-lingual or cross-platform hate speech detection. 2. Combining HMRF with external hate lexicons such as HurtLex.

No	Research Topic	Author(s)	Description	Dataset Used	Analysis Techniques	Evaluation Technique	Evaluation Result	Findings/Challenges	Limitations/Gap
7	A Multi-Task Learning Approach to Hate Speech Detection Leveraging Sentiment Analysis	Flor Miriam Plaza-Del-Arco, M. Dolores Molina-González, L. Alfonso Ureña-López, María Teresa Martín-Valdivia (August 17, 2021)	This paper mainly studies how to use sentiment analysis and multi-task learning methods to detect Spanish hate speech on Twitter.	**Hate Speech Datasets:** 1. HateEval 2. MEX-A3T **Emotion Datasets:** 1. InterTASS 2. EmoEvent	Transformer-based model BETO	Precision, Recall, F1-score	**HatEval**: Macro-F1=78.47% (MTLsent+emo) **MEX-A3T:** Macro-F1=86.58% (MTLsent+emo)	**Findings:** 1. Verify that sentiment analysis is effective for detecting hate speech in Spanish in multi-task learning. 2. The quality of corpora or dataset is important in a multitask learning environment. **Challenge:** 1. Tweets that contain offensive words but are not actually hateful, implicit hate speech without clear target words or with vague sentiment labels.	**Limitation:** 1. Multitasking leverages other corpora for classification computational cost is higher. 2. Lack of resources, especially in languages other than English. **Gaps:** 1. Expanding to other languages and combining with tasks such as sarcasm detection.
8	Transformer-Based Ensemble Learning for Hate Speech Detection Leveraging Sentiment and Emotion Knowledge Sharing	Prashant Kapil and Asif Ekbal	The paper proposes a transformer-based ensemble learning approach for hate speech detection, leveraging sentiment and emotion knowledge sharing in a multi-task learning (MTL) framework.	The 11 datasets are mainly from 1. Twitter 2. Facebook 3. Stormfront 4. YouTube	BERT, ALBERT, BERT-MTL, ALBERT-MTL, BERT-MTL + Sentiment/Emotion, ALBERT-MTL + Sentiment/Emotion, Stacking Ensemble (BERT-MTL and ALBERT-MTL)	Accuracy, Weighted F1-score, Misclassification rates	**Stacked Ensemble Model:** highest weight f1-score: 93.63% **BERT-MTL + Sentiment/Emotion Model:** highest Weighted-F1: 93.13%	**Findings:** 1. MTL with sentiment and emotion data can improve hate speech detection. 2. Ensemble learning further improves performance by combining BERT and ALBERT. **Challenge:** 1. Hate speech definitions lack of standardization. 2. In hate speech detection, high-quality annotated data is scarce, thus inherent biases occur in training.	**Limitations:** 1. Most datasets are in English, limiting applicability to other languages. 2. Models struggle with ambiguous or sarcastic hate speech.

The above table shows the contributions, challenges, limitations and gaps of different research papers in detecting hate speech. Some use ensemble models, multi-task models combined with emotion features, compare ML models and deep learning models and use keyword extraction. However, most research papers face the same problem of being unable to recognize subtle hate speech.

2.4 Technical Research

2.4.1 Programming Language Chosen

This project has selected Python as its primary programming language because it has a thoroughly designed data science and machine learning ecosystem. Python is a high-level, interpreted language that is well-suited for managing unstructured datasets, making it an essential player in handling social media text content for hate speech classification. Python integrate with many NLP libraries, including NLTK, spaCy, and Hugging Face's Transformers and provides the necessary tools to build and deploy intricate language models to accurately fuse the nuances.

Python's ability to work with machine learning libraries such as TensorFlow, PyTorch, and Scikit-Learn gives this project's emotion-aware hate speech detection model the flexibility it needs to create, train, and optimize models that take into account the complex expressions included in hate speech. The ability to seamlessly integrate pre-trained models, such as BERT, enhances model accuracy by leveraging deep semantic understanding, allowing for more precise detection of both overt and subtle forms of hate speech. Additionally, Python's robust data visualization tools, including Matplotlib, Seaborn, and Plotly, facilitate the clear presentation and analysis of findings, making it easier to get the insights and model performance (Butwall et al., 2019).

2.4.2 IDE Chosen

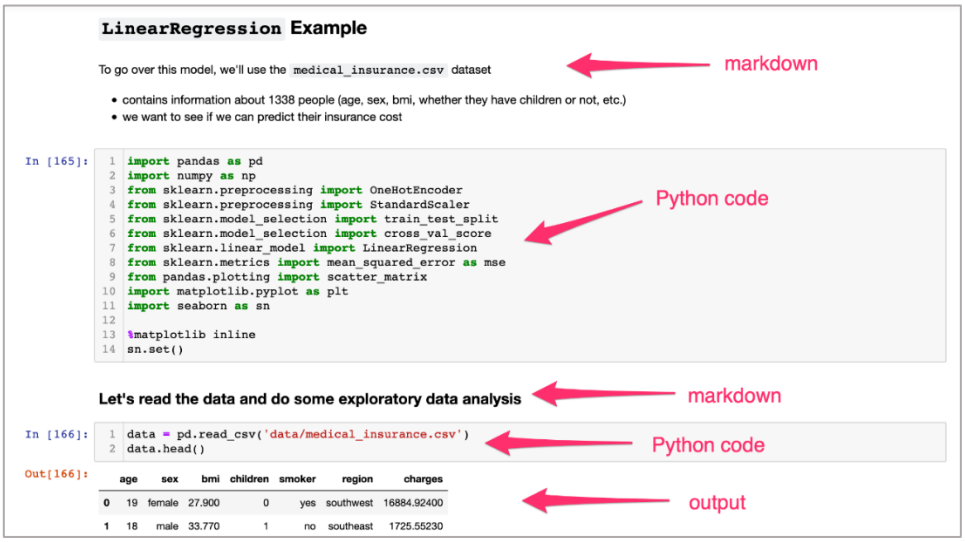


Figure 16: Interface of Jupyter Notebook (Ciprian Stratulat, 2023)

I picked Jupyter Notebook for this project as the IDE (Integrated Development Environment) because it has a flexible design that lets code run in interactive kernels. This means getting quick feedback and easy debugging. This is very important for natural language processing (NLP) tasks like hate speech detection, which need quick changes to improve models and look at complex data. More than 40 languages, including Python, R, and Scala, are supported by Jupyter Notebook, which makes it highly available when integrating multiple NLP libraries (What Is Jupyter Notebook | Domino Data Lab, n.d.).

An important benefit is Jupyter Notebook's real-time data exploration, which is especially useful for NLP workflows. As hate speech detection often involves processing and testing language models on large text datasets, Jupyter's structure supports incremental code execution and live output, making it easy to visualize and refine results throughout model training and evaluation. This interactive setup allows for quick adjustments and ensures an efficient and focused workflow (Erin Moore et al., 2023).

2.4.3 Libraries Chosen

Below are several Python libraries noticed by (Moez Ali, 2024) that are well-suited for data analysis, deep learning model development, and data visualization.

Natural Language Toolkit (NLTK) & spaCy

Provides basic functions for text preprocessing, such as word segmentation, lemmatization, and stop word removal. Its work in hate speech detection can help organize unformatted social media text into a form suitable for analysis, ensuring that the model only focuses on the relevant parts of the language.

TensorFlow and PyTorch

Models on these deep learning frameworks can be developed, trained and fine-tuned. The transfer learning and custom layers made possible by the flexibility of TensorFlow and PyTorch in dealing with complicated architectures continues to improve the model quality.

Scikit-Learn

This library is essential for implementing machine learning algorithms and statistical analysis tools. In case of hate speech detection, Scikit-Lean allows the users to perform data splitting

and model evaluation purposes and also aid in employing the primitive classifiers to benchmark and contrast performance with the deep learning models.

Pandas and NumPy

These libraries provide efficient dataset management functions, basic tools for numerical operations and data processing. In hate speech detection, these libraries can be used to clean, organize and manipulate data to prepare for model input.

Matplotlib and Seaborn

These visualization libraries support the creation of charts and plots that help analyse data distributions, visualize model performance metrics, and identify patterns and trends in hate speech data.

2.4.4 Operating System Chosen

The operating system (OS) is naturally a critical part of any computer as it supervises and coordinates the use of the hardware resources and provides a controlled environment for the software programs to be tested and developed. Picking up the right OS is essential to maintain compatibility with the project's tools, libraries, and development processes. For this project, Windows 11 (64-bit version) will be used since there are many people who use it and its compatibility with related applications like Microsoft Visual Studio Code is a perfect fit.

Windows 10 and Windows 11, Microsoft Windows operating systems have VS Code which forms an integral part of the software to be developed in this project (Microsoft, n.d.). In addition, they note that devices with such power are sufficient to run the application with a processor of 1GHZ and a RAM of 1GB at a minimum (Microsoft, 2021). However, this project employs a laptop with Windows 11, which has a 7730U AMD Ryzen 7 CPU and 16 GB ram that is more than the minimum level required, thus ensuring smooth running while performing data-heavy tasks in deep learning and natural language processing.

The project setup also consists of Visual Studio Code version 1.95.0 with Jupyter Notebook as well as deployment of Python 3.12.6, providing a solid environment for developments such as interface design, data editing, and editing algorithms.

2.4.5 Web browser chosen

Google Chrome has been chosen as the primary web browser for the model deployment phase of this project. Known for its speed, stability, and extensive developer tools, Chrome provides a reliable environment for accessing and testing the deployed model's web interface. Chrome's Developer Tools are particularly useful for debugging, inspecting elements, and monitoring network requests.

2.5 Summary

Existing research on hate speech detection mainly focuses on keyword matching or basic text classification methods, which often have limited effectiveness when faced with cryptic or indirectly expressed hate speech. In addition, while some studies have explored the application of sentiment analysis in text analysis, research on combining sentiment features with hate speech detection is still relatively scarce, and the quality of data is very important for multi-task models. Incorporating sentiment features helps capture complex contextual associations. For example, certain negative emotions, such as anger or fear, may enhance the ability to identify hate speech. This project aims to improve the accuracy and contextual understanding of hate speech detection by combining emotion features with hate speech detection.

CHAPTER 3: METHODOLOGY

3.1 Introduction

Chapter 3 outlines the methodology employed to achieve the project's objectives and explains the systematic approach used to ensure the project's success, focusing on the Cross-Industry Standard Process for Data Mining (CRISP-DM) framework. This methodology was chosen for its structured and iterative nature, allowing for the continuous refinement of processes. The chapter details the phases involved, including business and data understanding, data preparation, modelling, evaluation, and deployment. Additionally, data sources, preprocessing techniques, and data understanding are discussed. By applying this methodology, the project aims to integrate emotion analysis with hate speech detection to enhance the accuracy and contextual understanding of the model.

3.2 Methodology

3.2.1 Introduction of Methodology

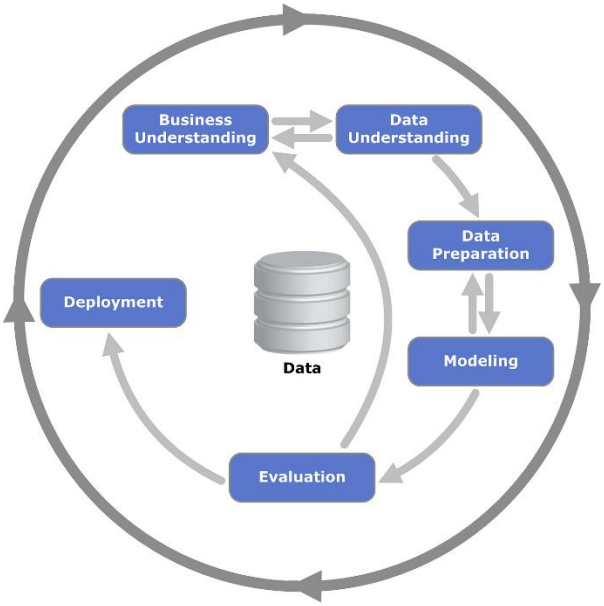


Figure 17: CRISP-DM Methodology (Chumbar, 2023)

CRISP-DM is a cornerstone for data mining and analytics projects worldwide. Developed in the late 1990s, CRISP-DM gives a clear and organized process that helps maintain consistency and accuracy from start to end. CRISP-DM provides a concrete step-by-step guide to transforming raw data into valuable insights (Richard O'Hara et al., 2023).

CRISP-DM has six key phases: business understanding, data understanding, data preparation, modeling, evaluation, and deployment. Each phase outlines specific goals and tasks, forming a clear path from initial project conception to final implementation. This iterative framework allows for early phases to be revisited and refined as needed, ensuring adaptability and accuracy throughout the project (Richard O'Hara et al., 2023).

The widespread adoption of CRISP-DM can be attributed to its versatility and extensibility. The most popular framework is CRISP-DM, which methodically handles business issues in a sequence that progresses from business issues to business solutions (Hotz, 2024).

3.2.2 Methodology of Choice and Justifications

CRISP-DM is preferred for this project because it includes deployment as its final phase and the need to deliver and implement the results. Unlike SEMMA, which concludes with an assessment and lacks an inherent deployment step, CRISP-DM ensures the project is production-ready. Although KDD is similar to CRISP-DM in its iterative nature and ability to revisit earlier stages, CRISP-DM's structured steps are more deployment-oriented, making it the optimal methodology.

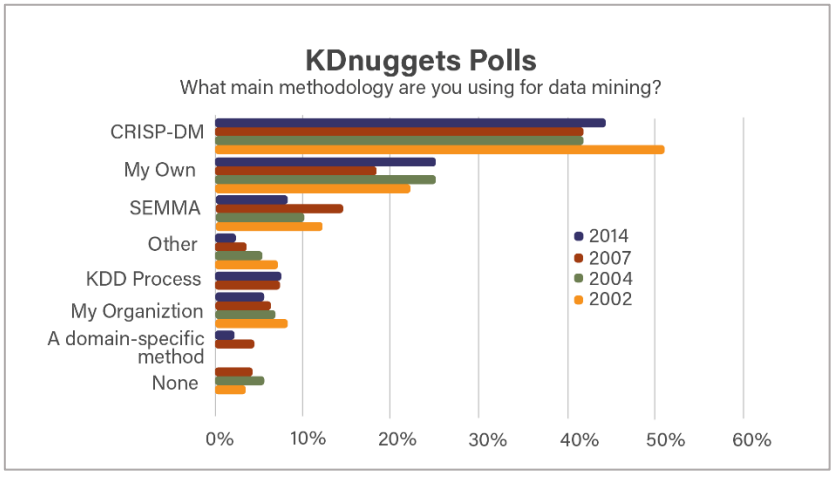


Figure 18: CRISP-DM methodology over 12 years (Hotz, 2024)

Compare the differences between the various methodologies provided by (YugmPatel, 2024).

Table 2: Comparison of Three Methodologies

Methodology	CRISP-DM	KDD	SEMMA
Phase	1. Business Understanding	--	--
	2. Data Understanding	1. Data Selection	1. Sample
		2. Data Preprocessing	2. Explore
	3. Data Preparation	3. Data Transformation	3. Modify
	4. Modelling	4. Data Mining	4. Model
	5. Evaluation	5. Interpretation/ Evaluation (YugmPatel, 2024)	5. Assess (YugmPatel, 2024)
	6. Deployment (Hotz, 2024)	--	--
Total Phase	6 Phases	5 Phases	5 Phases
Flexibility	CRISP-DM is flexible for an iterative approach, enabling phases to be revisited to progressively improve the outcomes (Andre Luiz, 2023).	KDD is flexible and follows an iterative approach, allowing the integration and transformation of new data to achieve varied and more suitable results. The insights gained can be fed back into the process, improving its overall effectiveness (Andre Luiz, 2023).	SEMMA itself is a loop, and the internal steps can be iterated as needed (Data Mining and the Case for Sampling Solving Business Problems Using SAS ® Enterprise Miner TM Software, n.d.).
Strength	CRISP-DM offers strong adaptability and ease of implementation, making it suitable for	KDD provides valuable advantages, such as uncovering hidden data patterns, facilitating evidence-based decision-	SEMMA provides a clear and structured process from data sampling to model evaluation, which is

	various data science projects. Its structured approach, starting with business understanding and concluding with deployment, ensures alignment with business needs and supports iterative refinement for deeper insights (Hotz, 2024).	making, and enabling predictive analytics. Its applications extend across various industries, including healthcare and retail, aiding in tasks like forecasting disease outbreaks or anticipating customer purchasing behavior (Knowledge Discovery in Databases, 2024).	suitable for handling more technical data mining projects. SEMMA's process helps data analysts focus on exploring, modifying and modeling data at each stage to find the best prediction results (Palacios et al., 2017).
Weakness	Although CRISP-DM allows iteration, in practice, its structured and phased nature can sometimes limit the efficiency of rapid iteration. Documentation is heavy, with almost every task having a documentation step. CRISP-DM is not a strict project management method because it assumes that users are individuals or small teams, ignoring the teamwork and coordination required	KDD encounters challenges like managing high-dimensional data, addressing noisy or incomplete data, and handling privacy and security concerns. Additionally, the value of the knowledge it uncovers largely depends on the quality of the input data (Knowledge Discovery in Databases, 2024).	SEMMA lacks a clear business understanding and deployment phase, so it may be insufficient in projects involving business needs and the final implementation of the project. The process focuses on technical modeling and data processing. It lacks guidance on the overall business context of the project, which limits its applicability in a wider range of business data science

	for large projects (Hotz, 2024).		projects (Palacios et al., 2017).

For this project, CRISP-DM was chosen as the primary methodology due to its comprehensive, structured and iterative framework that aligns with the objectives of this project. CRISP-DM's complete lifecycle approach ensures that this project is well-structured from start to finish, supporting continuous improvement and iteration as needed in each stage.

The flexibility of CRISP-DM allows for revisiting and refining early stages based on new insights or data, which is crucial for projects involving complex emotion-aware hate speech data analysis and interpretation. This iterative capability ensures that the project can adapt to unexpected challenges and evolving requirements, leading to better results over time. While SEMMA and KDD also allow for iterative processes, but lack the initial business understanding and final deployment stages, making them less suitable for project that need to align with the project's objectives.

CRISP-DM also faces some challenges, such as cumbersome documentation requirements. Despite these challenges, the advantages of CRISP-DM's structured, iterative approach make it the most appropriate method to achieve this project's goals.

3.2.3 Describe the phases, activities, techniques, and processes in each phase

Phase 1: Business Understanding

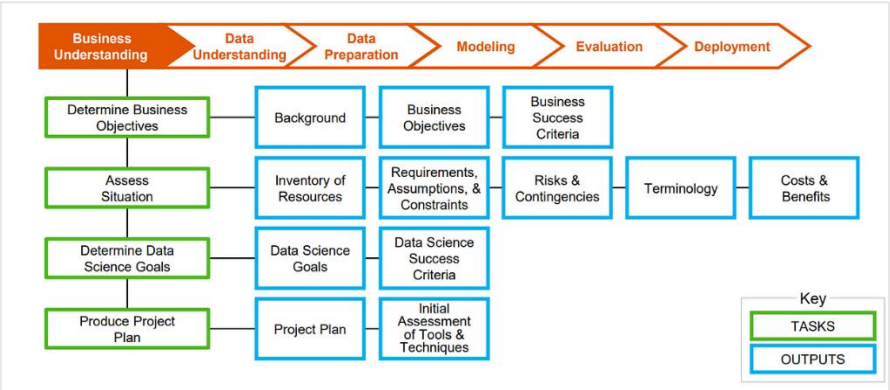


Figure 19: CRISP-DM Phase 1 Business Understanding (Luna, 2021)

In phase one, the business understanding phase is to determine business objectives, assess the situation, determine data science goals, and produce a project plan (Luna, 2021).

Determine Business Objectives

To ensure project success, it is crucial to clearly understand and define the business objectives and establish measurable criteria for success. This begins with understanding the needs of stakeholders. What stakeholders need from detecting hate speech with integrate emotions within user-generated content. Gathering stakeholders easily through interviews, surveys, and meetings helps to identify priorities and align the project with stakeholders' expectations. Defining the business goals involves outlining specific objectives, such as improving the accuracy of hate speech detection while incorporating emotion analysis to better understand user content. The project's structure must be organized and provide a detailed Work Breakdown Structure (WBS) developed using stakeholder feedback. A lesson from a PwC, indicates that only 2.5% of projects achieve full success due to issues failing to meet the project's goals, poor scope, and not meeting the deadlines and budget (Dwivedi, 2021).

Assess Situation

Evaluate the overall situation by reviewing the resource list, project requirements, assumptions, constraints, risks, and potential unexpected events. Begin by listing available datasets for analysis and the tools planned for use, such as Jupyter Notebook and VS Code. Identify specific requirements, including a minimum accuracy threshold for the model, as well as any project constraints, such as time limits or resource availability. Consider potential risks, such as data imbalance, overfitting, or the challenge of effectively balancing hate speech detection with emotional nuance. Weighed expected project outcomes, like enhanced hate speech detection accuracy, against the associated time and computational resource costs. Finally, outline the project's potential benefits, highlighting how it could contribute to a more nuanced and effective approach to content moderation.

Determine Data Science Goals

Determine the data science goals by outlining the project's objectives in technical terms and defining success criteria. Set specific goals, such as improving detection accuracy and examining the role of emotions in identifying hate speech. Establish success criteria through technical metrics, including F1-score, precision, recall, and accuracy. Justify the choice of models and methods by specifying the selected model, such as BERT or DNN for text analysis, and explain the model's relevance to both hate speech detection and emotion analysis,

emphasizing its suitability for capturing nuanced language patterns and emotional undertones in user-generated content.

Produce Project Plan

To accomplish the data science goals, draft a project plan that details the aims and procedures. Provide precise project phases, deadlines, necessary resources, and a predicted cost. Additionally, conduct an initial evaluation of the tools and technologies that will be used in the project.

Phase 2: Data Understanding

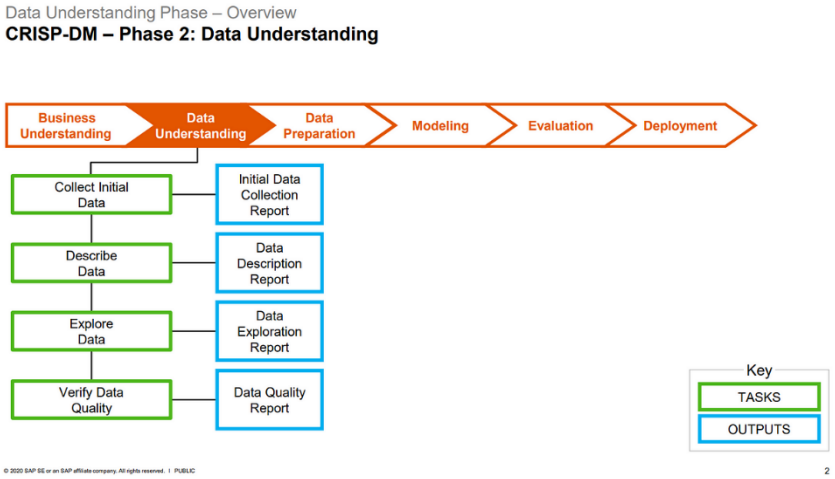


Figure 20: CRISP-DM Phase 2 Data Understanding (Luna, 2021)

The data understanding phase is divided into four stages, collect initial data, describe data, explore data, and verify data quality.

1. **Collect Initial Data**: This initial stage involves not only acquiring relevant datasets but also verifying their suitability for analytical requirements specific to the project. A detailed inventory of each dataset is prepared, documenting storage locations and acquisition methods. This ensures a structured approach to data management.

2. **Describe Data**: Once data is collected, a detailed description is conducted to outline each dataset's structure, contents, and primary attributes. This stage provides an essential overview, offering insights into the data types, format, and patterns or anomalies that may impact further analysis.

3. **Explore Data**: The following is an initial exploration of the data, providing an early understanding of the dataset's characteristics through Exploratory Data Analysis (EDA). This stage involves an examination of data distributions, identification of preliminary patterns, and formulation of initial hypotheses regarding potential relationships within the data. By applying EDA techniques such as summary statistics, data visualization, and correlation analysis, one can get insights into data trends, outliers, and underlying structures, which enables a clearer view of the data's key features and informs subsequent analysis and model-building phases. In addition, understanding the distribution of the target variables allows us to assess class balance, detect any skewness in the data, and identify potential biases.

4. **Verify Data Quality:** The final stage focuses on the quality of the data. Key quality metrics are examined, such as the presence of missing values, outliers, and data inconsistencies (Luna, 2021). If quality issues exist, propose possible solutions (Smart Vision Europe, 2017).

Phase 3: Data Preparation

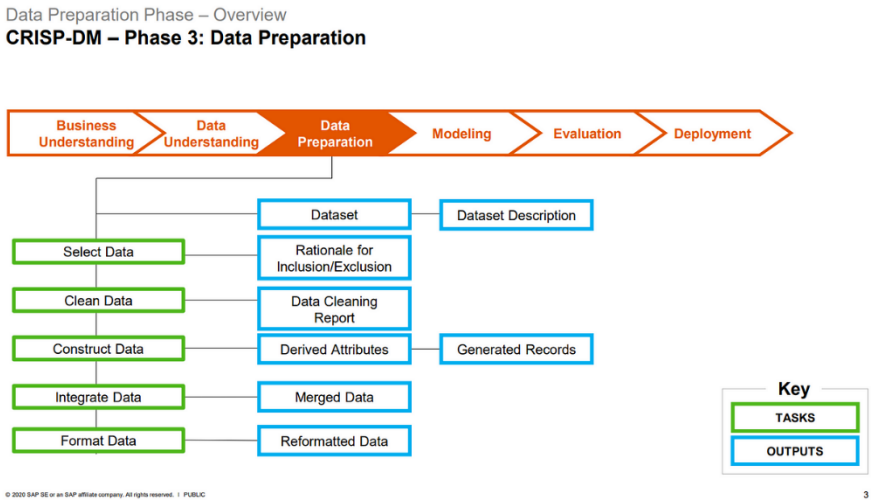


Figure 21: CRISP-DM Phase 3 Data Preparation (Luna, 2021)

Select Data

This step involves identifying the most relevant data for analysis based on the project's goals outlined in the Business Understanding phase. During selection, datasets are evaluated for relevance and usefulness in achieving the project objectives.

Clean Data

This includes handling missing values, correcting inconsistencies, and ensuring data completeness and accuracy to meet the requirements of the selected analysis techniques.

Construct Data

When building a dataset, data quality can be enhanced by creating new features or transforming existing features. For example, adding columns or features to capture the language, structure, and semantic information of the text. This additional information can make the dataset easier to understand, thereby providing more valuable support for natural language processing (NLP) analysis and modelling.

Integrate Data

Integration involves combining disparate data sources, such as combining text data with user demographic information or combining multiple records for a single entity into a single consolidated view. Use SQL commands such as JOIN, UNION, or MERGE to create a complete dataset with all relevant information for analysis. Consolidating datasets ensures that all necessary attributes are available in a single dataset for easy consumption, thereby increasing the consistency and comprehensiveness of the data for modelling and insights.

Format Data

In this final stage, prepare the data for modelling by applying transformations like standardization or normalization. These transformations adjust the data scale without changing its meaning, ensuring compatibility with modelling tools and enhancing model accuracy.

Phase 4: Modelling

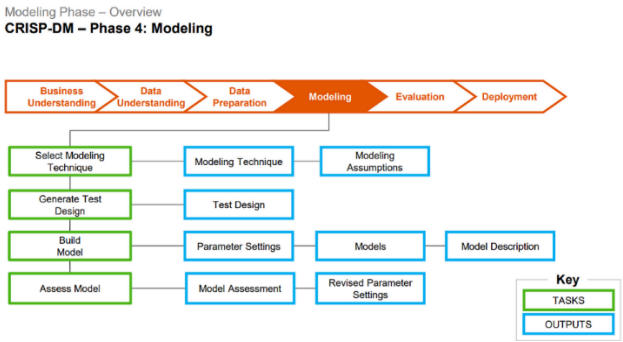


Figure 22: CRISP-DM Phase 4 Modelling (Luna, 2021)

According to (Luna, 2021) the Modelling phase involves selecting an appropriate modelling technique, designing a testing strategy, building the model, and assessing its performance in alignment with objectives for hate speech detection.

Select Modelling Techniques

Tasks	Description	Common Algorithms
Anomaly detection	Supervised or unsupervised models for detecting outliers	■ Distance-based ■ Density-based
Association analysis	An unsupervised model that looks for relationships between items in a set	■ Apriori ■ FP growth
Classification	A supervised model that estimates a categorical variable	■ Decision trees ■ K-nearest neighbors ■ Logistic regression ■ Naive Bayes ■ Neural networks ■ Support vector machines
Clustering	An unsupervised model that looks for meaningful groups within a data set	■ K-means ■ DBSCAN ■ Self-organizing maps
Regression	A supervised model that predicts a numerical variable	■ Simple linear regression ■ Multiple regression
Time series forecasting	A supervised model that predicts a future value of a variable	■ ARMA ■ ARIMA ■ Exponential smoothing

Figure 23: Data Analysis Algorithms (Richard O'Hara et al., 2023)

In the CRISP-DM Modelling phase, appropriate algorithms are selected based on the specific task requirements. For anomaly detection, distance-based and density-based models are common. Association analysis uses algorithms like Apriori to identify item relationships. Classification tasks rely on decision trees, logistic regression, and neural networks to estimate categorical outcomes. Clustering methods such as K-means group data into meaningful clusters. Regression tasks use linear and multiple regression to predict numerical values, while time series forecasting applies ARIMA and exponential smoothing to predict future values.

In selecting modelling techniques for text classification in data analysis, it is essential to consider both model performance and suitability for the task. Commonly used techniques include K-Nearest Neighbors (KNN), Support Vector Machines (SVM), and Random Forest (RF) (Li et al., 2022). These models excel at managing high-dimensional data, capturing non-linear features, and are well-suited for text classification tasks.

According to (Li et al., 2022) showing that eXtreme, XGBoost and LightGBM have gained popularity for their advanced performance and efficiency. These boosting techniques are

particularly powerful for large datasets and complex classification tasks. Both XGBoost and LightGBM have become strong choices for modern text classification projects, often providing excellent results due to model's capability to handle imbalanced data and optimize feature selection (Marzieh Babaeianjelodar et al., 2022).

Generate Test Design

This process begins by splitting the dataset into training, validation, and test sets, allowing distinct roles in model development (Luna, 2021). The training set is used for learning, the validation set is used for tuning, and the test set provides an unbiased evaluation of unseen data. A common practice is to split the data into two sets, with 80% allocated for training and 20% for testing. Some sampling techniques listed by (Shiksha, 2023), such as random, stratified sampling, and shuffle, are used to split the data into train and test.

When dividing data into training and test sets, the size of the training set is an essential consideration. A larger training set generally enhances model accuracy, but an excessively large set can increase training time and may lead to overfitting. Similarly, the test set size is important, as a larger test set typically yields more reliable evaluation results. However, if the test set is too large, it may be time-consuming and might not adequately represent the data distribution. It is crucial to maintain a balanced distribution of each class in both sets, as significant imbalances could skew the results (Shiksha, 2023).

Build Model

In the Build Model step, run the model using the prepared dataset to make sure it works properly. Once the model has been tested and proven to work properly, you need to record all the parameters used, including the specific values of each parameter and the reasons for choosing these values. This step is important because if you don't record these initial parameters, you may end up running the model again and again just because you forgot the previous parameter settings. Recording the parameters will help with subsequent model improvements and adjustments, making the entire process more efficient (Luna, 2021).

Assess Model

In this phase, the results of each model are summarized and ranked according to model's quality. The evaluation process involves analysing the performance of the models, viewing relevant

graphs, and examining the confusion matrix to fully understand the effectiveness and accuracy of each model.

Phase 5: Evaluation

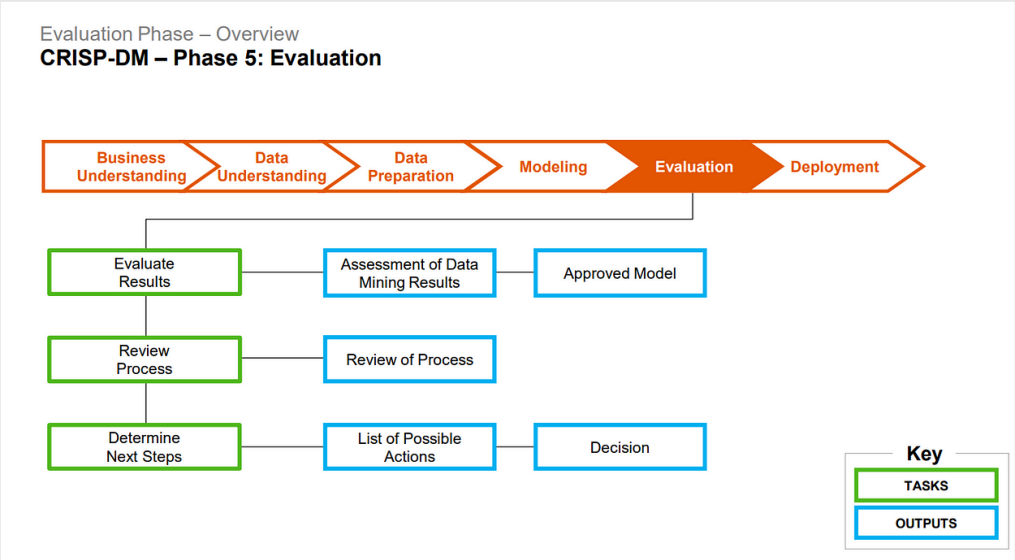


Figure 24: CRISP-DM Phase 5 Evaluation (Luna, 2021)

Evaluate Results

In the evaluation results step, it is necessary to evaluate to what extent the model has achieved the goal. And test these models in practical applications to verify the effectiveness of the model in real scenarios. This evaluation ensures that the model performs effectively both technically and in meeting the expected objectives. In model evaluation, (Shiksha, 2023) listed some metrics used to assess performance, including Accuracy, Precision, ROC-AUC score, Precision-Recall Score and F1 Score.

Evaluate Method: K-Fold Cross-Validation

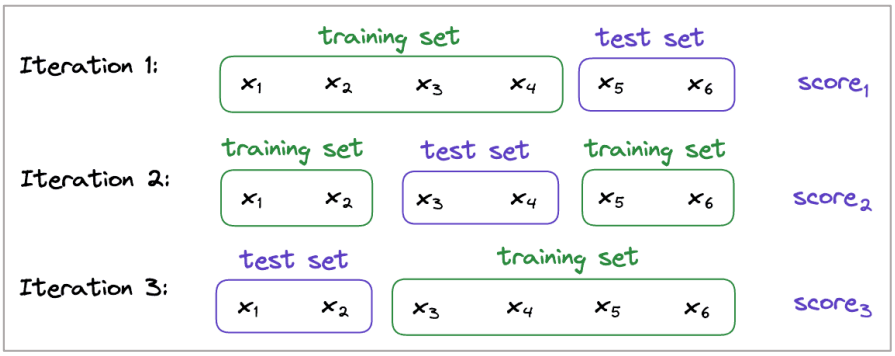


Figure 25: 5-Fold Cross-Validation (Antoniadis, 2022)

K-fold cross-validation is a widely used method for evaluating machine learning models, step one is to divide the dataset into several subsets, commonly referred to as k-folds. 5-fold or 10-fold cross-validation is frequently used during the training. During this process, each fold is assigned as a test set once, while the other k-1 folds are used for training. This process is repeated for a total of k iterations, ensuring that each subset is tested and evaluated. The final performance metric is computed as the average of all k evaluation results. Selecting the value of k requires careful consideration. A smaller k value results in larger test sets but fewer training data, potentially reducing the reliability of the evaluation. Conversely, a larger k value will make this K-Fold Cross-Validation the same as Leave-One-Out Cross-Validation, k equals the number of data points, and more training data, but significantly increases computational cost (Antoniadis, 2022). K-fold cross-validation is suitable for evaluating hate speech data because it accounts for potential class imbalances and variations in language patterns, which are common challenges in hate speech datasets. An accuracy rate between 80% and 90% is generally considered an optimal range.

Review Process

Through an in-depth review of the entire data mining work, ensure that no critical factors or tasks have been overlooked during the process. This step is to identify any possible quality assurance issues and ensure that all steps meet the requirements. Finally, the review results need to be summarized to indicate those activities that were omitted or need to be re-performed. This ensures the integrity and quality of the entire data mining process.

Determine Next Steps

Decide how to proceed with the project by evaluating the next steps. This section needs to list potential follow-up actions and analyse the pros and cons of each option. At the same time, it also needs to describe in detail how to proceed with the project to ensure that the next steps are clear and reasonable.

Phase 6: Deployment

According to Richard O'Hara et al. (2023), in the final stage of CRISP-DM, the information should be structured and presented in a manner that is clear and actionable for the intended audience. Depending on the needs of the project, the final output can be a simple report. The main tasks include planning deployment, developing monitoring and maintenance plans,

generating final reports, and conducting project reviews. Deployment and monitoring plans should be based directly on model insights to ensure that core issues are effectively addressed. According to a report by IMA (Institute of Management Accountants), good data visualization provides insights and tells a story that resonates with the audience. Through storytelling, information is easier to remember and understand.

3.3 Summary

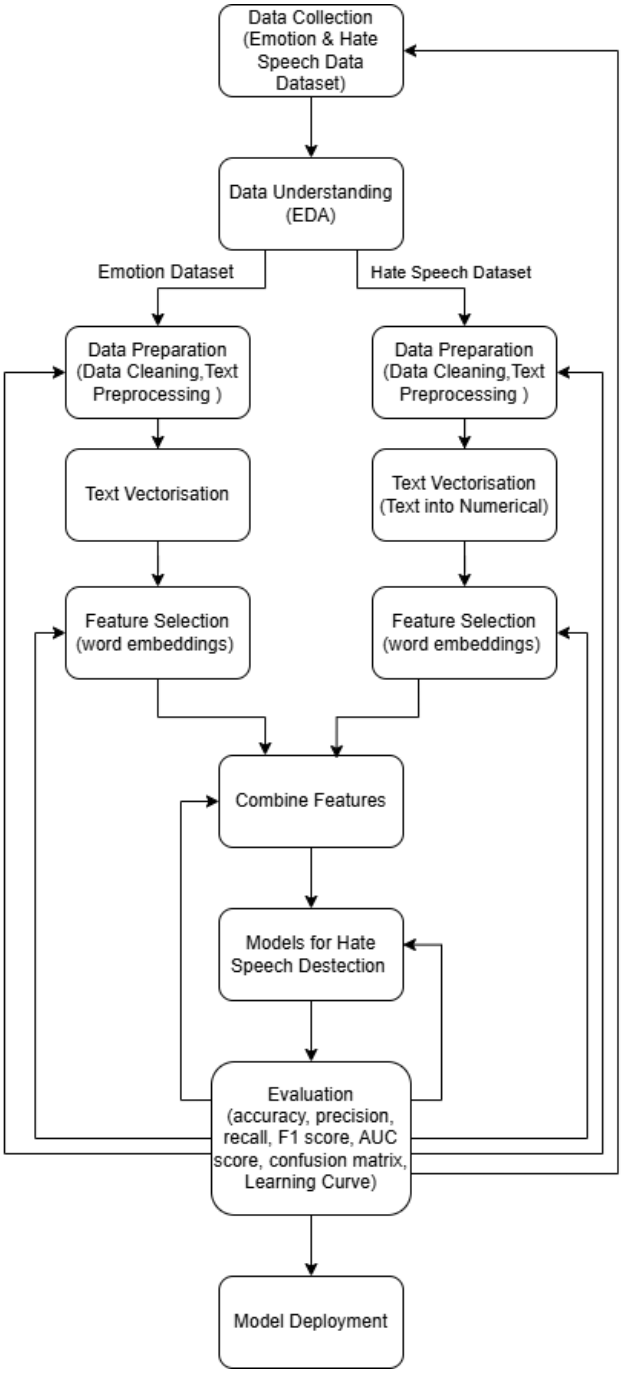


Figure 26: Overview of the Project Flow Chart

Chapter 3 outlines the steps of the CRISP-DM methodology and highlights key factors essential for project success and achieving project objectives. Successful implementation of these steps requires clear objectives, effective communication among stakeholders, and a thorough understanding of both the business problem and the data characteristics.

In addition, during the text preprocessing phase, it was observed that the hate speech dataset and the emotion dataset contain numerous Internet-specific words and abbreviations, which makes it necessary to create a customized abbreviation list specifically tailored to hate speech and emotion contexts.

Another significant challenge identified is the problem of data imbalance, which is also a core issue highlighted in the project problem statement. To solve this issue, plan to utilize various techniques such as resampling, oversampling minority classes or undersampling majority classes, employing data augmentation strategies, or leveraging algorithmic approaches like cost-sensitive learning. These methods will help create a more balanced representation of each class in the hate speech and emotion dataset.

CHAPTER 4: DESIGN AND IMPLEMENTATION

4.1 Introduction

This chapter introduces the sources of hate speech and emotion datasets, then combines datasets for data cleaning, including identifying and processing unknown words, removing unnecessary symbols and so on. This chapter also uses machine learning, deep learning, concept-level fusion, and multi-task learning (MTL) models building for hate speech detection.

4.2 Data Collection Source

Hate Speech Dataset

Table 3: Hate Speech Data Collection Source

Data Set Name	Dataset Explanation	Dataset Size	The Proportion of Target Column	Observation	Data Source	Data Source Link
Train Tweet	The dataset for this project originates from the GitHub repository "Twitter-Sentiment-Analysis" created by Roshan Sharma. This dataset is designed for sentiment analysis with a focus on identifying hate	31962 rows and 2 columns	Label 0: 92.99% Label 1: 7.01% Dataset Name: Train_Tweet Label Count 0 29720 1 2242 Label '1' is racist/sexist Label '0' is not racist/sexist	The dataset consists of labelled tweets, each assigned a binary label. A larger proportion of tweets are labelled as 0, indicating an imbalanced dataset.	GitHub	https://github.com/sharmaroshan/Twitter-Sentiment-Analysis

	speech in racist/sexist.					
Hate Speech Dataset	The dataset originates from the Hate Speech Dataset repository by Vicomtech on GitHub. This dataset was designed to support research in hate speech detection using Natural Language Processing (NLP) techniques.	10944 rows and 5 columns	Label noHate: 86.87%, Label hate: 10.93%, Label relation: 1.54%, Label idk/skip: 0.67% Label Count noHate 9507 hate 1196 relation 168 idk/skip 73	The dataset consists of labelled entries and associated text data, making it suitable for NLP-based hate speech detection tasks. This dataset stores text data in separate file_id.txt files and does not contain a direct "hate speech text" column within the main dataset file, so preprocessing will be required to prepare it for analysis. Dataset Name: Hate_Speech_Dataset Column Names: file_id user_id subforum_id num_contexts label	GitHub	https://github.com/Vicomtech/hate-speech-dataset/tree/master
Dynamically Generated Hate Dataset v0.2.3	This dataset was created by (Vidgen et al, 2021) through a human-and-model-in-the-loop process aimed at enhancing the detection of online hate speech.	41,144 rows and 13 columns	Hate: 53.90% Not Hate: 46.10% Label Count hate 22175 nothate 18969	This dataset has 40,000 entries, including around 15,000 challenging perturbations. Each hateful entry is accompanied by fine-grained labels specifying the type and target of hate. Notably, hateful entries constitute 53.90% of the	GitHub	https://github.com/bvidgen/Dynamically-Generated-Hate-Speech-Dataset/tree/main

				dataset, a higher proportion compared to similar datasets. Dataset Name: Dynamically Generated Hate Dataset v0.2.3 Column Names: | Data Types: Unnamed: 0 | int64 acl.id | object X1 | int64 text | object label | object type | object target | object level | object split | object round.base | int64 annotator | object round | object acl.id.matched | object		
Ethos Dataset Binary	This dataset designed to aid in detecting hate speech on social media. The dataset has two parts. The binary version, Ethos_Dataset_Binary.csv, contains 998 comments, each labeled to indicate whether it includes hate speech. The multi-label version, Ethos_Dataset_Multi_Label.csv, provides more detailed tags,	998 rows and 2 columns	Label >= 0.5 is hate Label < 0.5 is no hate Label Count isHate column >= 0.5: 433 isHate column < 0.5: 565	This dataset contains binary labels (hate and non-hate) for hate speech detection, helping to identify and analyse instances of hate. 565 of them do not contain hate speech, while the rest of them, 433 contain. Dataset Name: Ethos_Dataset_Binary Column Names: Data Types: comment | object isHate | float64	GitHub	https://github.com/intelligence-csd-auth-gr/Ethos-Hate-Speech-Dataset/tree/master/ethos/ethos_data

	such as whether the comment incites violence, whether it targets an individual or a group, and additional categories that identify the specific nature of the hate speech, such as gender, race, or religion. In this project only used Ethos_Dataset_Binary.csv.					

Emotion Dataset

Table 4: Emotion Data Collection Source

Data Set Name	Dataset Explanation	Dataset Size	The Proportion of Target Column	Observation	Data Source	Data Source Link
Emotion	This dataset prepared by (Bhavik Jikadara, 2023), this dataset collected from Twitter text messages labelled with six different emotions:	416809 rows and 2 columns (text and emotion label)	Emotion label column: Joy: 33.84% Sadness: 29.07% Anger: 13.75% Fear: 11.45% Love: 8.29% Surprise: 3.59%	This dataset contains two columns and six emotion labels. The distribution of labels is imbalanced, with Love and Surprise each accounting for less than 10% of the total data.	Kaggle	https://www.kaggle.com/datasets/bhavikjikadara/emotions-dataset/data

	sadness, joy, love, anger, fear, and surprise, is a multi-class classification dataset.		Label Joy : 141067 rows (33.84%) Label Sadness : 121187 rows (29.07%) Label Anger : 57317 rows (13.75%) Label Fear : 47712 rows (11.45%) Label Love : 34554 rows (8.29%) Label Surprise : 14972 rows (3.59%)	Dataset Name: Emotions Column Names: Data Types: text | object label | int64		

Conclusion of Data Collection

The hate speech dataset was collected from four different sources on GitHub, and one emotion dataset was derived from the Kaggle, there are six emotion labels.

If the current datasets are found to be insufficient or inadequate in the future, additional datasets will be incorporated to enhance the analysis.

4.3 Exploratory Data Analysis (EDA)

4.3.1 Dataset Description

Hate Speech Dataset

1. Dynamically Generated Hate Dataset

```
file_1_path = r".\Ori_Hate_Speech_Dataset\Dynamically Generated Hate Dataset v0.2.3.csv"
data1= pd.read_csv(file_1_path, encoding='ISO-8859-1')
data1.head(10)
```

✓ 0.1s

	Unnamed: 0	acl.id	X1	text	label	type	target	level	split	round.base	annotator	round	acl.id.matched
0	1	acl1	1	dalits are lowlives	hate	notgiven	notgiven	original	train	1	GNZuCtwed3	1	NaN
1	2	acl10	2	gay people are a burden to our society	hate	notgiven	notgiven	original	train	1	GNZuCtwed3	1	NaN
2	3	acl100	3	Arabs are not welcome	hate	notgiven	notgiven	original	train	1	vDe7GN0NrL	1	NaN
3	4	acl1000	4	I'm not saying we should actually eliminate he...	hate	notgiven	notgiven	original	train	1	oemYWm1Tjg	1	NaN
4	5	acl10000	5	bananas are for black people	hate	notgiven	notgiven	original	test	1	QiOKkCi7F8	1	NaN
5	6	acl10001	6	asian people are the shit	nothate	none	none	original	test	1	GNZuCtwed3	1	NaN
6	7	acl10002	7	women can not reproduce kids but men can in al...	hate	notgiven	notgiven	original	test	1	vDe7GN0NrL	1	NaN
7	8	acl10003	8	i hate wearing black in the summer!	nothate	none	none	original	test	1	LqLKTtrOmx	1	NaN
8	9	acl10004	9	Who cares what Chinese people think?	hate	notgiven	notgiven	original	test	1	CAgNlUizNm	1	NaN
9	10	acl10005	10	i try to punch a letter box when i see one	hate	notgiven	notgiven	original	test	1	LqLKTtrOmx	1	NaN

Figure 27: 1st Hate Speech Dataset: Dynamically Generated Hate Dataset

```
file_1_path = r".\Ori_Hate_Speech_Dataset\Dynamically Generated Hate Dataset v0.2.3.csv"
data1 = pd.read_csv(file_1_path, encoding='ISO-8859-1')

# Checking unique values in the 'label' column
unique_labels = data1['label'].unique()
print(unique_labels, len(unique_labels))
data1.info()
```

✓ 0.2s

```
['hate' 'nothate'] 2
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41144 entries, 0 to 41143
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Unnamed: 0      41144 non-null  int64
 1   acl.id          41144 non-null  object
 2   X1              41144 non-null  int64
 3   text            41144 non-null  object
 4   label           41144 non-null  object
 5   type            41144 non-null  object
 6   target          41136 non-null  object
 7   level           41144 non-null  object
 8   split           41144 non-null  object
 9   round.base      41144 non-null  int64
 10  annotator       41144 non-null  object
 11  round           41144 non-null  object
 12  acl.id.matched  30098 non-null  object
dtypes: int64(3), object(10)
memory usage: 4.1+ MB
```

Figure 28: Overview of Dynamically Generated Hate Dataset

The Dynamically Generated Hate Dataset consists of 13 columns. The target column ("label") includes two values, "hate" and "nohate". Other columns provide metadata useful for model building. For example, the "split" column marks data as train, test, or dev, while the "level" column indicates whether a row is original or perturbation data. Additionally, the "type" column categorizes the hate speech data by type.

2. "Ethos Binary" Hate Speech Dataset

```
file_2_path = r".\Ori_Hate_Speech_Dataset\Ethos_Dataset_Binary.csv"
data2= pd.read_csv(file_2_path, encoding='ISO-8859-1')
data2.head(10)
```

✓ 0.0s

	comment	isHate
0	You should know women's sports are a joke	1.0
1	You look like Sloth with deeper Down□s syndrome	1.0
2	You look like Russian and speak like Indian. B...	1.0
3	Women deserve to be abused, I guess.	1.0
4	Women are made for making babies and cooking d...	1.0
5	why haven't you faggots taken the bi-pill yet	1.0
6	Why don□t you go beck your god to murder you? ...	1.0
7	why does that african american monkey hold gun...	1.0
8	Why does Bulgaria exist as a country? It's one...	1.0
9	Why do negroes tear down their own neighborhoo...	1.0

Figure 29: 2nd Hate Speech Dataset: Ethos Binary Dataset

```
file_2_path = r".\Ori_Hate_Speech_Dataset\Ethos_Dataset_Binary.csv"
data2= pd.read_csv(file_2_path, encoding='ISO-8859-1')
unique_labels = data2['isHate'].unique()
unique_labels, len(unique_labels)
```

✓ 0.0s

```
(array([1.        , 0.98387097, 0.98360656, 0.97826087, 0.97333333,
        0.96666667, 0.95454545, 0.94545455, 0.9375    , 0.90384615,
        0.85714286, 0.8490566 , 0.84615385, 0.83333333, 0.82142857,
        0.75      , 0.72222222, 0.67857143, 0.66666667, 0.60344828,
        0.53061225, 0.5       , 0.4       , 0.33333333, 0.30232558,
        0.296875  , 0.25      , 0.2       , 0.16666667, 0.16071429,
        0.15254237, 0.11111111, 0.10344828, 0.09090909, 0.03896104,
        0.03773585, 0.03174603, 0.03030303, 0.02985075, 0.02631579,
        0.01886793, 0.01639344, 0.        ]),
 43)
```

```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 998 entries, 0 to 997
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   comment  998 non-null    object
 1   isHate   998 non-null    float64
dtypes: float64(1), object(1)
memory usage: 15.7+ KB
```

Figure 30: Overview of Ethos Binary Dataset

The Ethos Binary Dataset contains 998 rows and includes two columns, one for hate speech text and a target column with float values.

3. Vicomtech's Hate Speech Dataset

```
file_3_path = r".\Ori_Hate_Speech_Dataset\Hate_Speech_Dataset.csv"
data3= pd.read_csv(file_3_path, encoding='ISO-8859-1')
data3.head(10)
```

✓ 0.0s

	file_id	user_id	subforum_id	num_contexts	label
0	12834217_1	572066	1346	0	noHate
1	12834217_2	572066	1346	0	noHate
2	12834217_3	572066	1346	0	noHate
3	12834217_4	572066	1346	0	hate
4	12834217_5	572066	1346	0	noHate
5	12834217_6	572066	1346	0	noHate
6	12834217_7	572066	1346	0	noHate
7	12834217_8	572066	1346	0	noHate
8	12834217_9	572066	1346	0	noHate
9	12834217_10	572066	1346	0	noHate

Figure 31: 3rd Hate Speech Dataset: Hate Speech Dataset

```
file_3_path =r".\Ori_Hate_Speech_Dataset\Hate_Speech_Dataset.csv"
data3= pd.read_csv(file_3_path, encoding='ISO-8859-1')
unique_labels = data3['label'].unique()
unique_labels, len(unique_labels)
```

```
(array(['noHate', 'hate', 'idk/skip', 'relation'], dtype=object), 4)
```

```
data3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10944 entries, 0 to 10943
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   file_id       10944 non-null  object
 1   user_id       10944 non-null  int64 
 2   subforum_id   10944 non-null  int64 
 3   num_contexts  10944 non-null  int64 
 4   label         10944 non-null  object
dtypes: int64(3), object(2)
memory usage: 427.6+ KB
```

Figure 32: Overview of Hate Speech Dataset

This dataset contains 10,944 rows across 5 columns. The target column, labelled "label," includes four unique values, "noHate", "hate", "idk/skip", and "relation".

4. "Train Tweet" Hate Speech Dataset

```
file_4_path = r".\Ori_Hate_Speech_Dataset\Train_Tweet.csv"
data4= pd.read_csv(file_4_path, encoding='ISO-8859-1')
data4.head(10)
```

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s...
1	2	0	@user @user thanks for #lyft credit i can't us...
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ...
4	5	0	factsguide: society now #motivation
5	6	0	[2/2] huge fan fare and big talking before the...
6	7	0	@user camping tomorrow @user @user @user @use...
7	8	0	the next school year is the year for exams.Ã°Â...
8	9	0	we won!!! love the land!!! #allin #cavs #champ...
9	10	0	@user @user welcome here ! i'm it's so #gr...

Figure 33: 4th Hate Speech Dataset: Train Tweet

```
file_4_path = r".\Ori_Hate_Speech_Dataset\Train_Tweet.csv"
data4= pd.read_csv(file_4_path, encoding='ISO-8859-1')
unique_labels = data4['label'].unique()
unique_labels, len(unique_labels)
```

```
(array([0, 1]), 2)
```

```
data4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31962 entries, 0 to 31961
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      31962 non-null  int64
 1   label   31962 non-null  int64
 2   tweet   31962 non-null  object
dtypes: int64(2), object(1)
memory usage: 749.2+ KB
```

Figure 34: Overview of Train Tweet Dataset

This dataset contains 31,962 rows and 3 columns. The target column is in binary format, where 1 represents hate speech, and 0 represents non-hate speech.

In these four hate speech datasets, there are various columns and types of information. However, for this project on "hate speech detection", only need the hate speech text (text column) and target labels column. Each target column has different values, is needed to standardize the values across all target columns before merging the datasets.

Emotion Dataset

```
#·Dataset·Info
import pandas as pd
file_path = r".\Ori_Emotion_Dataset\Emotions.csv"
data= pd.read_csv(file_path, encoding='ISO-8859-1')
data.head(10)
```

	text	label
0	i just feel really helpless and heavy hearted	4
1	ive enjoyed being able to slouch about relax a...	0
2	i gave up my internship with the dmrg and am f...	4
3	i dont know i feel so lost	0
4	i am a kindergarten teacher and i am thoroughl...	4
5	i was beginning to feel quite disheartened	0
6	i would think that whomever would be lucky eno...	2
7	i fear that they won t ever feel that deliciou...	1
8	im forever taking some time out to have a lie ...	5
9	i can still lose the weight without feeling de...	0

Figure 35: Top 10 Rows of Emotion Dataset

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 416809 entries, 0 to 416808
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   text    416809 non-null  object
 1   label   416809 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 6.4+ MB
```

Figure 36: Overview of Emotion Dataset

```
file_path = r".\Ori_Emotion_Dataset\Emotions.csv"
data= pd.read_csv(file_path, encoding='ISO-8859-1')
unique_labels = data['label'].unique()
unique_labels, len(unique_labels)
```

```
(array([4, 0, 2, 1, 5, 3], dtype=int64), 6)
```

Figure 37: Count Unique Emotion Label

This emotion dataset contains 416809 rows and two columns (text and emotion label column). In emotion label column, there are six different categories emotion label is a multi-class classification dataset. The dataset has no missing values and it already lowercasing all the text data.

4.3.2 Proportion of the Target Column

Hate Speech Distribution

```
plt.pie(data['label'].value_counts(), labels=data['label'].value_counts().index, autopct='%1.1f%%')
plt.title("Distribution of Hate and NoHate Labels")
plt.show()
```

✓ 0.3s

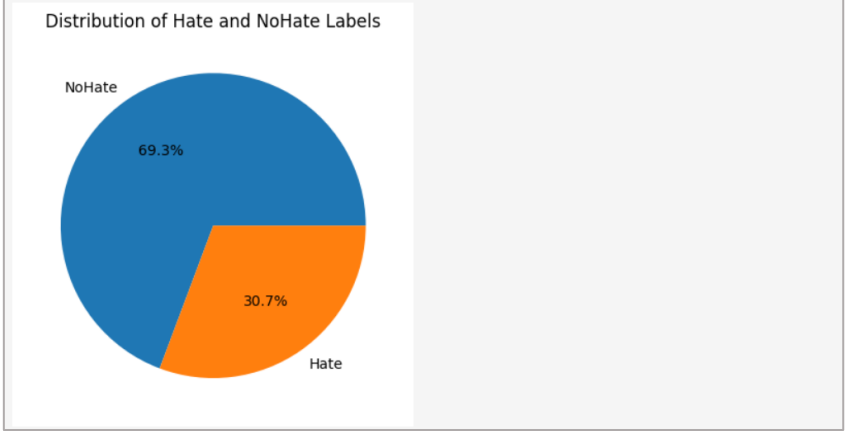


Figure 38: Pie Chart of Distribution of Hate and No Hate

After merging the four datasets into a single dataset, there are 69.3% of the data is labelled as "No Hate", while 30.7% is labelled as "Hate".

Emotion Tag Distribution

```python
import pandas as pd
file_path = r".\Ori_Emotion_Dataset\Emotions.csv"
data = pd.read_csv(file_path, encoding='ISO-8859-1')

label_counts = data['label'].value_counts()
total_rows = len(data)
label_mapping = {
    0: "Sadness",
    1: "Joy",
    2: "Love",
    3: "Anger",
    4: "Fear",
    5: "Surprise"
}
for label, count in label_counts.items():
    label_name = label_mapping.get(label, "Unknown")
    percentage = (count / total_rows) * 100
    print(f"Label {label_name:<10}: {count:<6} rows ({percentage:.2f}%)")

```

```
Label Joy       : 141067 rows (33.84%)
Label Sadness   : 121187 rows (29.07%)
Label Anger     : 57317  rows (13.75%)
Label Fear      : 47712  rows (11.45%)
Label Love      : 34554  rows (8.29%)
Label Surprise  : 14972  rows (3.59%)
```

Figure 39: Number rows of each Emotion Tag

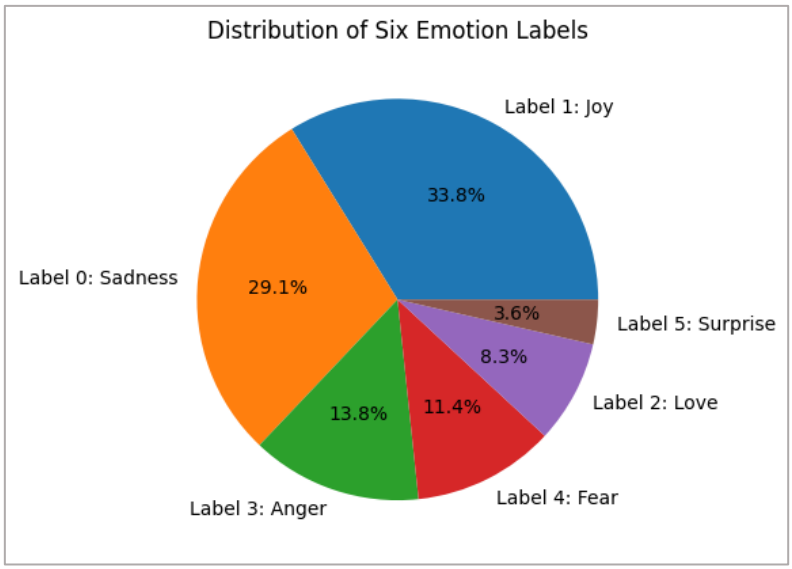


Figure 40: Pie Chart of Emotion Distribution

This pie chart provides a clear visualization of the distribution of emotion labels across the dataset. The emotions are Joy (33.8%), Sadness (29.1%), and Anger (13.8%), Fear (11.4%), Love (8.3%), Surprise (3.6%).

4.3.3 Check Missing Value

Hate Speech Dataset

```
# info of dataset
file_path = r"C:\Users\Asus\OneDrive\桌面\103\Hate Data Excel\Text Preprocessing File\3 Merge Dataset\3.Combine ALL Dataset.csv"
data = pd.read_csv(file_path, encoding='ISO-8859-1')
data.info()
print("\n")

# missing_text_rows = data[data['text'].isnull()]
# print(missing_text_rows)
```

✓ 0.1s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 85048 entries, 0 to 85047
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    85048 non-null  object
 1   label   85048 non-null  int64
dtypes: int64(1), object(1)
memory usage: 1.3+ MB
```

Figure 41: Check NULL Value – Hate Speech

The output indicates that the dataset consists of 85048 rows and two columns: the “text” column (data type: object) and the “label” column (data type: integer). This combined dataset contains no missing values.

Emotion Dataset

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 416809 entries, 0 to 416808
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   text    416809 non-null  object
 1   label   416809 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 6.4+ MB
```

Figure 42: Check NULL Value - Emotion

The figure shows that there is no empty column in this emotion dataset.

4.3.4 Check Duplicated

Hate Speech Dataset

```
# check duplicated
file_path = r".\Pre_Hate_Dataset\3_CombineAllHateSpeech_Dataset.csv"
data = pd.read_csv(file_path, encoding='ISO-8859-1')

duplicate_rows = data[data.duplicated()]
print(f"Number of duplicated rows: {duplicate_rows.shape[0]}")
if not duplicate_rows.empty:
    print(duplicate_rows)
```

✓ 0.2s

```
Number of duplicated rows: 2611
                                                    text  label
41047                          Blacks are such pondlife      1
41048                          Blacks are such pondlife      1
41051                          Blacks are such pondlife      1
41055                          Blacks are such pondlife      1
41094                            Muslims are a disgrace      1
...                                                 ...    ...
84731  100 amazing health benefits of cucumbers!    #...      0
84749  #model   i love u take with u all the time in ...      0
84775  @user #feminismiscancer #feminismisterrorism #...      1
84796  i've just posted a new blog: #secondlife #lone...      0
84805  @user #sikh #temple vandalised in in #calgary,...      1

[2611 rows x 2 columns]
```

Figure 43: Check Duplicated Value – Hate Speech

This shows that there are a total of 2,611 duplicated rows across the hate speech datasets.

Emotion Dataset

```
# check duplicated before drop unsed column
file_path = r".\Ori_Emotion_Dataset\Emotions.csv"
data = pd.read_csv(file_path, encoding='ISO-8859-1', low_memory=False)

num_duplicates = data.duplicated().sum()
print(f"Number of duplicated rows: {num_duplicates}")
```

✓ 1.4s

```
Number of duplicated rows: 686
```

Figure 44: Check Duplicated Before Dropping Unused Columns – Emotion Dataset

There are 686 duplicated data before dropping the unused emotion label.

```
# Check Dup
num_duplicates = data.duplicated().sum()
print(f"Number of duplicated rows: {num_duplicates}")
```

✓ 0.1s

```
Number of duplicated rows: 418
```

Figure 45: Check Duplicated After Dropping Unused Column – Emotion Dataset

There are 418 duplicated data after dropping the unused emotion label.

4.4 Data Pre-processing

Text preprocessing is an important step in converting raw text data into a clean and organized format for use in NLP tasks. According to research by (Siino et al., 2024), data preprocessing can involve a variety of techniques, such as removing punctuation and stop words, applying lemmatization, converting text to lowercase, and expanding contractions.

Research by (Siino et al., 2024) doing text pre-processing on the IMDB dataset reveals that the choice of preprocessing techniques can significantly impact model performance. According to the research, lowercasing alone or in conjunction with stop word removal and lowercasing produced the greatest accuracy of 0.857, while stemming, stop word removal, and lowercasing produced an accuracy of 0.853. This suggests that, although stemming is a widely studied and commonly used preprocessing technique, it may not contribute positively to model performance and is not involved in any of the best-performing combinations in this IMDB dataset.

4.4.1 Import Libraries

```
# Standard libraries
import os
import re
import ast

# Data manipulation
import pandas as pd

# NLP libraries
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

# Emoji handling
import emoji
```

Figure 46: Import Libraries – Text Preprocessing

The libraries imported purposes in text preprocessing. Standard libraries "os" manage file paths and directories, "re" handles regular expressions for pattern matching and text cleaning, and "ast" safely converts strings back into Python objects. "pandas" is used to efficiently process and analyse structured data in tabular format. NLP-specific libraries include "nltk" for natural language processing tasks, with "WordNetLemmatizer" and "wordnet" used in lemmatization to reduce words into base form while retaining meaning. Lastly, the "emoji" library enables the processing and handling of emojis in text.

4.4.2 Import text column data

Hate Speech Dataset

```
input_csv_path = r".\Ori_Hate_Speech_Dataset\Hate_Speech_Dataset.csv"
file_id_folder = r".\HateData_FileID"
output_csv_path = r".\Ori_Hate_Speech_Dataset\Hate_Speech_Dataset_Text.csv"

df = pd.read_csv(input_csv_path, encoding='ISO-8859-1')

texts = []
for file_id in df['file_id']:
    file_path = os.path.join(file_id_folder, f"{file_id}.txt")
    if os.path.exists(file_path):
        with open(file_path, 'r', encoding='ISO-8859-1') as file:
            text = file.read()
    else:
        text = ""
    texts.append(text)

# create new column for text
df['text'] = texts
df.to_csv(output_csv_path, index=False, encoding='ISO-8859-1')
print("Successful Done", output_csv_path)
```

```
Successful Done .\Ori_Hate_Speech_Dataset\Hate_Speech_Dataset_Text.csv
```

Figure 47: Code for Importing and Merging Text Content into the Hate Speech Dataset

	A	B	C	D	E	F	G	H	I	J
1	file_id	user_id	subforum_	num_conte	label	text				
2	12834217_	572066	1346	0	noHate	As of March 13th , 2014 , the booklet had been downloa				
3	12834217_	572066	1346	0	noHate	In order to help increase the booklets downloads , it wou				
4	12834217_	572066	1346	0	noHate	(Simply copy and paste the following text into your YouT				
5	12834217_	572066	1346	0	hate	Click below for a FREE download of a colorfully illustrate				
6	12834217_	572066	1346	0	noHate	Click on the `` DOWNLOAD (7.42 MB) " green banner li				
7	12834217_	572066	1346	0	noHate	Booklet updated on Feb. 14th , 2014 .				
8	12834217_	572066	1346	0	noHate	(Now with over 18,300 Downloads.)				
9	12834217_	572066	1346	0	noHate	PDF file : http://www.mediafire.com/download/7p3p3g				
10	12834217_	572066	1346	0	noHate	Then why not simply copy this text (& links) and paste it i				
11	12834217_	572066	1346	0	noHate	Thank you in advance. :) Download the youtube `` desc				
12	12834493_	572058	1396	0	hate	She may or may not be a Jew but she 's certainly stupid ,				
13	12843221_	575003	1396	0	noHate	Thank you for the info. i will deffinetly look into them .				

Figure 48: Dataset with Merged Text Column

This original dataset includes essential metadata such as file_id, user_id, subforum_num, and label but lacks the textual information necessary for effective hate speech detection. By merging the text content into a new “text" column, enabling a more comprehensive approach to analysis.

4.4.3 Convert All Labels Column into Binary Format

Hate Speech Dataset

```
file_1_path = r".\Pre_Hate_Dataset\1_Drop_Unsed_Column\Dynamically_Drop.csv"
file_2_path = r".\Pre_Hate_Dataset\1_Drop_Unsed_Column\Ethos_Drop.csv"
file_3_path = r".\Pre_Hate_Dataset\1_Drop_Unsed_Column\Hate_Speech_Dataset_Drop.csv"
# file_4_path = r".\Pre_Hate_Dataset\1_Drop_Unsed_Column\Train_Tweet_Drop.csv"

output_file_1 = r".\Pre_Hate_Dataset\2_Convert_Label_To_Binary\Dynamically_Binary.csv"
output_file_2 = r".\Pre_Hate_Dataset\2_Convert_Label_To_Binary\Ethos_Binary.csv"
output_file_3 = r".\Pre_Hate_Dataset\2_Convert_Label_To_Binary\Hate_Speech_Dataset_Binary.csv"
# output_file_4 =  r".\Pre_Hate_Dataset\2_Convert_Label_To_Binary\Train_Tweet_Standardize.csv"


def process_label_column_DynamicallyDataset(file_path,output_csv_path):
    df = pd.read_csv(file_path, encoding='ISO-8859-1')
    df['label'] = df['label'].apply(lambda x: 1 if x == 'hate' else 0) #(Python, n.d.)
    df.to_csv(output_csv_path, index=False, encoding='ISO-8859-1')
    print(f"Processed and saved: {output_csv_path}")
process_label_column_DynamicallyDataset(file_1_path,output_file_1)  # Dynamically_Generated_Hate_Dataset

def process_label_column_HSDataset(file_path,output_csv_path):
    df = pd.read_csv(file_path, encoding='ISO-8859-1')
    df_filtered = df[~df['label'].isin(["idk/skip", "relation"])] #remove label as "idk/skip" & "relation"
    df_filtered['label'] = df_filtered['label'].apply(lambda x: 1 if x == 'hate' else 0)
    df_filtered.to_csv(output_csv_path, index=False, encoding='ISO-8859-1')
    print(f"Filtered dataset saved to: {output_csv_path}")
process_label_column_HSDataset(file_3_path,output_file_3)  # Hate_Speech_Dataset_with_Text

def process_label_column_ethosDataset(file_path,output_csv_path):
    df = pd.read_csv(file_path, encoding='ISO-8859-1')
    df['label'] = df['label'].apply(lambda x: 1 if x >=0.5 else 0)
    df.to_csv(output_csv_path, index=False, encoding='ISO-8859-1')
    print(f"Processed and saved: {output_csv_path}")
process_label_column_ethosDataset(file_2_path,output_file_2)  # Ethos_Dataset_Binary_Dataset
```

Figure 49: Code for Convert Labels Column to Binary Format

In this step, all target label columns are standardized by converting them into a consistent binary format to ensure uniformity across the dataset. This standardization simplifies

interpretation and analysis of the target data. The "train tweet" dataset does not require reformatting as it already adheres to the correct binary label format. For the "Hate_Speech_Dataset," which contains four target labels, the labels "idk/skip" and "relation" are removed, retaining only "noHate" and "hate" for further processing and analysis.

Table 5: Output of Convert Labels Column into Binary Format (Dataset 1)

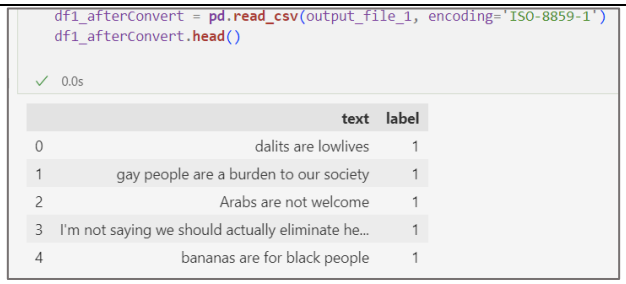
	"Dynamically_Generated_Hate_Dataset_Filtered" Dataset
Before Convert to Binary Format	 text label 0 dalits are lowlives hate 1 gay people are a burden to our society hate 2 Arabs are not welcome hate 3 I'm not saying we should actually eliminate he... hate 4 bananas are for black people hate
After Convert to Binary Format	 text label 0 dalits are lowlives 1 1 gay people are a burden to our society 1 2 Arabs are not welcome 1 3 I'm not saying we should actually eliminate he... 1 4 bananas are for black people 1

Table 6: Output of Convert Labels Column into Binary Format (Dataset 2)

	"Hate_Speech_Dataset_with_Text_Filtered" Dataset
Before Convert to Binary Format	text label 428 Do you mean ..... noHate 429 It is better to be hated for what you are than... noHate 430 Andre Gide French critic , essayist , & noveli... noHate 431 Yeah, we could do with the same amount of blac... hate 432 No . noHate 433 Rule Britannia . noHate 434 Yes true It reminds me of this incident Muslim... noHate 435 surely gotta be more than that. town to town y... noHate 436 Personal Card for Teacher - Johann Warzecha in... noHate 437 The name of the village was changed to Dramast... noHate 438 You are telling me that 78 % of kids under 6 i... relation 439 Jeeze its worst than the UK . relation
After Convert to Binary Format	text label 428 No . 0 429 Rule Britannia . 0 430 Yes true It reminds me of this incident Muslim... 0 431 surely gotta be more than that. town to town y... 0 432 Personal Card for Teacher - Johann Warzecha in... 0 433 The name of the village was changed to Dramast... 0 434 First I 'm not from Moldavia ... 0 435 Second , I really hope one day you chauvinists... 0 436 Budapest 1919 , Trianon 1920 , ring a bell ? 2... 0 437 There is no way you are taking our ancestral h... 0 438 Oh that 's just great . 0 439 I enjoy when a Brit comes in and tells Bulgari... 0

Table 7: Output of Convert Labels Column into Binary Format (Dataset 3)

	"Ethos_Dataset_Binary_Standardize" Dataset

Before Convert to Binary Format	`df1 = pd.read_csv(file_2_path, encoding='ISO-8859-1')`
	`print(df1.iloc[428:440])`
	✓ 0.0s
	text label
	428 All people are fucked up because of these quee... 0.500000
	429 Actually THIS WEIRD PEOPLE are HARMFUL! They n... 0.500000
	430 ABU DA BI ABA DU BA BA HAMUD! See? I□m an arab... 0.500000
	431 a homo? so you fuck your brother instead of yo... 0.500000
	432 (((They))) will be chased out of every white c... 0.500000
	433 men and women are not equal. irrational contr... 0.400000
	434 I'm very sad about this fucking situation with... 0.400000
	435 You look like someone who would do an electric... 0.333333
	436 You can ask me to call you a male, but you'll ... 0.333333
	437 Yo, if I□m going to hell for being homosexual,... 0.333333
	438 Why are people so dumb? The police were yellin... 0.333333
	439 White police officers are always the victims. 0.333333

After Convert to Binary Format	`df1_afterConvert = pd.read_csv(output_file_2, encoding='ISO-8859-1')`
	`print(df1_afterConvert.iloc[428:440])`
	✓ 0.0s
	text label
	428 All people are fucked up because of these quee... 1
	429 Actually THIS WEIRD PEOPLE are HARMFUL! They n... 1
	430 ABU DA BI ABA DU BA BA HAMUD! See? I□m an arab... 1
	431 a homo? so you fuck your brother instead of yo... 1
	432 (((They))) will be chased out of every white c... 1
	433 men and women are not equal. irrational contr... 0
	434 I'm very sad about this fucking situation with... 0
	435 You look like someone who would do an electric... 0
	436 You can ask me to call you a male, but you'll ... 0
	437 Yo, if I□m going to hell for being homosexual,... 0
	438 Why are people so dumb? The police were yellin... 0
	439 White police officers are always the victims. 0

4.4.4 Drop Unused Column

<u>Hate Speech Dataset</u>

```
file_1_path = r".\Ori_Hate_Speech_Dataset\Dynamically Generated Hate Dataset v0.2.3.csv"
file_2_path = r".\Ori_Hate_Speech_Dataset\Ethos_Dataset_Binary.csv"
file_3_path = r".\Ori_Hate_Speech_Dataset\Hate_Speech_Dataset_Text.csv"
file_4_path = r".\Ori_Hate_Speech_Dataset\Train_Tweet.csv"


output_file_1 = r".\Pre_Hate_Dataset\1_Drop_Unsed_Column\Dynamically_Drop.csv"
output_file_2 = r".\Pre_Hate_Dataset\1_Drop_Unsed_Column\Ethos_Drop.csv"
output_file_3 = r".\Pre_Hate_Dataset\1_Drop_Unsed_Column\Hate_Speech_Dataset_Drop.csv"
output_file_4 = r".\Pre_Hate_Dataset\1_Drop_Unsed_Column\Train_Tweet_Drop.csv"


df1 = pd.read_csv(file_1_path, encoding='ISO-8859-1', low_memory=False)[['text', 'label']]
df1.to_csv(output_file_1, index=False, encoding='ISO-8859-1')

df2 = pd.read_csv(file_2_path, encoding='ISO-8859-1', low_memory=False)[['comment', 'isHate']]
df2.columns = ['text', 'label']
df2.to_csv(output_file_2, index=False, encoding='ISO-8859-1')

df3 = pd.read_csv(file_3_path, encoding='ISO-8859-1', low_memory=False)[['text', 'label']]
df3.to_csv(output_file_3, index=False, encoding='ISO-8859-1')

df4 = pd.read_csv(file_4_path, encoding='ISO-8859-1', low_memory=False)[['tweet', 'label']]
df4.columns = ['text', 'label']
df4.to_csv(output_file_4, index=False, encoding='ISO-8859-1')
```

Figure 50: Drop Unused Column Code

In this step, remove unneeded columns from the datasets, including Hate_Speech_Dataset, Dynamically Generated Hate Dataset v0.2.3, Train_Tweet, and Ethos_Dataset_Binary. This cleanup ensures that only relevant columns are retained for further analysis, only maintain "text" column and "label" column for each hate speech dataset.

Emotional Dataset

```
file_path = r".\Ori_Emotion_Dataset\Emotions.csv"
data = pd.read_csv(file_path, encoding='ISO-8859-1')

# sadness (0), joy (1), love (2), anger (3), fear (4), and surprise (5)
# drop sadness , love , surprise emotion label
data = data.drop(data[data['label'].isin([0, 2, 5])].index)
output_file =r".\Pre_Emotion_Dataset\1_drop_column.csv"
data.to_csv(output_file, index=False, encoding='ISO-8859-1')
```

Figure 51: Drop Unused Column

```
Label Joy      : 141067 rows (33.84%)
Label Sadness  : 121187 rows (29.07%)
Label Anger    : 57317  rows (13.75%)
Label Fear     : 47712  rows (11.45%)
Label Love     : 34554  rows (8.29%)
Label Surprise : 14972  rows (3.59%)
```

Figure 52: Emotion Label Distribution

In this emotion dataset, remove labels sadness, love and surprise are removed due to their low representation, with love (3.59%) and surprise (8.29%) being insufficient for the model to learn the features. Sadness is excluded as it is less relevant to hate speech, whereas anger and fear are more closely related. Joy is retained as a neutral emotion for comparison. Ensures a more balanced dataset.

4.4.5 Merge Dataset

Merge Hate Speech Dataset

```
# combine all dataset

file_1_path = r".\Pre_Hate_Dataset\2_Convert_Label_To_Binary\Dynamically_Binary.csv"
file_2_path = r".\Pre_Hate_Dataset\2_Convert_Label_To_Binary\Ethos_Binary.csv"
file_3_path = r".\Pre_Hate_Dataset\2_Convert_Label_To_Binary\Hate_Speech_Dataset_Binary.csv"
file_4_path =  r".\Pre_Hate_Dataset\2_Convert_Label_To_Binary\Train_Tweet_Binary.csv"

df1 = pd.read_csv(file_1_path, encoding='ISO-8859-1')
df2 = pd.read_csv(file_2_path, encoding='ISO-8859-1')
df3 = pd.read_csv(file_3_path, encoding='ISO-8859-1')
df4 = pd.read_csv(file_4_path, encoding='ISO-8859-1')

combined_df = pd.concat([df1, df2, df3, df4], ignore_index=True)
print("Combined DataFrame shape:", combined_df.shape)

output_file_path = r".\Pre_Hate_Dataset\3_CombineAllHateSpeech_Dataset.csv"
combined_df.to_csv(output_file_path, index=False, encoding='ISO-8859-1')
print(f"Combined dataset saved to: {output_file_path}")
```

```
Combined DataFrame shape: (84807, 2)
Combined dataset saved to: .\Pre_Hate_Dataset\3_CombineAllHateSpeech_Dataset.csv
```

Figure 53: Merge Dataset

The merged dataset consolidates multiple datasets into a single, unified structure, which is used to streamline analysis and processing. Combining data from multiple sources consolidates all useful information into a single dataset, facilitating easier data cleaning and exploration, modeling, and insight generation.

4.4.6 Mapping Emotion Label

Emotion Dataset

```
# Emotion Label Remapping: joy (1 → 0), anger (3 → 1), fear (4 → 2)
label_mapping = {1: "joy → 0", 3: "anger → 1", 4: "fear → 2"}

# Print label distribution before mapping
print("📌 Original label distribution:")
print(data['label'].value_counts())

# Apply label mapping
data['label'] = data['label'].replace({1: 0, 3: 1, 4: 2})

# Print label distribution after mapping
print("\n📌 New label distribution:")
print(data['label'].value_counts())

# Save processed dataset
output_file = r".\Pre_Emotion_Dataset\3_label_remapping.csv"
data.to_csv(output_file, index=False, encoding='ISO-8859-1')
```

```
✓ 0.4s

📌 Original label distribution:
label
1    140779
3     57235
4     47664
Name: count, dtype: int64

📌 New label distribution:
label
0    140779
1     57235
2     47664
Name: count, dtype: int64
```

Figure 54: Code of Mapping Emotion Label

Emotion label remapping is performed to standardize label encoding by assigning sequential numerical values to emotion categories. The labels are remapped as follows: 0 for joy, 1 for anger, and 2 for fear.

4.4.7 Remove Duplicated Data

Hate Speech Dataset

```
# 4. remove duplicated
file_path = r".\Pre_Hate_Dataset\3_CombineAllHateSpeech_Dataset.csv"
data = pd.read_csv(file_path, encoding='ISO-8859-1')

duplicate_rows = data[data.duplicated()]
print(f"Number of duplicated rows: {duplicate_rows.shape[0]}")

data = data.drop_duplicates()

print("After Remove Dup")
duplicate_rows = data[data.duplicated()]
print(f"Number of duplicated rows: {duplicate_rows.shape[0]}")

output_file = r".\Pre_Hate_Dataset\4_Remove_Duplicated.csv"
data.to_csv(output_file, index=False, encoding='ISO-8859-1')
```

```
Number of duplicated rows: 2611
After Remove Dup
Number of duplicated rows: 0
```

Figure 55: Code of Remove Duplicated Rows in Hate Speech Dataset

The hate speech dataset initially contained 2,611 duplicate rows. After dropping these duplicates, the dataset now has zero duplicate entries, ensuring data consistency and accuracy for further analysis and processing.

Emotion Dataset

```
# Check Dup
num_duplicates = data.duplicated().sum()
print(f"Number of duplicated rows: {num_duplicates}")
```

✓ 0.1s

```
Number of duplicated rows: 418
```

```
# remove duplicated
data = data.drop_duplicates()
num_duplicates_after = data.duplicated().sum()
print(f"Number of duplicated rows after removal: {num_duplicates_after}")
data.info()
output_file =r".\Pre_Emotion_Dataset\2_remove_duplicated.csv"
data.to_csv(output_file, index=False, encoding='ISO-8859-1')
```

✓ 0.7s

```
Number of duplicated rows after removal: 0
<class 'pandas.core.frame.DataFrame'>
Index: 245678 entries, 0 to 416807
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   text    245678 non-null  object
 1   label   245678 non-null  int64
dtypes: int64(1), object(1)
memory usage: 5.6+ MB
```

Figure 56: Code of Remove Duplicated Data

There are 418 rows of data that are duplicated. After removing duplicated rows, the emotion dataset has 245678 rows of data.

4.4.8 Text Cleaning and Normalization

4.4.8.1 Lowercasing

```
df['text'] = df['text'].str.lower()
print(df['text'].head())
```

Figure 57: Lowercasing in Hate Speech and Emotion Dataset

Lowercase all the text data to ensure consistency and uniformity in the dataset. This step helps to avoid treating the same word with different cases ("Happy" and "happy") as separate entities during text analysis and model training.

4.4.8.2 Remove URL & HTML & Emoji

Hate Speech & Emotion Dataset

```
file_path = r".\Pre_Hate_Dataset\5_Lowercassing.csv"
df = pd.read_csv(file_path, encoding='ISO-8859-1')

# 6. Remove URLs
df['text'] = df['text'].apply(lambda x: re.sub(r'(http|https|www)\S+', '', str(x)))

# 7. Remove HTML
html_tags_pattern = r'<.*?>'
# sub(pattern,replace,text)
df['text'] = df['text'].apply(lambda x: re.sub(html_tags_pattern, '', str(x)))
df['text'] = df['text'].apply(lambda x: re.sub(r'\S+html\b', '', str(x)))

# 8. Remove emojis
def remove_emojis(text):
    text = emoji.demojize(text)
    text = re.sub(r':\w+:', '', text)
    return text
df['text'] = df['text'].apply(remove_emojis)

output_file_path = r".\Pre_Hate_Dataset\6_8_removeURL_HTML_Emoji.csv"
df.to_csv(output_file_path, index=False, encoding='ISO-8859-1')
```

Figure 58: Code of Remove URL & HTML & Emoji

This step aims to reduce noise in the dataset, such as URL links. This project focuses on detecting hate speech if straight away removing all non-English words could lead to the loss of critical content, as many users express hate-related terms in unconventional spellings, like "ihate." Therefore, cannot deleting non-English words outright would risk removing a significant amount of relevant hate speech content.

4.4.8.3 Common symbol substitutions

Hate Speech

```
# 9.Common symbol substitutions
file_path = r".\Pre_Hate_Dataset\6_8_removeURL_HTML_Emoji.csv"
df = pd.read_csv(file_path, encoding='ISO-8859-1')

replace_dict = {
    '@': 'a',
    '$': 's',
    '$$':'ss',
    '0': 'o',
    '3': 'e',
    '1': 'i',
    '5': 's',
    '7': 't',
    '4': 'a',
    '9' : 'g',
}
def replace_symbols(text):
    text = str(text)
    for symbol, letter in replace_dict.items():
        # between the alp: (?<=[A-Za-z])symbol(?=[A-Za-z])
        # after the alp: (?<=[A-Za-z])symbol
        # before the alp: symbol(?=[A-Za-z])
        pattern = rf'(?<=[A-Za-z]){re.escape(symbol)}(?=[A-Za-z])|(?<=[A-Za-z]){re.escape(symbol)}|{re.escape(symbol)}(?=[A-Za-z])'
        text = re.sub(pattern, letter, text)
    return text

df['text'] = df['text'].apply(replace_symbols)

output_file_path = r".\Pre_Hate_Dataset\9_Symbol_Substitutions.csv"
df.to_csv(output_file_path, index=False, encoding="utf-8")
print(f"Data has been saved to {output_file_path}")
```

Figure 59: Code for Symbol and Number replacement

Symbol replacement is a key step in data preprocessing because many hate speech data contain non-standard characters or symbol replacements, such as "@" for "a", "$" for "s", etc, which brings challenges to accurate text analysis, natural language processing (NLP), and modelling. Therefore, replacing these symbols with standard letters can make the text more consistent and standardized, which helps improve the model's understanding ability and the accuracy of analysis. This code does not replace all the symbols or numbers to the alphabet, only replace which symbols are between the alphabet or before and after the alphabet.

4.4.8.4 Replace Abbreviations

Hate Speech Dataset

```python
# Abbreviations List base on unknow word visualization
abbreviations = {
    # it's, we're,i'll,"let's": "let us",
    "it's":"it is",
    "we're":"were are",
    "let's":"let us",
    "i'll":"i will",
}
# 9. Replace Abbreviations
def replace_abbreviations(text):
    text = str(text)
    for abbr, full_form in abbreviations.items():
        text = re.sub(r'\b' + re.escape(abbr) + r'\b', full_form, text)
    return str(text)
df['text'] = df['text'].apply(replace_abbreviations)
```

Figure 60: Code of Replace Abbreviations Before Remove Punctuation

```python
# Abbreviations List base on unknow word visualization
abbreviations = {
    "auser" : "",
    "werent":"were not","arent": "are not",
    "isnt": "is not",
    "cant": "can not",
    "shes": "she is","hes": "he is",
    "youre": "you are",
    "youll": "you will",
    "youve": "you have",
    "weve": "we have",
    "yall":"you all",
    "theyre": "they are",
    "theyve": "they have",
    "doesnt": "does not",
    "dont":"do not",
    "didnt": "did not",
    "wont": "will not",
    "wouldnt": "would not",
    "shouldnt": "should not",
    "couldnt": "could not",
    "im": "i am",
    "iam": "i am",
    "ive": "i have",
    "id": "i would",
    "wth":"what the heal", "wtf":"what the fuck",
    "fk":"fuck", "f**k":"fuck","fu*k":"fuck", "f*ck":"fuck","fck":"fuck","fcking":"fucking",
    "cuz":"because", "bcuz":"because","becuz":"because",
    "bihday":"birthday",
    "etc":"et cetera",
    "selfie":"self portrait photograph",
    "lol":"laughing out loud",
    "lmao":"laughing my ass off",
    "forex":" foreign exchange",
    "lgbt":"transgender",
    "blm":"black lives matter",
    "obama":"Barack Obama",
    "omg":"oh my god",
    "ppl":"people",
    "fathersday":"father day",
}
# 9. Replace Abbreviations
def replace_abbreviations(text):
    text = str(text)
    for abbr, full_form in abbreviations.items():
        text = re.sub(r'\b' + re.escape(abbr) + r'\b', full_form, text)
    return str(text)
df['text'] = df['text'].apply(replace_abbreviations)
```

Figure 61: Code of Replace Abbreviations After Remove Punctuation

Emotion Dataset

```
# Abbreviations List Get from (Pallavi Padav, 2024) and base on EDA unknown word
abbreviations = {
    "arent": "are not","werent":"were not",
    "isnt": "is not","wasnt":"was not",
    "wont": "will not",
    "wouldve":"would have",
    "couldve":"could have",
    "couldnt": "could not",
    "wouldnt": "would not",
    "shouldnt": "should not",
    "cant": "can not",
    "hadnt":"had not",
    "hasnt":"has not",
    "shes": "she is","hes": "he is",
    "youre": "you are","youve": "you have", "yall":"you all","youll":"you will",
    "theyve":"they have", "theyd":"they had",
    "doesnt": "does not","dont": "do not","didnt": "did not",
    "im": "i am", "she's": "she is","he's": "he is",
    "ive": "i have", "id": "i would",
    "itll":"it will", "itd":"it had",
    "wth":"what the heal","wtf":"what the fuck",
    "fk":"fuck","fked":"fuck","fuk":"fuck","fck":"fuck", "fcking":"fucking","fuckin":"fuck",
    "thats":"that is",
    "jk":"just kidding",
    "cuz":"because",
    "bcuz":"because",
    "fb":"facebook",
    "u":"you",
    "lol":"laughing out loud",
    "lmao":"laughing my ass off",
    "omg ":"oh my god",
    "bro":"brother",
    "tbh":"to be honest",
    "idk":"i do not know",
    "congrats":"congratulations",
    "upvote":"up vote",
    "downvote":"down vote","downvoted":"down vote",
    "btw":"by the way",
    "etc":"et cetera",
    "img":"image",
    "href":"",
    "http":"",
    "src":"",
    "ww":"",
}
```

```
def replace_abbreviations(text):
    text = str(text)
    for abbr, full_form in abbreviations.items():
        text = re.sub(r'\b' + re.escape(abbr) + r'\b', full_form, text)
    return str(text)

df['text'] = df['text'].apply(replace_abbreviations)
```

Figure 62: Code of Replace Abbreviations – Emotion Dataset

Based on the visualization of unknown words (4.5.5 Unknown Word Visualization) in the hate speech dataset and the emotion dataset, create separate abbreviation lists for each dataset. Nowadays, online communication often includes the use of abbreviations and slang such as “wtf” and “asap”. To ensure the consistency and interpretability of the dataset, replacing such abbreviations with their full forms is an essential step.

This normalization helps align informal language with a standard vocabulary, improving the model’s ability to understand the words and process the text accurately. Replace abbreviations in the dataset using \b around abbreviations to ensure the pattern matches only when

abbreviations appear as a standalone word, not as part of another word. For example, if the abbreviation is “cat,” \bcat\b would match “cat” but would not match “category” or “scatter.”

4.4.8.5 Remove ASCII Characters & Punctuation & Excessive Whitespace & Number Rows & Elongation

Hate Speech Dataset & Emotion Dataset

```
# 11. Remove ASCII characters & delete unused punctuation (reserve .,!?)
# df['text'] = df['text'].apply(lambda x: re.sub(r'[^A-Za-z0-9\s.,!?]', '', str(x)))
df['text'] = df['text'].apply(lambda x: re.sub(r'[^A-Za-z0-9\s]', '', str(x)))

# 12. Remove excessive whitespace
# .strip() removes any leading or trailing whitespace from the text
df['text'] = df['text'].apply(lambda x: re.sub(r'\s+', ' ', x).strip())

# 13.Remove only number rows
only_numbers_df = df[df['text'].astype(str).str.strip().str.isdigit()]
# print(only_numbers_df)
df = df[~df['text'].astype(str).str.strip().str.isdigit()] #turn all data into string (astype(str), remove space (strip()), check is whole string is digit)

# 14.Remove repeated punctuation
# df['text'] = df['text'].apply(lambda x: re.sub(r'([.!?])\s*\1+', r'\1', x))

# 15.Removing elongation (example: goodddddddddd)
df['text'] = df['text'].apply(lambda x: re.sub(r'(.)\1{2,}', r'\1\1', x))
```

Figure 63: Code for Remove ASCII Characters, Punctuation, White Space, Whole Number Rows, Elongation

Remove emojis, excessive whitespace, rows containing only numbers, and repeated punctuation to further clean the dataset and enhance data quality.

Remove Repeated Punctuation

Certain punctuation marks, such as periods, question marks, and exclamation marks, can express emotions. In text processing, some models will replace repeated punctuation marks with a representative label before deleting them, for example, replacing "???" with "multiQuestionMark". This processing method is to avoid word segmentation problems when the model processes these repeated punctuation marks. But if there is a sentence "Are you sure???", if there is no space before "???", the model may treat "sure???" as a word (token). This will cause the same sentence to generate different word lists in various situations, which may affect the behaviour and effect of the classifier (Siino et al., 2024).

Remove Elongation

Users use character repetition to emphasize and express their emotions. If repeated characters are not simplified to one, the classifier may interpret them as different words and longer words may be underestimated because the word appears less frequently in the text (Siino et al., 2024).

4.4.8.6 Tokenization

Hate Speech Dataset & Emotion Dataset

```
filteredTokens = []
for token in df['text']:
    token = str(token)
    wordtokens = nltk.tokenize.word_tokenize(token)
    filteredTokens.append(wordtokens)
df['text']=filteredTokens
```

Figure 64: Code for Data Tokenization

Tokenizing each word streamlines the processes of removing stop words and performing lemmatization. In this step, use the NLTK "word_tokenize" function to split sentences into individual words for more effective preprocessing.

4.4.8.7 Remove Stop-Words

Hate Speech Dataset & Emotion Dataset

```
# 17. Remove Stop Word
file_path = r".\Pre_Hate_Dataset\16_Tokenization.csv"
df = pd.read_csv(file_path, encoding='ISO-8859-1')

# Convert strings back to lists, stored as strings in the CSV file
df['text'] = df['text'].apply(lambda x: ast.literal_eval(x) if isinstance(x, str) else x) #(Yadav, 2023)

stopTokens = nltk.corpus.stopwords.words("english")
stopTokens.remove('not')
stopTokens.remove('no')

def removeStopWord(words):
    return [word for word in words if word.lower() not in stopTokens]
df['text'] = df['text'].apply(removeStopWord)
```

Figure 65: Code for Remove Stop Word

Removing stop words is crucial because if not removed, common words like "a", "and" and "the" will dominate the frequency analysis, as these are frequently used in English sentences. This can negatively impact the model's ability to capture meaningful emotion features, as the model might incorrectly associate the presence of these common words with specific emotions or hate speech. However, words like "not" and "no" are retained, those words play a critical role in conveying negation and sentiment, which are essential for accurate classification. By eliminating stop words, the dataset becomes more focused on words representing emotional expressions, improving the model's accuracy and effectiveness.

4.4.8.8 Lemmatization

Hate Speech Dataset & Emotion Dataset

```python
# 18. Lemmatization
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
import pandas as pd
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger') #used for Part-of-Speech (POS) tagging

lemmatizer = WordNetLemmatizer()
# file_path = r".\Pre_Hate_Dataset\17_Remove_StopWord.csv"
# df = pd.read_csv(file_path, encoding='ISO-8859-1')

def get_pos_tagging(word):
    #[0][1]:('running', 'VBG')
    #[0][1][0]:('V')
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V": wordnet.VERB, "R": wordnet.ADV} #need this for wordnet cuz wordnet only have 4 postag
    return tag_dict.get(tag, wordnet.NOUN)  # Default to noun if no match


def lemmatize_text(text):
    lemmatized_words = [lemmatizer.lemmatize(word, get_pos_tagging(word)) for word in text]
    return ' '.join(lemmatized_words)

df['text'] = df['text'].apply(lemmatize_text)

output_file_path = r".\Pre_Hate_Dataset\19_Lemmatization.csv"
df.to_csv(output_file_path, index=False, encoding="utf-8")
```

Figure 66: Code for Lemmatization

Lemmatization was chosen over stemming in this project due to its ability to preserve semantic accuracy and contextual meaning, which are critical for analysing user-generated content. Unlike stemming, which simply removes suffixes without considering the grammatical role or meaning of the word, lemmatization reduces words to proper base form while maintaining the linguistic context. Stemming reduces unrelated words such as “news” and “new” to the same root form “new”, which would result in a loss of important semantic distinctions. Therefore, lemmatization ensures a more accurate and meaningful text representation, making it the preferred choice for this project.

4.4.8.9 Drop Empty Text Data

Hate Speech Dataset & Emotion Dataset

```python
# drop na
file_path = r".\Pre_Hate_Dataset\19_Lemmatization.csv"
data = pd.read_csv(file_path, encoding='ISO-8859-1')

print("\nMissing values per column:\n", data.isnull().sum())
data = data.dropna(subset=['text'])

print("\nMissing values per column after dropping NaN:\n", data.isnull().sum())

output_file =r".\Pre_Hate_Dataset\20_Hate_Final.csv"
data.to_csv(output_file, index=False, encoding='ISO-8859-1')
```

Figure 67: Drop Empty Value in Text Column

The original dataset does not contain any empty or null values in the text column. However, after applying text preprocessing steps, some rows become empty. Drop an empty column to prevent errors in the code execution.

4.5 Data Understanding

4.5.1 Dataset Description After Text Preprocessing

Hate Speech Dataset Overview

```
Before Text Preprocessing:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 84807 entries, 0 to 84806
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    84807 non-null  object
 1   label   84807 non-null  int64
dtypes: int64(1), object(1)
memory usage: 1.3+ MB


After Text Preprocessing:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82064 entries, 0 to 82063
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    82064 non-null  object
 1   label   82064 non-null  int64
dtypes: int64(1), object(1)
memory usage: 1.3+ MB
```

Figure 68: Overview Hate Speech Dataset

After text preprocessing, the number of rows in the hate speech dataset reduced from 84,807 to 82,064. This reduction occurred due to the removal of irrelevant or unusable data, such as empty rows resulting from preprocessing steps like eliminating URLs, HTML, and other non-text elements.

Emotion Dataset Overview

```
import pandas as pd
file_path = r".\Pre_Emotion_Dataset\8_Lemmatization.csv"
data = pd.read_csv(file_path, encoding='ISO-8859-1')
print("After Text Preprocessing:")
data.info()
```

✓ 0.7s

```
After Text Preprocessing:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 245666 entries, 0 to 245665
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   text    245666 non-null  object
 1   label   245666 non-null  int64
dtypes: int64(1), object(1)
memory usage: 3.7+ MB
```

Figure 69: Emotion Dataset Overview After Text Preprocessing

After the preprocessing steps, the dataset contains 245666 rows and 2 columns.

4.5.2 Hate Speech Distribution

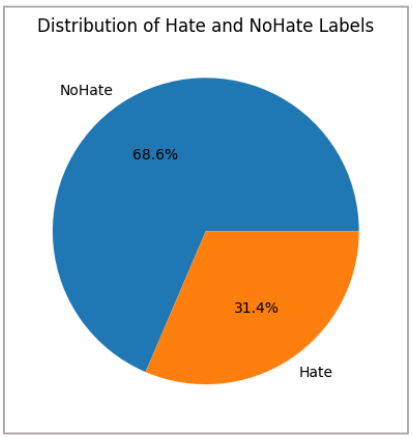


Figure 70: Pie Chart: Percentage of "Hate" and "No Hate"

After text preprocessing, there are 68.6% of the data is labelled as "No Hate," while 31.4% is labelled as "Hate."

4.5.3 Emotion Distribution

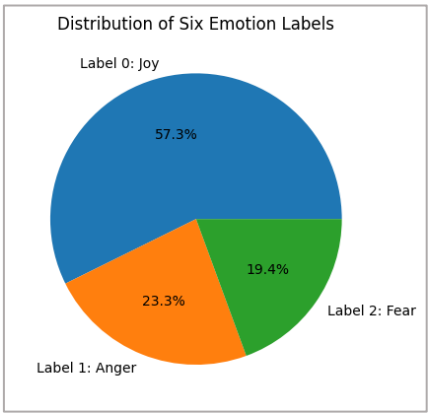


Figure 71: Emotion Distribution Pie Chart

```
Label Joy       : 140776 rows (57.30%)
Label Anger     : 57229  rows (23.30%)
Label Love      : 47661  rows (19.40%)
```

Figure 72: Number of Rows in Each Emotion Label

This bar chart illustrates the distribution of emotions. The label "Joy" has the highest proportion, accounting for 57.3% of the data. "Anger" and "Fear" have distributions of 23.3% and 19.4%, respectively, with a relatively close gap. Given this imbalance, undersampling may be considered when building the model to ensure better performance across all emotion categories.

4.5.4 Word Frequency Visualization

Hate Speech Dataset

Hate Speech Word Frequency (Whole Dataset)

Figure 73: Word Frequency Visualization in Whole Hate Speech Dataset

```
Most common words:
 [('not', 23504), ('people', 10012), ('like', 7150), ('get', 7112), ('woman', 6359), ('fuck', 5574), ('go', 5249), ('black', 5150), ('white', 4442), ('would', 4430)]
```

Figure 74: Frequency Word in Hate Speech Dataset

This figure illustrates that “not”, “people”, “woman”, “fuck”, “black”, and “white” are among the top 10 most frequently appearing words in the hate speech dataset.

Hate Speech Word Frequency (Hate Speech Only)

```
Most common words (label == 1):
 [('not', 10815), ('people', 4552), ('woman', 4017), ('like', 3116), ('black', 3091), ('get', 2649), ('white', 2248), ('fuck', 2228), ('would', 2096), ('want', 1844)]
```

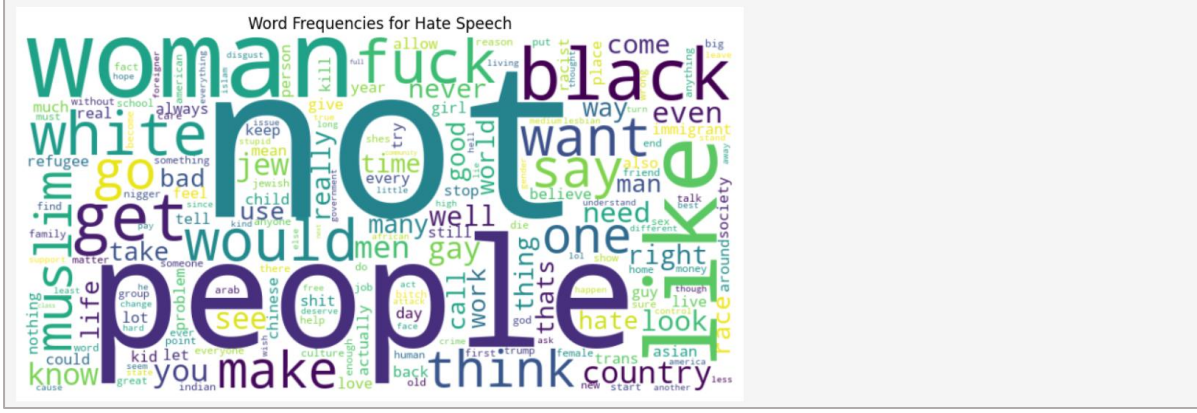


Figure 75: Word Frequency in Label as Hate Speech

Hate Speech Word Frequency (Non-Hate Speech)

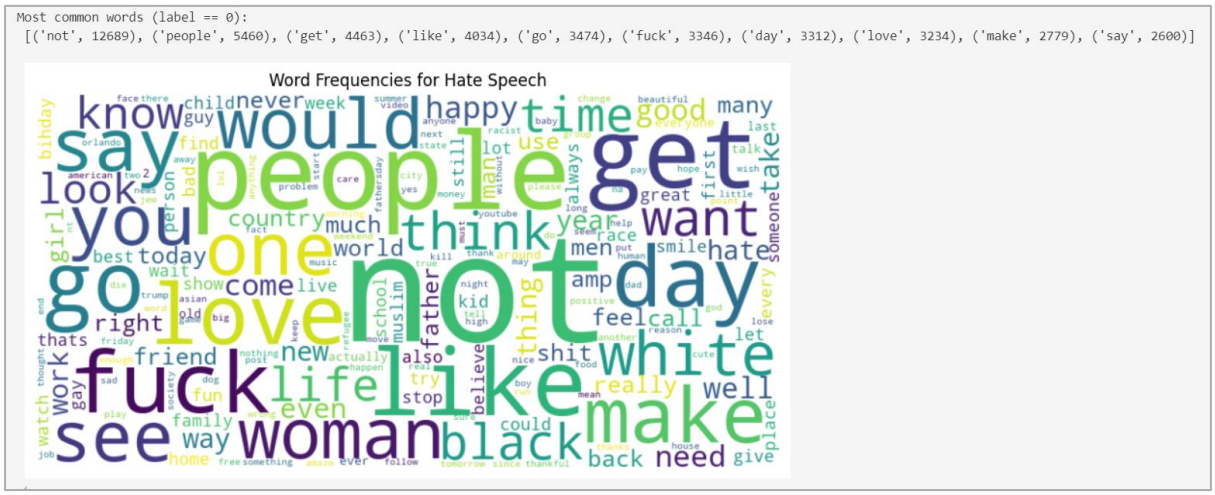


Figure 76: Word Frequency in Label as Non-Hate Speech

From the high-frequency words in both hate and non-hate contexts, we observe that terms like "people", "like", and "fuck" appear frequently in both categories. This demonstrates that relying solely on high-frequency words to determine whether a sentence constitutes hate speech is unreliable, as these words can occur in varying contexts with different meanings and implications.

Emotion Dataset

Word Frequency in Emotion Dataset

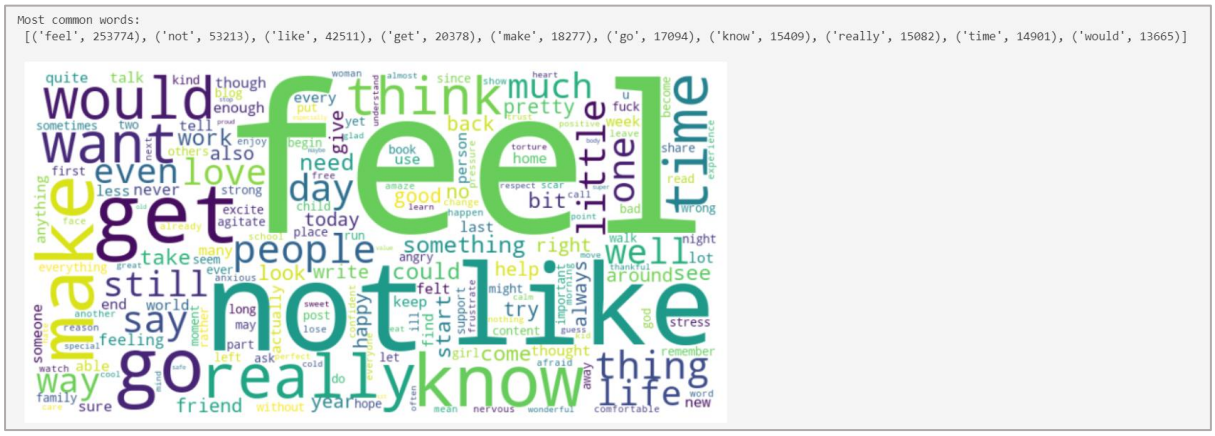


Figure 77: Word Frequency in Whole Emotion Dataset

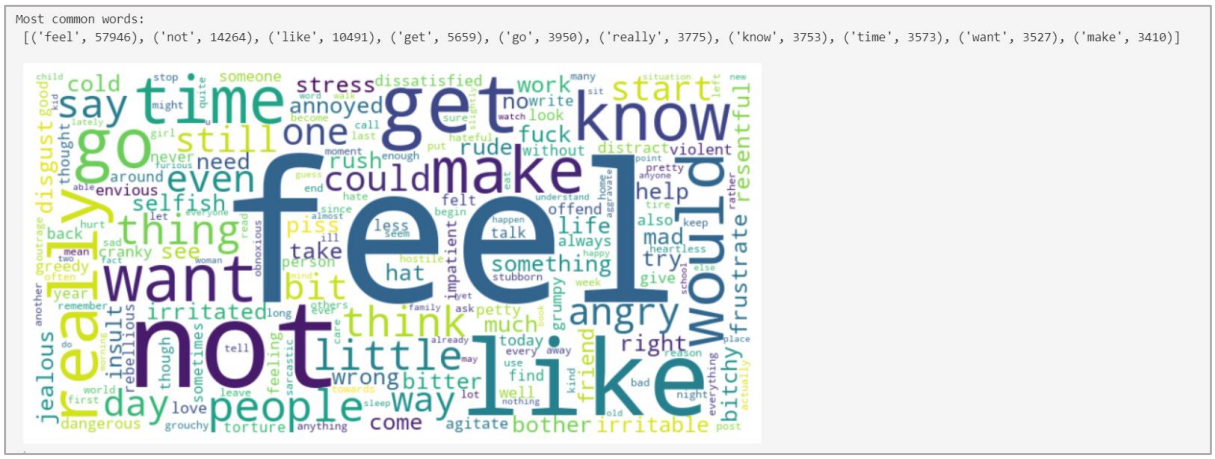


Figure 78: Word Frequency in Anger Emotion Data

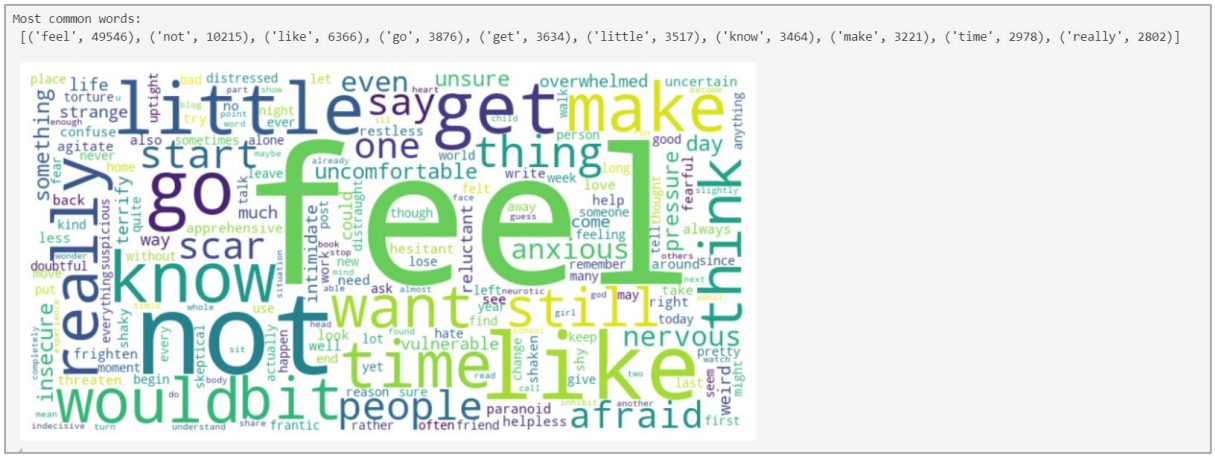


Figure 79: Word Frequency in Fear Emotion Data

These plots show that the most frequently occurring words in the emotion dataset are feel, not, like, get, regardless of the emotion category.

4.5.5 Unknown Word Visualization

Hate Speech Dataset

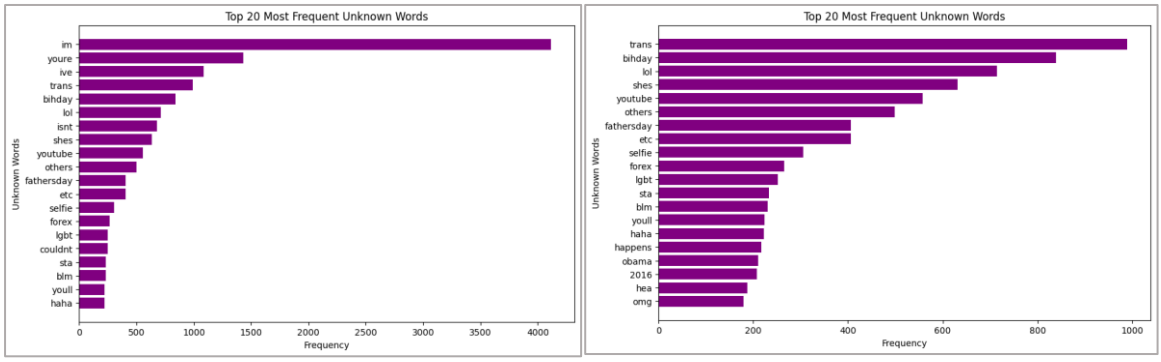


Figure 80: Unknown Words in Hate Speech Dataset

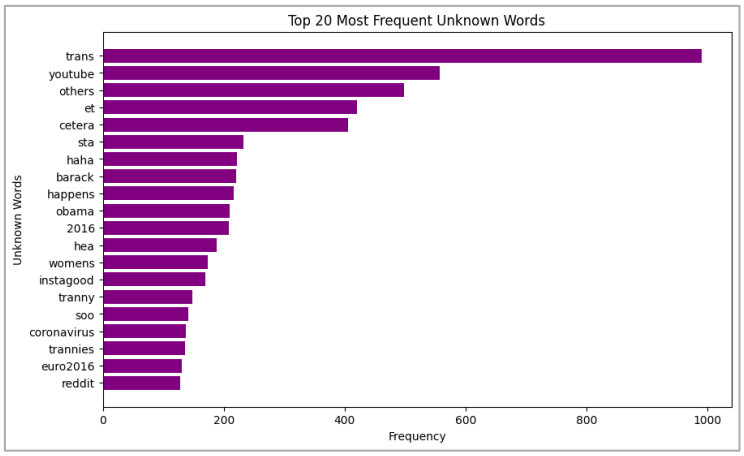


Figure 81: Unknown Words in Hate Speech Dataset (After Multiple Time of Text Preprocessing)

Using a combination of NLTK's WordNet and NLTK words dictionary to identify unknown words in the dataset. This figure highlights the presence of several unknown words in the first text preprocessing, specifically before removing punctuation, such as “im” (I am) and “youre” (you are) are frequent. In the future data preprocessing step, manually adding unknown words to a contractions and abbreviations list can enhance the model's understanding of the text.

Emotion Dataset

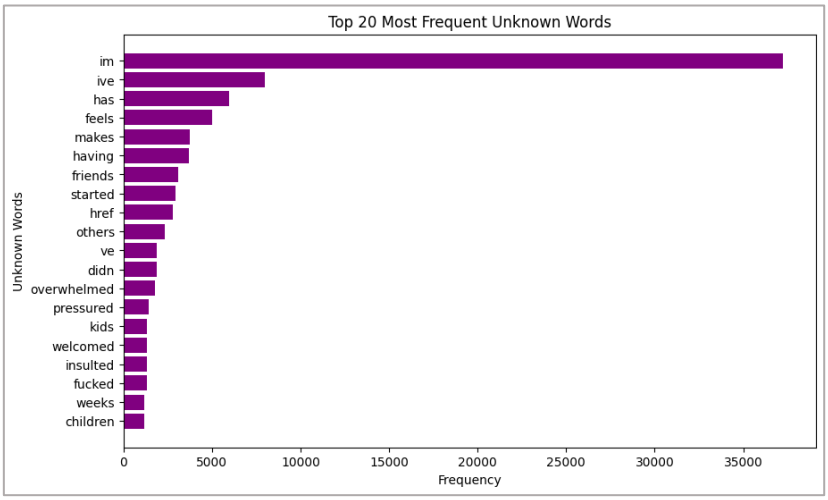


Figure 82: Top 20 Unknown Words in Emotion Dataset

This step helps identify commonly used contractions and abbreviations frequently appearing in online text, such as “im”, “ive”, as well as certain words like “has”, “feels”, and “makes” that are not included in NLTK's WordNet or NLTK's word dictionary because those are not root word forms. In the emotion dataset, those words appear over 5,000.

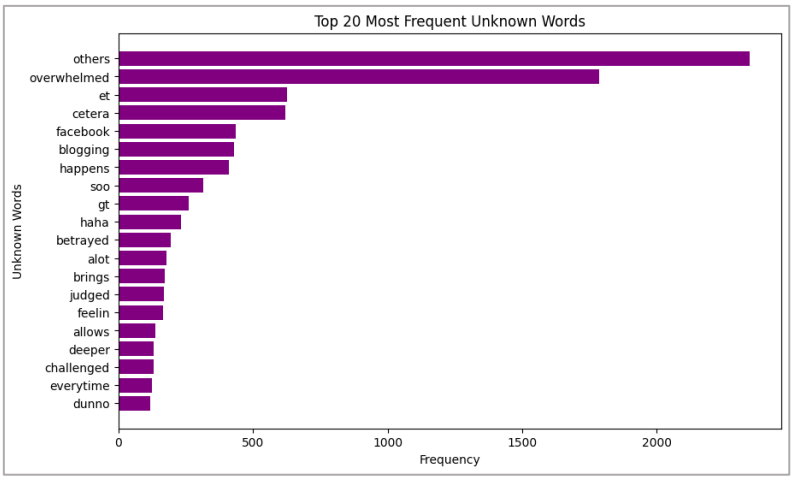


Figure 83: Unknown Words in Emotion Dataset (After Multiple Time of Text Preprocessing)

By replacing shortcut words with their full forms during text preprocessing, the model's understanding is enhanced, leading to more accurate text analysis.

4.5.6 N-grams Analysis

Hate Speech Dataset

Most Frequently Unigrams

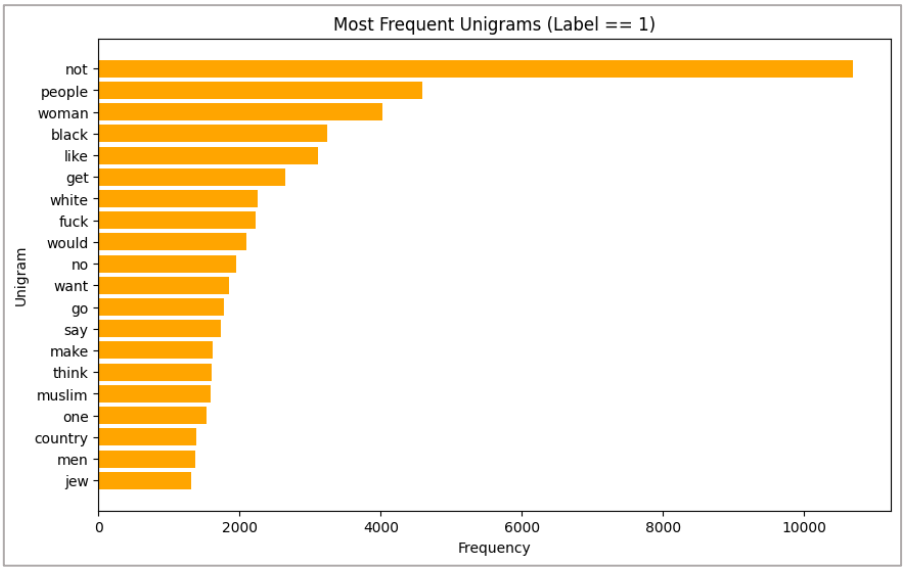


Figure 84: Horizontal Bar Chart - Most Frequently Unigrams in Hate Speech Dataset

Most Frequently Bigrams

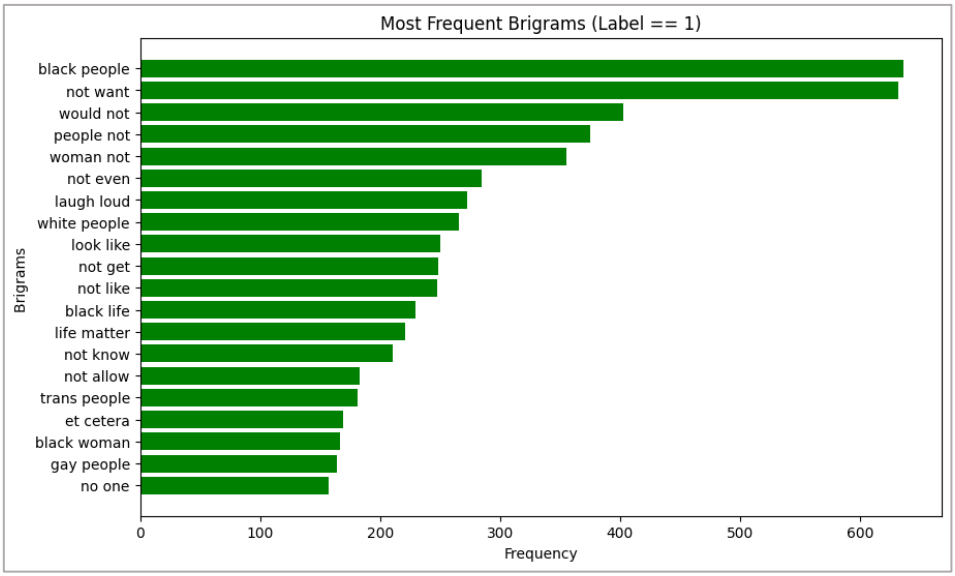


Figure 85: Horizontal Bar Chart - Most Frequently Bigrams in Hate Speech Dataset

Most Frequently Trigrams

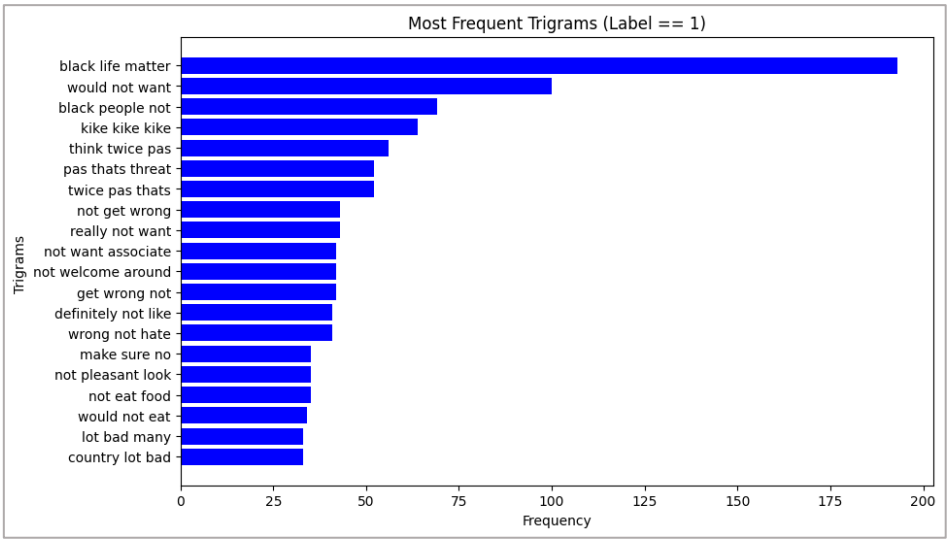


Figure 86: Horizontal Bar Chart - Most Frequently Trigrams in Hate Speech Dataset

Emotion Dataset

Most Frequently Bigrams

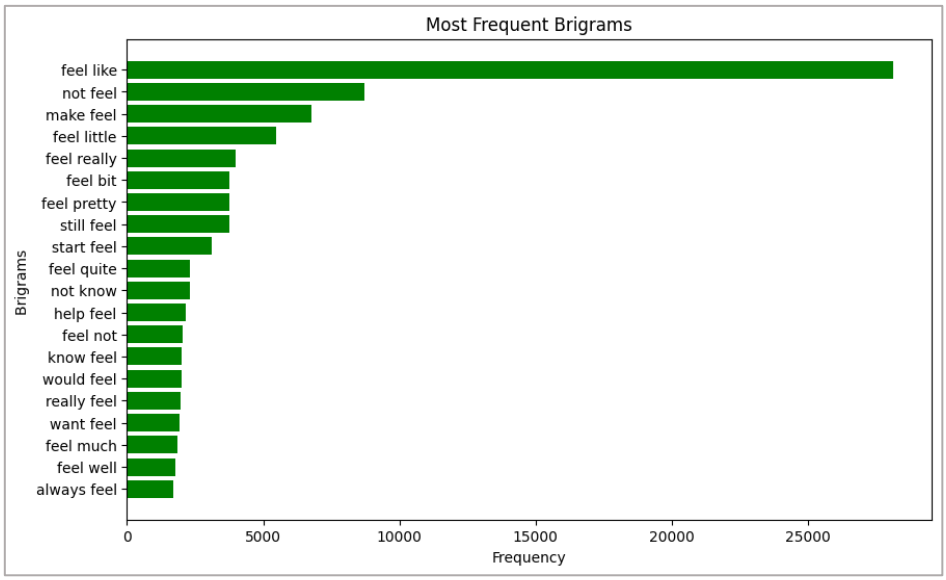


Figure 87: Horizontal Bar Chart - Most Frequently Bigrams in Emotion Dataset

Most Frequently Trigrams

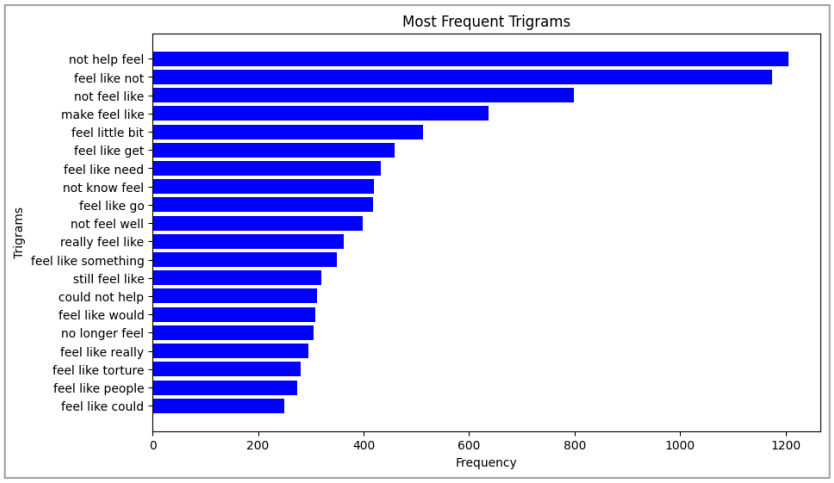


Figure 88: Most Frequently Trigrams in Emotion Dataset

The figures above illustrate the text n-grams identified as hate speech and emotion. N-gram can capture contextual information and semantics in word sequences, thereby improving language models, enhancing text prediction, and helping information retrieval. N-grams are commonly used in traditional statistical models, such as Naive Bayes and SVM. Also, in some deep learning models, such as RNN and CNN, especially in text classification, language modelling, information retrieval, sentiment analysis, and sequence labelling tasks.

4.5.7 POS Tagging Distribution

Hate Speech Dataset

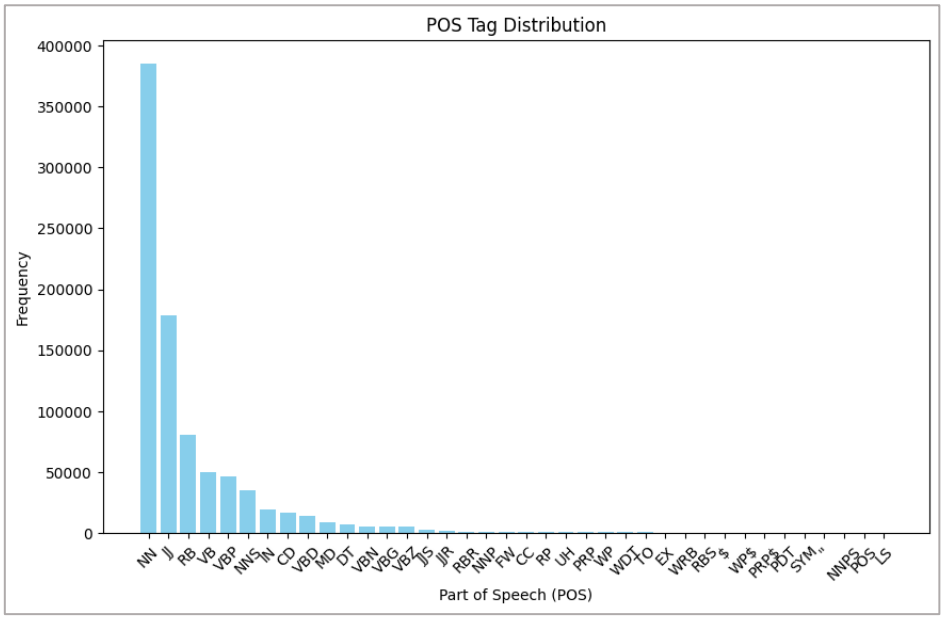


Figure 89: POS Tagging Distribution in Hate Speech Dataset

This bar chart represents the Part of Speech (POS) Tag Distribution within the hate speech dataset, showing the frequency of various POS tags. The most frequent tag is "NN" (noun), followed by “JJ” (adjective) and “RB” (Adverb) (Penn Treebank P.O.S. Tags, n.d.). Less frequent tags include “SYM” (Symbol) and “NNPS" (Proper noun, plural) (Penn Treebank P.O.S. Tags, n.d.).

Emotion Dataset

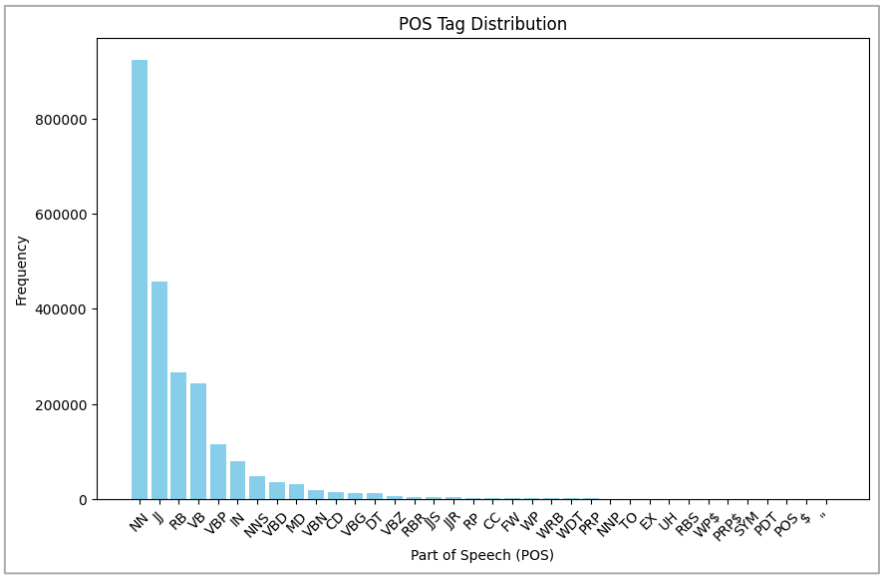


Figure 90: POS Tagging Distribution in Whole Emotion Dataset

This graph shows the distribution of parts of speech (POS) tags in the dataset. Nouns ("NN") are the most common, followed by adjectives ("JJ"), adverbs ("RB") and verbs ("VB"). POS tagging is important because it helps identify the role of each word in a sentence based on its context, making it easier to understand and analyse the text.

4.6 Hate Speech Model Building

4.6.1 Machine Learning Model

4.6.1.1 Import Library & Data Preparation

```
# Import library
import joblib
import pandas as pd
import numpy as np
from collections import Counter

# Feature Engineering
from sklearn.feature_extraction.text import TfidfVectorizer

# Data Preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from imblearn.under_sampling import RandomUnderSampler

# Model & Evaluation Metrics
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score, RandomizedSearchCV
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, f1_score, precision_score, recall_score
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.metrics import roc_curve, auc,roc_auc_score
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
```

Figure 91: Import Library - XGB

Various libraries are imported for different stages to build a text classification model, including data preprocessing, feature extraction, model training, and evaluation.

```
# Load File ----------------------------------------------------------------------------------
file_path = r"..\Pre_Hate_Dataset\20_Hate_Final.csv"
df = pd.read_csv(file_path)

# TF-IDF Vectorization  (Feature Engineering) ------------------------------------------------
tfidf = TfidfVectorizer(max_features=10000)
textData = tfidf.fit_transform(df['text'])

x = textData
y = df['label']
label_encoder = LabelEncoder() # Encode target labels into numeric values
y = label_encoder.fit_transform(y)

# Train Test Split ---------------------------------------------------------------------------
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
print("Train distribution:", Counter(y_train))
print("Validation distribution:", Counter(y_test))

# Resampling ---------------------------------------------------------------------------------
print("Class distribution before resampling:", Counter(y_train))
undersampler = RandomUnderSampler(random_state=42)

x_train_resampled, y_train_resampled = undersampler.fit_resample(x_train, y_train)
print("Class distribution after resampling:", Counter(y_train_resampled))
```

Figure 92: Data Preparation - XGB

```
Train distribution: Counter({0: 39404, 1: 18043})
Validation distribution: Counter({0: 16860, 1: 7761})
Class distribution before resampling: Counter({0: 39404, 1: 18043})
Class distribution after resampling: Counter({0: 18043, 1: 18043})
```

Figure 93: Output of Data Preparation

Table 8: Information of Data Preparation

Train Test Split	Resampling Method	Feature Engineering
Train Size: 70% Validation Size: 30%	• Undersampling After undersampling size: Non-hate (0): 18043 rows Hate (1): 18043 rows	• TfidfVectorizer (TF-IDF) Max length of feature: 100,000

The XGBoost machine learning model applies an undersampling method to balance the hate label. During model building, SMOTE oversampling and class weight adjustment were also tested to resample the hate label. However, the final results showed that undersampling provides the best performance.

4.6.1.2 XGBoost - Extreme Gradient Boosting

4.6.1.2.1 XGBoost Hyperparameter Explanation

Table 9: XGB Hyperparameter Explanation

Hyperparameter	Explanation
reg_lambda (L2 regularization)	Increasing this value makes the model more conservative, helping to prevent or reduce overfitting.
reg_alpha (L1 regularization)	L1 regularization drives the weights of unimportant features closer to zero to reduce their influence on the model
max_depth	Increasing this value enhances model complexity. A value of 0 imposes no depth limit. It helps control overfitting, as greater depth can lead to excessive complexity. This parameter is typically fine-tuned using cross-validation.
min_child_weight	If a tree partition results in the sum of instance weights of a leaf node being less than “min_child_weight”, stop further splitting. Setting this value too high may limit the growth of the tree, resulting in underfitting.

learning_rate	Controls the step size of weight updates during training. A high learning rate may cause the model to converge quickly but risk overshooting the optimal solution, while a low learning rate ensures more stable convergence but may require more training time.
gamma	Gamma sets the threshold for splitting the decision tree. A split will only occur if it reduces the loss by at least gamma. The larger the value, the more conservative the model is and the fewer splits the tree will have. This can effectively prevent overfitting.
subsample	Controls the proportion of random sampling for each tree. By reducing the value of this parameter, the algorithm will be more conservative and avoid overfitting. However, if this value is set too small, it may lead to underfitting.
colsample_bytree	"colsample_bytree" selects a part of the features for training each time to prevent over-reliance on a certain feature. The trained model will be smarter and more flexible.
n_estimators	"n_estimators" controls the number of trees. Tell the model how many trees can be divided into at most.

4.6.1.2.2 XGBoost Base Model

```
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix, f1_score, precision_score, recall_score, roc_curve, auc, accuracy_score
from sklearn.model_selection import cross_val_score
import joblib

xgb_default = XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    random_state=42
)

xgb_default.fit(x_train_resampled, y_train_resampled)

# Save Model
joblib.dump(xgb_default, "XGB_Default_Model.pkl")
print("Training completed and model saved.")
```

Figure 94: XGB Base Model Code

This XGBoost model is built using the default hyperparameters from the XGBoost documentation (XGBoost Parameters — XGBoost 1.5.2 Documentation, 2022).

Table 10: XGB Default Value of Hyperparameter

Hyperparameter	Default Value
reg_lambda (L2 regularization)	1
reg_alpha (L1 regularization)	0
max_depth	6
min_child_weight	1
learning_rate	0.3
gamma	0
subsample	1
colsample_bytree	1
n_estimators	100

4.6.1.2.3 XGBoost Hyperparameter Tuning

```
# HP base on: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
param_grid = {
    'reg_lambda': [5, 10, 15, 20],
    'reg_alpha': [3, 5, 10],
    'n_estimators': [50, 100, 150, 200],
    'max_depth': [8, 10, 15, 20],
    'learning_rate': [0.1, 0.2, 0.3],
    'gamma': [0, 0.1, 0.2, 0.3],
    'subsample': [0.5, 0.7, 0.8, 1.0],
    'colsample_bytree': [0.5, 0.7, 0.8, 1.0],
    'min_child_weight': [1, 3, 5, 7]
}

xgb = XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
)

XGBoost_random_search = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=param_grid,
    scoring='f1_weighted',
    cv=5,
    verbose=True,
    random_state=42,
    n_iter=10,
)
```

Figure 95: XGB Hyperparameter Tuning Code

The model is to do binary classification using “binary:logistic”. Each sample will output a probability between 0 and 1, and “0.5” will be used as the threshold to determine whether it is “0” represents non-hate or “1” is hate. Use “logloss” to calculate the error between the model prediction value and the true value. The smaller the “logloss”, the better the model effect. For hyperparameter tuning, “n_iter” setting as 10 and cross validation split into 5-fold, indicates that the model will randomly select 10 different hyperparameter combinations for training.

Each combination with 5-fold cross-validation results in a total of 50 training and evaluation runs.

The hyperparameter tuning values are based on (Analytics Vidhya, 2025) and refined through extensive tuning experiments. The goal of hyperparameter tuning is to achieve the highest accuracy. However, prioritizing accuracy can lead to overfitting. Through adjustments, the most suitable hyperparameter values were selected to balance performance and generalization.

Table 11: XGB Hyperparameter Tuning Values

- Hyperparameter Tuning Technique: scikit-learn "**RandomizedSearchCV"**

Hyperparameter	Value Setting
reg_lambda (L2 regularization)	5,10,15,20
reg_alpha (L1 regularization)	3,5,15
max_depth	8, 10, 15, 20
min_child_weight	1, 3, 5, 7
learning_rate	0.1, 0.2, 0.3
gamma	0, 0.1, 0.2, 0.3
subsample	0.5, 0.7, 0.8, 1
colsample_bytree	0.5, 0.7, 0.8, 1.0
n_estimators	50, 100, 150, 200

4.6.1.2.4 XGBoost Best Hyperparameter

```
================================================================================================================
✅ RandomizedSearchCV Object Type: <class 'sklearn.model_selection._search.RandomizedSearchCV'>
================================================================================================================
Best Parameters Found:
   subsample           : 0.8
   reg_lambda          : 10
   reg_alpha           : 5
   n_estimators        : 200
   min_child_weight    : 3
   max_depth           : 20
   learning_rate       : 0.3
   gamma               : 0.1
   colsample_bytree    : 0.7

⏳ Best Score (Weighted F1): 0.7510
```

Figure 96: XGB Best Hyperparameter

This figure presents the best hyperparameters obtained using RandomizedSearchCV. The optimal settings include subsample rate of 0.8, L2 regularization (reg_lambda) of 10, L1

regularization (reg_alpha) of 5, number of estimators (n_estimators) set to 200, minimum child weight (min_child_weight) of 3, maximum tree depth (max_depth) of 20, learning rate of 0.3, regularization parameter (gamma) of 0.1, and column subsampling ratio for trees (colsample_bytree) of 0.7.

4.6.1.2.5 XGBoost Fine Tuning

```
best_params = {
    'reg_lambda': 10,
    'reg_alpha': 1,
}

xgb_fine = XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    random_state=42,
    **best_params
)
eval_set = [(x_train_resampled, y_train_resampled), (x_test, y_test)]
xgb_fine.fit(x_train_resampled, y_train_resampled,
        eval_set=eval_set,
        verbose=True, )

# Save Model
joblib.dump(xgb_fine, "XGB_Fine_Model.pkl")
print("Training completed and model saved.")
```

Figure 97: XGB Fine Tuning Code

A comparison in section 5.2.2.3 Comparison Between XGBoost Hyperparameter Tuning and Base Model shows that hyperparameter tuning results in worse performance than the default XGBoost settings. While the base model exhibits slight overfitting, the hyperparameter-tuned model suffers from severe overfitting. Therefore, the base model will be chosen as the foundation for fine-tuning modifications, with the addition of “reg_lambda” = 10 and “reg_alpha” = 1.

4.6.2 Deep Learning Models

4.6.2.1 Import Library & Data Preparation

```
# Import Library
import numpy as np
import pandas as pd
import tensorflow as tf
import joblib
import pickle
from collections import Counter

# Data Preparation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from imblearn.under_sampling import RandomUnderSampler
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Deep Learning
from gensim.models import FastText, KeyedVectors
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.optimizers import Adam, SGD, Adagrad
from tensorflow.keras.callbacks import EarlyStopping

# Evaluation Metrics
from sklearn.metrics import classification_report, confusion_matrix, f1_score, precision_score, recall_score, roc_curve, auc, make_scorer,roc_auc_score
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.model_selection import KFold, cross_validate

# Hyperparameter Tuning
from keras_tuner import RandomSearch

# Model Performance
import seaborn as sns
import matplotlib.pyplot as plt
```

Figure 98: Import Library – DNN

Import those libraries for data preparation, model building, evaluation metrics, and model performance graph.

```
# Word Embedding
max_words = 20000  # vocab size
max_len = 100  # the max length of the each sentences

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
# Pad the sequence to ensure all sentences have the same length (max_len), truncating or padding as needed
X_padding = pad_sequences(X_seq, maxlen=max_len)

# Train Test Split
# 'stratify=Y' ensures the class distribution in Y is preserved in both sets
X_train, X_test, Y_train, Y_test = train_test_split(X_padding, Y, test_size=0.3, stratify=Y, random_state=42)
print("X_train shape", X_train.shape)
print("Y_train shape", Y_train.shape)
print("X_test shape", X_test.shape)
print("Y_test shape", Y_test.shape)

# Undersampling
print("\nClass distribution before resampling (total train and test):", Counter(Y))
undersampler = RandomUnderSampler(random_state=42)
x_train_resampled, y_train_resampled = undersampler.fit_resample(X_train,Y_train)

print("Class distribution after resampling (70% Train Set):", Counter(y_train_resampled))
print("Class distribution (30% Validation SET):", Counter(Y_test))

print(f"\nTotal number of training samples after resampling: {x_train_resampled.shape[0]}")
```

Figure 99: Data Preparation – DNN

```
X_train shape (57447, 100)
Y_train shape (57447,)
X_test shape (24621, 100)
Y_test shape (24621,)

Class distribution before resampling (total train and test): Counter({0: 56264, 1: 25804})
Class distribution after resampling (70% Train Set): Counter({0: 18063, 1: 18063})
Class distribution (30% Validation Set): Counter({0: 16880, 1: 7741})

Total number of training samples after resampling: 36126
```

Figure 100: Output of After Train Test Split

In the data preparation step, the labels of the target column are encoded as numeric values. The dataset is split into 70% for training and 30% for validation. Tokenization and word filling techniques are used to process the text data for model training. Due to the imbalanced dataset, undersampling method and "stratify=Y" are used to ensure that the proportion of each class in the dataset remains the same in the training and validation sets. Without stratification, the split may produce unbalanced validation and training sets, which will affect the model's performance. After resampling, there are 36126 rows of data for training the model.

4.6.2.2 Word Embedding Technique

Different word embedding methods impact the accuracy of a Bi-LSTM deep learning model. The following code loads pre-trained Word2Vec and FastText embeddings and creates an embedding matrix for use in a Keras Embedding layer. The process involves initializing a zero matrix, assigning pre-trained vectors to known words and randomly initializing vectors for out-of-vocabulary words. The embedding layer then uses this matrix with fixed weights to improve model performance.

Code of Word2Vec

```
MAX_VOCAB_SIZE = len(tokenizer.word_index) + 1

# Load Word2Vec
# https://www.kaggle.com/datasets/sugataghosh/google-word2vec
word2vec_path = "GoogleNews-vectors-negative300.bin"
word_vectors = KeyedVectors.load_word2vec_format(word2vec_path, binary=True)

embedding_matrix = np.zeros((MAX_VOCAB_SIZE, EMBEDDING_DIM))
for word, i in tokenizer.word_index.items():
    if i < MAX_VOCAB_SIZE:
        if word in word_vectors:     # Check if word exists in word_vectors
            embedding_matrix[i] = word_vectors[word]
        else:                        # Else: Out-of-Vocabulary (OOV) Words
            embedding_matrix[i] = np.random.normal(size=(EMBEDDING_DIM,))

embedding_layer = Embedding(
    input_dim=MAX_VOCAB_SIZE,
    output_dim=EMBEDDING_DIM,
    weights=[embedding_matrix],
    input_length=MAX_SEQUENCE_LENGTH,
    trainable=False
)
```

Figure 101: Word2Vec Code

Word2Vec is a word vector model proposed by Google. It is based on the large-scale Google News corpus (100 billion words). It uses two model architectures, Continuous Bag of Words (CBOW) and Skip-gram, to learn the relationship between words. This code loads GoogleNews-vectors-negative300.bin, which is a 300-dimensional pre-trained Word2Vec word vector model.

Code of FastText

```
MAX_VOCAB_SIZE = len(tokenizer.word_index) + 1
MAX_SEQUENCE_LENGTH = 100
EMBEDDING_DIM = 300

# https://fasttext.cc/docs/en/english-vectors.html
# Load FastText Model
fasttext_path = "crawl-300d-2M-subword.bin"
word_vectors = FastText.load_fasttext_format(fasttext_path)

# Initialize Embedding Matrix
embedding_matrix = np.zeros((MAX_VOCAB_SIZE, EMBEDDING_DIM))

for word, i in tokenizer.word_index.items():
    if i < MAX_VOCAB_SIZE:
        if word in word_vectors.wv:     # Check if word exists in FastText
            embedding_matrix[i] = word_vectors.wv[word]
        else:                           # Else Out-of-vocabulary (OOV)
            embedding_matrix[i] = np.random.normal(size=(EMBEDDING_DIM,))

# Create Embedding Layer
embedding_layer = Embedding(
    input_dim=MAX_VOCAB_SIZE,
    output_dim=EMBEDDING_DIM,
    weights=[embedding_matrix],
    input_length=MAX_SEQUENCE_LENGTH,
    trainable=False  # Keep embeddings fixed
)
```

Figure 102: FastText Code

FastText is a word embedding model proposed by Facebook. The model is trained on the large Common Crawl corpus and can effectively handle out-of-vocabulary (OOV) words and morphologically rich languages such as hate speech. FastText represents words as character n-grams, which can better learn rare or unseen words. This code loads crawl-300d-2M-subword.bin, which is a 300-dimensional pre-trained FastText word embedding model.

4.6.2.3 Bi-LSTM - Bidirectional Long Short-Term Memory Network

4.6.2.3.1 Bi-LSTM Hyperparameter Explanation

Table 12: Bi-LSTM Hyperparameter Explanation

Hyperparameter	Explanation
Units & Dense Units	Control model capacity more units can capture complex patterns but may increase risk of overfitting.
Optimizer	Determines how the model updates the weights when during training. The choice of optimizer affects both the speed of convergence and the quality of the final model. For example, Adam is commonly used for its adaptive learning rate and

	efficiency, while SGD with momentum can be useful for fine-tuning.
l2_reg (L2 regularization)	Helps prevent overfitting by adding a penalty to the loss function for large weights. This encourages the model to keep weights small, which can improve generalization to unseen data. L2 regularization is particularly useful in large size of the training data.
Activation	The activation function introduces non-linearity into the neural network, enabling it to learn and model complex patterns in the data. Common choices include ReLU, which is computationally efficient and helps mitigate the vanishing gradient problem and sigmoid or tanh functions, which are often used in specific scenarios, such as binary classification tasks.
Dropout Rate	Helps prevent overfitting by randomly dropping out a small percentage of neurons during training. This allows the model to learn features that are not dependent on any particular neurons.
Learning Rate	Controls the step size of weight updates during training. A high learning rate may cause the model to converge quickly but risk overshooting the optimal solution, while a low learning rate ensures more stable convergence but may require more training time.
Batch Size	Determines the number of samples processed before weight updates. Larger batch sizes provide stable gradients but require more memory.
Epochs	Determines the number of times the model will iterate over the entire training dataset. Too few epochs may result in underfitting, model has not learned enough from the data. Too many epochs may lead to overfitting, where the model memorizes the training data and performs poorly on unseen data. Techniques like early stopping can help identify the optimal number of epochs for training.

4.6.2.3.2 Bi-LSTM Base Model

```
MAX_SEQUENCE_LENGTH = 100
EMBEDDING_DIM = 300
MAX_VOCAB_SIZE = 200000

# https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM
model = Sequential([
    Embedding(input_dim=MAX_VOCAB_SIZE, output_dim=EMBEDDING_DIM, input_length=MAX_SEQUENCE_LENGTH),
    Bidirectional(LSTM(128, return_sequences=True)),  # by default: activation='tanh', recurrent_activation='sigmoid')
    Bidirectional(LSTM(64, return_sequences=False)),  # by default: activation='tanh', recurrent_activation='sigmoid',return_sequences=False)
    Dense(32, activation='tanh') ,
    Dense(1, activation='sigmoid')
])

model.compile(
    loss="binary_crossentropy",
    optimizer=Adam(),
    metrics=["accuracy"]
)

model.summary()

early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(
    x_train_resampled,
    y_train_resampled,
    epochs=10,
    batch_size=32,
    validation_data=(X_test, Y_test),
    callbacks=[early_stopping],
    verbose=1
)
```

Figure 103: Bi-LSTM Base Model

This Bi-LSTM model is built by default hyperparameters get from (Keras, n.d.).

Table 13: Bi-LSTM Hyperparameter Default Value

Hyperparameter	Default Value
Units	No default value. Common values include 256, 128, 64, 32, 16.
Activation	tanh
Recurrent Activation	sigmoid
Dropout	0
Recurrent Dropout	0
Adam default Learning Rate	0.001
Hyperparameter	**Selected Value**
Epochs	10
Batch Size	32
Optimizer	Adam
Loss Function	binary_crossentropy

4.6.2.3.3 Bi-LSTM Hyperparameter Tuning

```
def build_model(hp):
    model=Sequential()
    # Embedding Layer
    model.add(Embedding(input_dim=MAX_VOCAB_SIZE, output_dim=EMBEDDING_DIM, input_length=MAX_SEQUENCE_LENGTH))
    # 1st Hidden Layer
    model.add(Bidirectional(LSTM(units=hp.Int("LSTM_layer_1", min_value=64, max_value=256, step=64), return_sequences=True)))
    model.add(Dropout(hp.Float("dropout_1", min_value=0.2, max_value=0.4, step=0.1)))
    # 2nd Hidden Layer
    model.add(Bidirectional(LSTM(units=hp.Int("LSTM_layer_2", min_value=32, max_value=128, step=32), return_sequences=True )))
    model.add(Dropout(hp.Float("dropout_2", min_value=0.2, max_value=0.4, step=0.1)))
    # 3rd Hidden Layer
    model.add(Bidirectional(LSTM(units=hp.Int("LSTM_layer_3", min_value=16, max_value=64, step=32), return_sequences=False )))
    model.add(Dropout(hp.Float("dropout_3", min_value=0.2, max_value=0.4, step=0.1)))
    # 4th Dense Layer
    model.add(Dense(units=hp.Int("dense_units", min_value=16, max_value=64, step=16),
                    activation=hp.Choice("activation", ["relu", "tanh","sigmoid"]),
                    kernel_regularizer=tf.keras.regularizers.l2(hp.Float("l2_reg", min_value=0.001, max_value=0.1, step=0.01))))
    # Output Layer
    model.add(Dense(1, activation="sigmoid"))

    # Momentum = SGD + Mini-batch
    optimizer_choice = hp.Choice('optimizer', ['adam', 'sgd', 'adagrad'])
    learning_rate = hp.Choice("learning_rate", [0.0001, 0.003, 0.01, 0.3])

    if optimizer_choice == 'sgd':
        optimizer = SGD(learning_rate=learning_rate, momentum=0.9)  # (Jitendarpal Singn, 2018)
    elif optimizer_choice == 'adam':
        optimizer = Adam(learning_rate=learning_rate)
    elif optimizer_choice == 'adagrad':
        optimizer = Adagrad(learning_rate=learning_rate)

    model.compile(
        optimizer = optimizer, loss="binary_crossentropy", metrics=["accuracy"] )

    return model
```

```
tuner = RandomSearch(build_model,
                     objective='val_accuracy',
                     max_trials=10,
                     executions_per_trial=1,
                     directory='my_dir_LSTM_Hate',
                     project_name='hyperparam_tuning_LSTM_3_Initialized')

early_stopping = EarlyStopping(monitor="val_loss", patience=3, restore_best_weights=True)

tuner.search(x_train_resampled,y_train_resampled,epochs=10,batch_size=32,validation_data=(X_test,Y_test), callbacks=[early_stopping])

best_model = tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"""Best Hyperparameter:
        Bi-LSTM  Layer 1: {best_model.get('LSTM_layer_1')}
        Bi-LSTM  Layer 2: {best_model.get('LSTM_layer_2')}
        Bi-LSTM  Layer 3: {best_model.get('LSTM_layer_3')}
        Dropout 1: {best_model.get('dropout_1')}
        Dropout 2: {best_model.get('dropout_2')}
        Dropout 3: {best_model.get('dropout_3')}
        Dense : {best_model.get('dense_units')}
        Activation : {best_model.get('activation')}
        Learning Rate : {best_model.get('learning_rate')}
        Optimizer Choice: {best_model.get('optimizer')}
        """)

model = build_model(best_model)
history = model.fit(x_train_resampled, y_train_resampled, epochs=10, batch_size=32, validation_data=(X_test, Y_test))
```

Figure 104: Bi-LSTM Hyperparameter Tuning Code

Bi-LSTM using “RandomSearch” hyperparameter tuning techniques and Early Stop Function to monitor valid loss.

Table 14: Bi-LSTM Hyperparameter Setting

- Hyperparameter Tuning Technique: Keras Tuner "**RandomSearch"**

Hyperparameter	Range/Value
Bi-LSTM Units	16-256
Dropout	0.2-0.4
Dense Units	16-64
Activation	ReLU, Tanh, Sigmoid
Optimizer	Adam, SGD-Momentum, Adagrad
l2_reg (L2 regularization)	0.001-0.1
Learning Rate	0.0001, 0.003, 0.01, 0.3
Epochs	10
Batch Size	32

Justification of Hyperparameter Setting in Recurrent Neural Network (RNN)

Based on the deep learning book by (Goodfellow et al., 2016) (Chapter 11.4: Selecting Hyperparameters), the following common hyperparameter tuning values are selected for building the model. The Bi-LSTM units range from 32 to 256, allowing flexibility in model complexity. Depending on the specific use case, the model utilizes different activation functions, including ReLU, Tanh, and Sigmoid. The ReLU activation function should only be used in the hidden layers. Sigmoid, Logistic and Tanh functions are not suitable for use in hidden layers because those are more likely to suffer from the gradient vanishing problem during training (Baheti, 2022). Additionally, the learning rate values are determined based on prior extensive experimentation, as referenced in the materials provided by (R. RAJESWARI, personal communication, 2025).

Justification of Hyperparameter Tuning Technique Selected - "RandomSearch"

The hyperparameter tuning technique chooses to use Random Search. The main reason is that Random Search is more suitable for processing continuous and large range hyperparameters. Taking the Bi-LSTM model as an example, there are 7 hyperparameters. Grid Search needs to predefine a set of candidate values for each hyperparameter, and then exhaustively search all possible combinations. For example, there are 7 hyperparameters and each hyperparameter has 3 candidate values, Grid Search needs to try $3^7 = 2187$ sets of hyperparameter combinations, which will result in extremely high computational costs during training. The benefit of using

Random Search is that a certain number of combinations are randomly sampled within the hyperparameter range. For example, by setting "max_trials=10", the number of training times can be reduced. Avoid the computational complexity of Grid Search and can more efficiently explore the hyperparameter space in high-dimensional space, especially when the hyperparameter range is large and continuous. More information can refer to the research paper provide by (Bergstra & Bengio, 2012) mainly discusses hyperparameter optimization in neural networks and proves through experiments and theoretical analysis that random search is more efficient than grid search.

4.6.2.3.4 Bi-LSTM Best Hyperparameter

The table below presents different train-test split sizes, and different numbers of hidden layers and compares the use of various embedding layers, including FastText and Word2Vec.

Table 15: Bi-LSTM Hyperparameter Tuning - Best Hyperparameter

Tuning No.	Train-Test Split	Number of Hidden Layer	Embedding Layer	Best Hyperparameter
1	0.3	3	Randomly Initialized Trainable Embedding	Best Hyperparameters: ---------- Bi-LSTM_layer_1 : 192 dropout_1 : 0.2 Bi-LSTM_layer_2 : 64 dropout_2 : 0.2 dense_units : 16 activation : relu l2_reg : 0.07100000000000001 optimizer : adam learning_rate : 0.003 ----------
2	**0.2**	**4**	Randomly Initialized Trainable Embedding	Best Hyperparameters: ---------- Bi-LSTM_layer_1 : 192 dropout_1 : 0.2 Bi-LSTM_layer_2 : 96 dropout_2 : 0.2 Bi-LSTM_layer_3 : 48 dropout_3 : 0.30000000000000004 dense_units : 16 activation : sigmoid l2_reg : 0.031 optimizer : adagrad learning_rate : 0.3 ----------

3	**0.3**	4	Randomly Initialized Trainable Embedding	Best Hyperparameters: ---------- Bi-LSTM_layer_1 : 256 dropout_1 : 0.2 Bi-LSTM_layer_2 : 96 dropout_2 : 0.4 Bi-LSTM_layer_3 : 48 dropout_3 : 0.30000000000000004 dense_units : 32 activation : sigmoid l2_reg : 0.011 optimizer : adam learning_rate : 0.0001 ----------
4	0.3	4	Pre-trained **FastText** Embedding	Best Hyperparameters: ---------- Bi-LSTM_layer_1 : 64 dropout_1 : 0.4 Bi-LSTM_layer_2 : 96 dropout_2 : 0.4 Bi-LSTM_layer_3 : 48 dropout_3 : 0.2 dense_units : 48 activation : relu l2_reg : 0.041 optimizer : adam learning_rate : 0.003 ----------
5	0.3	4	Pre-trained **Word2Vec** Embedding	Best Hyperparameters: ---------- Bi-LSTM_layer_1 : 128 dropout_1 : 0.2 Bi-LSTM_layer_2 : 128 dropout_2 : 0.2 Bi-LSTM_layer_3 : 16 dropout_3 : 0.4 dense_units : 64 activation : tanh l2_reg : 0.051000000000000004 optimizer : adam learning_rate : 0.003 ----------

4.6.2.3.5 Bi-LSTM Fine Tuning with the Best Parameter

```
# ----------------------------------------------------------------------------------------------------------------------
MAX_VOCAB_SIZE = len(tokenizer.word_index) + 1
MAX_SEQUENCE_LENGTH = 100
EMBEDDING_DIM = 300
# https://fasttext.cc/docs/en/english-vectors.html
# Load FastText Model
fasttext_path = "crawl-300d-2M-subword.bin"
word_vectors = FastText.load_fasttext_format(fasttext_path)

# Initialize Embedding Matrix
embedding_matrix = np.zeros((MAX_VOCAB_SIZE, EMBEDDING_DIM))

for word, i in tokenizer.word_index.items():
    if i < MAX_VOCAB_SIZE:
        if word in word_vectors.wv:     # Check if word exists in FastText
            embedding_matrix[i] = word_vectors.wv[word]
        else:                           # Else Out-of-vocabulary (OOV)
            embedding_matrix[i] = np.random.normal(size=(EMBEDDING_DIM,))

# Create Embedding Layer
embedding_layer = Embedding(
    input_dim=MAX_VOCAB_SIZE,
    output_dim=EMBEDDING_DIM,
    weights=[embedding_matrix],
    input_length=MAX_SEQUENCE_LENGTH,
    trainable=False
)
```

```
model = Sequential([
    embedding_layer,
    Bidirectional(LSTM(64, return_sequences=True)),
    Dropout(0.2),
    Bidirectional(LSTM(96, return_sequences=True)),
    Dropout(0.2),
    Bidirectional(LSTM(48, return_sequences=False)), # 32
    Dropout(0.2),
    Dense(48, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.04)), # denses 16
    Dense(1, activation='sigmoid')
])

from tensorflow.keras.saving import register_keras_serializable
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras import backend as K

# When saving the model, Keras will automatically recognize my_custom_loss and load it correctly
@register_keras_serializable(package="CustomLoss")
# Function focal_loss code et from GitHub (aldi-dimara, 2018)
# Gamma, Alpha default values recommended by the paper (Lin et al., 2018) # https://arxiv.org/pdf/1708.02002v2
def focal_loss(y_true, y_pred, gamma=2.0, alpha=0.55): # 1.5 , 2.0 # 0.25,0.75
    epsilon = K.epsilon()
    y_pred = K.clip(y_pred, epsilon, 1.0-epsilon)
    # Predicted probability of the correct class
    pt = tf.where(K.equal(y_true, 1), y_pred, 1-y_pred)
    # Alpha factor for balancing class imbalance.
    alpha_factor = K.ones_like(y_true)*alpha
    # Dynamically assigns different alpha values to different classes
    alpha_t = tf.where(K.equal(y_true, 1), alpha_factor, 1-alpha_factor)
    # Cross-entropy loss
    cross_entropy = -K.log(pt)
    # Alpha > 0.5, then class 1 weigth > class 0 weigth. Alpha < 0.5, then class 0 weigth > class 1 weigth
    weight = alpha_t * K.pow((1-pt), gamma) #alpha_t*(1 - pt) ^ gamma
    loss = weight * cross_entropy
    # loss = K.sum(loss, axis=1)
    loss = K.mean(loss, axis=1)
    return loss
```

```
model.compile(
    loss=focal_loss,
    optimizer=Adam(learning_rate=0.003),
    metrics=["accuracy"]
)

# Reduce learning rate when loss has stopped reducing
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,           # Decrease LR (current LR * 0.5)
    patience=1,           # If val_loss does not improve for 1 epoch, reduce LR
    min_lr=0.0001,        # Setting minimum learning rate to prevent it from being too low
    verbose=1
)
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(
    x_train_resampled,
    y_train_resampled,
    epochs=15,
    batch_size=32,
    validation_data=(X_test, Y_test),
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)
```

Figure 105: Bi-LSTM Fine Tuning Code

According to the evaluation results of 5.2.3.2 Comparison Bi-LSTM Hyperparameter Tuning and Base Model, the best hyperparameters from the 4th hyperparameter architecture with Fastext are used for fine-tuning. In the fine-tuning process, all hyperparameters remained the same. Trying the adjustments included increasing the batch size from 32 to 64, reducing the third hidden layer units from 48 to 32, adjusting the learning rate 0.003 to 0.001 and dense layer size from 48 to 16. However, these changes did not improve the performance of the model. In the end, remain the best hyperparameters after the 4th tuning using the "ReduceLROnPlateau" function and manual Focal Loss function getting from GitHub (aldi-dimara, 2018). "ReduceLROnPlateau" function of is to monitor the validation loss of each epoch and reduce the learning rate when the loss stops decreasing according to the hyperparameter settings in "factor". The focal Loss function is a method that can let the model notice the misclassifications and increase the weight of the minority class by changing the gamma and alpha. Gamma = 2.0 and Alpha = 0.25 is the best value found (Lin et al., 2018). The table below shows showing Focal Loss function calculation with different alpha values (0.25 and 0.75).

Table 16: Focal Loss Function Calculation with Different Alpha Values (0.25 and 0.75).

```
# Predicted probability of the correct class
pt = tf.where(K.equal(y_true, 1), y_pred, 1-y_pred)
```

	Y pred	**PT Value (Predicted probability of the correct class)**
True Predict Class 1	0.7	0.7

True Predict Class 0	0.2	1-0.2=0.8
False Predict Class 1	0.4	0.4
False Predict Class 0	0.7	1-0.7=0.3

```
# Alpha factor for balancing class imbalance.
alpha_factor = K.ones_like(y_true)*alpha
# Dynamically assigns different alpha values to different classes
alpha_t = tf.where(K.equal(y_true, 1), alpha_factor, 1-alpha_factor)
```

Class 1 & Class 0	**Case 1: Alpha=0.25**	**Case 2: Alpha=0.75**
	"alpha_t" Value	**"alpha_t" Value**
Class 1	0.25	0.75
Class 0	1-0.25=0.75	1-075=0.25

```
# Alpha > 0.5, then class 1 weigth > class 0 weigth. Alpha < 0.5, then class 0 weigth > class 1 weigth
weight = alpha_t * K.pow((1-pt), gamma) #alpha_t*(1 - pt) ^ gamma
```

	Class Weight	**Class Weight**
True Predict Class 1	0.0225	0.0675
True Predict Class 0	0.03	0.01
False Predict Class 1	0.09	**0.27**
False Predict Class 0	**0.3675**	0.1225

```
# Cross-entropy loss
cross_entropy = -K.log(pt)
loss = weight * cross_entropy
loss = K.mean(loss, axis=1)
return loss
```

	Cross Entropy & Loss	**Cross Entropy & Loss**
True Predict Class 1	cross_entropy = 0.3567 Loss = 0.0080	cross_entropy = 0.3567 loss = 0.0241
True Predict Class 0	cross_entropy = 0.2231 loss = 0.0067	cross_entropy = 0.2231 loss = 0.0022
False Predict Class 1	cross_entropy = 0.9163 loss = 0.0825	cross_entropy = 0.9163 **loss = 0.2474**
False Predict Class 0	cross_entropy = 1.2040 **loss = 0.4425**	cross_entropy = 1.2040 loss = 0.1475

In this scenario, setting Alpha 0.25 would make the model pay more attention to the majority class (non-hate), which could lead to the model being biased toward the majority class and potentially performing worse on the minority class (hate). Therefore, it is better to use an Alpha value greater than 0.5 for my model.

4.6.2.4 Bi-GRU - Bidirectional Gated Recurrent Unit

4.6.2.4.1 Bi-GRU Hyperparameter Explanation

Table 17: Bi-GRU Hyperparameter Explanation

Hyperparameter	Explanation
Units & Dense Units	Control model capacity more units can capture complex patterns but may increase the risk of overfitting.
Optimizer	Determine how the model updates the weights during training. The choice of optimizer affects both the speed of convergence and the quality of the final model. For example, Adam is commonly used for its adaptive learning rate and efficiency, while SGD with momentum can be useful for fine-tuning.
l2_reg (L2 regularization)	Help to prevent overfitting by adding a penalty to the loss function for large weights. This encourages the model to keep weights small, which can improve generalization to unseen data. L2 regularization is particularly useful in the large size of the training data.
reset_after (reset gate & update gate)	Used to control the calculation method of reset gate and update gate. When reset_after=True, the GRU calculation method is compatible with CuDNN (a GPU library optimized for deep learning), which can improve the training speed on GPU.
Activation	The activation function introduces non-linearity into the neural network, enabling it to learn and model complex patterns in the data. Common choices include ReLU, which is computationally efficient and helps mitigate the vanishing gradient problem and sigmoid or tanh functions, which are often used in specific scenarios, such as binary classification tasks.

Dropout Rate	Helps prevent overfitting by randomly dropping out a small percentage of neurons during training. This allows the model to learn features that are not dependent on any particular neurons.
Learning Rate	Controls the step size of weight updates during training. A high learning rate may cause the model to converge quickly but risk overshooting the optimal solution, while a low learning rate ensures more stable convergence but may require more training time.
Batch Size	Determines the number of samples processed before weight updates. Larger batch sizes provide stable gradients but require more memory.
Epochs	Determines the number of times the model will iterate over the entire training dataset. Too few epochs may result in underfitting, model has not learned enough from the data. Too many epochs may lead to overfitting, where the model memorizes the training data and performs poorly on unseen data. Techniques like early stopping can help identify the optimal number of epochs for training.

4.6.2.4.2 Bi-GRU Base Model

```
MAX_SEQUENCE_LENGTH = 100
EMBEDDING_DIM = 300
MAX_VOCAB_SIZE = 200000

# https://www.tensorflow.org/api_docs/python/tf/keras/layers/GRU
model = Sequential([
    Embedding(input_dim=MAX_VOCAB_SIZE, output_dim=EMBEDDING_DIM, input_length=MAX_SEQUENCE_LENGTH),
    Bidirectional(GRU(128, return_sequences=True)),
    Bidirectional(GRU(64, return_sequences=False)),
    Dense(32, activation='tanh'),
    Dense(1, activation='sigmoid')
])

model.compile(
    loss="binary_crossentropy",
    optimizer=Adam(),  # Default Learning Rate: 0.001
    metrics=["accuracy"]
)

model.summary()

early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(
    x_train_resampled,
    y_train_resampled,
    epochs=10,
    batch_size=32,
    validation_data=(X_test, Y_test),
    callbacks=[early_stopping],
    verbose=1
)
```

Figure 106: Bi-GRU Base Model

This Bi-LSTM model is built with default hyperparameters obtained from (Cho, 2014).

Table 18: Bi-GRU Hyperparameter Default Value

Hyperparameter	Default Value
Units	No default value. Common values include 256, 128, 64, 32, 16.
Activation	tanh
Recurrent Activation	sigmoid
Dropout	0
Recurrent Dropout	0
Adam default Learning Rate	0.001
reset_after	True
Hyperparameter	**Selected Value**
Epochs	10
Batch Size	32
Optimizer	Adam
Loss Function	binary_crossentropy

4.6.2.4.3 Bi-GRU Hyperparameter Tuning

```
MAX_VOCAB_SIZE = len(tokenizer.word_index) + 1

# https://fasttext.cc/docs/en/english-vectors.html
fasttext_path = "crawl-300d-2M-subword.bin"
word_vectors = FastText.load_fasttext_format(fasttext_path)

# declare matrix
embedding_matrix = np.zeros((MAX_VOCAB_SIZE, EMBEDDING_DIM))

for word, i in tokenizer.word_index.items():
    if i < MAX_VOCAB_SIZE:
        if word in word_vectors.wv:
            embedding_matrix[i] = word_vectors.wv[word]
        else:
            embedding_matrix[i] = np.random.normal(size=(EMBEDDING_DIM,))

embedding_layer = Embedding(
    input_dim=MAX_VOCAB_SIZE,
    output_dim=EMBEDDING_DIM,
    weights=[embedding_matrix],
    input_length=MAX_SEQUENCE_LENGTH,
    trainable=False
)
```

```
def build_model(hp):
    model=Sequential()
    model.add(embedding_layer)

    model.add(Bidirectional(GRU(units=hp.Int("GRU_layer_1", min_value=64, max_value=256, step=64),return_sequences=True)))
    model.add(Dropout(hp.Float("dropout_1", min_value=0.2, max_value=0.4, step=0.1)))

    model.add(Bidirectional(GRU(units=hp.Int("GRU_layer_2", min_value=32, max_value=128, step=32), return_sequences=True )))
    model.add(Dropout(hp.Float("dropout_2", min_value=0.2, max_value=0.4, step=0.1)))

    model.add(Bidirectional(GRU(units=hp.Int("GRU_layer_3", min_value=16, max_value=64, step=32), return_sequences=False )))
    model.add(Dropout(hp.Float("dropout_3", min_value=0.2, max_value=0.5, step=0.1)))

    model.add(Dense(units=hp.Int("dense_units", min_value=16, max_value=64, step=16),
                    activation=hp.Choice("activation", ["relu", "tanh","sigmoid"]),
                    kernel_regularizer=tf.keras.regularizers.l2(hp.Float("l2_reg", min_value=0.001, max_value=0.1, step=0.01))))
    # output
    model.add(Dense(1, activation="sigmoid"))

    # Momentum = SGD + Mini-batch
    optimizer_choice = hp.Choice('optimizer', ['adam', 'sgd', 'adagrad'])
    learning_rate = hp.Choice("learning_rate", [0.0001, 0.003, 0.01, 0.3])

    if optimizer_choice == 'sgd':
        optimizer = SGD(learning_rate=learning_rate, momentum=0.9)  # use Momentum (!!! add cita : why is 0.9 default value)
    elif optimizer_choice == 'adam':
        optimizer = Adam(learning_rate=learning_rate)
    elif optimizer_choice == 'adagrad':
        optimizer = Adagrad(learning_rate=learning_rate)

    model.compile(
        optimizer = optimizer,
        loss="binary_crossentropy",
        metrics=["accuracy"]
    )
    return model
```

```
tuner = RandomSearch(build_model,
                     objective='val_accuracy',
                     max_trials=10,
                     executions_per_trial=1,
                     directory='my_dir_GRU_Hate',
                     project_name='hyperparam_tuning_LSTM_4_FastText')

early_stopping = EarlyStopping(monitor="val_loss", patience=3, restore_best_weights=True)

tuner.search(x_train_resampled,y_train_resampled,epochs=10,batch_size=32,validation_data=(X_test,Y_test), callbacks=[early_stopping])

best_model = tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"""Best Hyperparameter:
        Bi-GRU  Layer 1: {best_model.get('GRU_layer_1')}
        Bi-GRU  Layer 2: {best_model.get('GRU_layer_2')}
        Bi-GRU  Layer 3: {best_model.get('GRU_layer_3')}
        Dropout 1: {best_model.get('dropout_1')}
        Dropout 2: {best_model.get('dropout_2')}
        Dropout 3: {best_model.get('dropout_3')}
        Dense : {best_model.get('dense_units')}
        Learning Rate : {best_model.get('learning_rate')}
        Optimizer Choice: {best_model.get('optimizer')}
        """)

model = build_model(best_model)
history = model.fit(x_train_resampled, y_train_resampled, epochs=10, batch_size=32, validation_data=(X_test, Y_test))
```

Figure 107: Bi-GRU Hyperparameter Tuning Code

Compared to the results from Bi-LSTM hyperparameter tuning, FastText was chosen as the word embedding layer in the Bi-GRU model because using Word2Vec led to overfitting in the Bi-LSTM model. The hyperparameter tuning objective is to get the highest validation accuracy ("val_accuracy") by using the early stop function to monitor validation loss ("val_loss") to prevent overfitting.

Table 19: Bi-GRU Hyperparameter Setting

- Hyperparameter Tuning Technique: Keras Tuner "**RandomSearch"**

Hyperparameter	Range/Value
Bi-GRU Units	16-256
Dropout	0.2-0.5
Dense Units	16-64
Activation	ReLU, Tanh, Sigmoid
Optimizer	Adam, SGD-Momentum, Adagrad
l2_reg (L2 regularization)	0.001-0.1
Learning Rate	0.0001, 0.003, 0.01, 0.3
Epochs	10
Batch Size	32

Table 20: Reference Table for Hyperparameter Settings and Tuning Techniques – Bi-GRU

Tittle	Links Refer To
RNN Hyperparameter Setting Explanation	Explaining Hyperparameter Setting in RNN
Explanation of Hyperparameter Tuning Technique – "RandomSearch"	Explanation of Hyperparameter Tuning Technique Selection

4.6.2.4.4 Bi-GRU Best Hyperparameter

```
================================================================================================
✅ RandomSearch Object Type: <class 'keras_tuner.src.tuners.randomsearch.RandomSearch'>
================================================================================================

Best Hyperparameters:
---------------------------------------
GRU_layer_1          : 128
dropout_1            : 0.2
GRU_layer_2          : 32
dropout_2            : 0.4
GRU_layer_3          : 16
dropout_3            : 0.2
dense_units          : 16
activation           : tanh
l2_reg               : 0.07100000000000001
optimizer            : adam
learning_rate        : 0.003
---------------------------------------

⏳ Best Score (Weighted F1): 0.8115
```

Figure 108: Bi-GRU Hyperparameter Tuning - Best Hyperparameter

This figure presents the best hyperparameters obtained from Bi-GRU hyperparameter tuning. The first layer consists of 128 units with a dropout rate of 0.2, followed by a second layer with 32 units and a dropout rate of 0.4. The third layer has 16 units with a dropout rate of 0.2. The fully connected dense layer consists of 16 units, utilizing the tanh activation function and an L2 regularization of 0.07. The model is optimized using the Adam optimizer with a learning rate of 0.003.

4.6.2.4.5 Bi-GRU Fine Tuning with the Best Parameter

```
# https://fasttext.cc/docs/en/english-vectors.html
fasttext_path = "crawl-300d-2M-subword.bin"
word_vectors = FastText.load_fasttext_format(fasttext_path)
embedding_matrix = np.zeros((MAX_VOCAB_SIZE, EMBEDDING_DIM))

for word, i in tokenizer.word_index.items():
    if i < MAX_VOCAB_SIZE:
        if word in word_vectors.wv:
            embedding_matrix[i] = word_vectors.wv[word]
        else:
            embedding_matrix[i] = np.random.normal(size=(EMBEDDING_DIM,))

embedding_layer = Embedding(
    input_dim=MAX_VOCAB_SIZE,
    output_dim=EMBEDDING_DIM,
    weights=[embedding_matrix],
    input_length=MAX_SEQUENCE_LENGTH,
    trainable=False
)
```

```
def build_model():
    model = Sequential()
    model.add(embedding_layer)

    model.add(Bidirectional(GRU(units=128, return_sequences=True)))
    model.add(Dropout(0.2))

    model.add(Bidirectional(GRU(units=32, return_sequences=True)))
    model.add(Dropout(0.4))

    model.add(Bidirectional(GRU(units=16, return_sequences=False)))
    model.add(Dropout(0.2))

    model.add(Dense(units=16, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)))
    model.add(Dense(1, activation="sigmoid"))

    optimizer = Adam(learning_rate=0.003)

    @register_keras_serializable(package="CustomLoss")
    def focal_loss(gamma=2.0, alpha=0.53):
        def focal_loss_1(y_true, y_pred):
            epsilon = K.epsilon()
            y_pred = K.clip(y_pred, epsilon, 1.0-epsilon)
            pt = tf.where(K.equal(y_true, 1), y_pred, 1-y_pred) # Predicted probability of the correct class
            alpha_factor = K.ones_like(y_true)*alpha # Alpha factor for balancing class imbalance.
            alpha_t = tf.where(K.equal(y_true, 1), alpha_factor, 1-alpha_factor)  # Dynamically assigns different alpha values to different classes
            cross_entropy = -K.log(pt) # Cross-entropy loss
            # Alpha > 0.5, then class 1 weigth > class 0 weigth. Alpha < 0.5, then class 0 weigth > class 1 weigth
            weight = alpha_t * K.pow((1-pt), gamma) #alpha_t*(1 - pt) ^ gamma
            loss = weight * cross_entropy
            loss = K.mean(loss, axis=1)
            return loss
        return focal_loss_1

    model.compile(optimizer=optimizer, loss=focal_loss(), metrics=['accuracy'])
    return model
```

```
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,           # Decrease LR (current LR * 0.5)
    patience=1,           # If val_loss does not improve for 1 epoch, reduce LR
    min_lr=0.0001,        # Setting minimum learning rate to prevent it from being too low
    verbose=1
)
early_stopping = EarlyStopping(monitor="val_loss", patience=2, restore_best_weights=True)

model = build_model()
history = model.fit(x_train_resampled, y_train_resampled, epochs=10, batch_size=32,
                    validation_data=(X_test, Y_test), callbacks=[early_stopping,reduce_lr])
```

Figure 109: Bi-GRU Fine Tuning Code

According to Bi-GRU hyperparameter result Evaluation Result of Bi-GRU Hyperparameter Tuning, result of the learning curve is very unstable, so we add focal loss algorithm and use "ReduceLROnPlateau" function in the fine-tuning part. The focal loss algorithm is to strengthen the model's attention to minority categories, and the "ReduceLROnPlateau" function is to reduce the learning rate when a valid loss has stopped decreasing, making the model more stable without a curve with large fluctuations.

The hyperparameter settings are not changed using the best hyperparameter found through Bi-GRU hyperparameter tuning. In the focal loss parameter selection, gamma=0.2 and alpha=0.53 are used. The alpha parameter cannot be set too high, which will lead to a high false positive rate. The "ReduceLROnPlateau" function factor parameter is set as 0.5, and when the valid loss stops decreasing, the learning curve is automatically reduced by 50%.

4.6.2.5 CNN-Bi LSTM Hybrid Model

4.6.2.5.1 CNN-Bi LSTM Hyperparameter Explanation

For justification of Bi-LSTM hyperparameters, please refer to the Bi-LSTM Hyperparameter Explanation.

Table 21: CNN Hyperparameter Explanation

Hyperparameter	Explanation
Conv1D Layer Filters	Determines the number of feature maps output by the convolutional layer and controls the complexity of the model. The more filters there are, the richer the features that the model can learn, but the computational cost will also increase.
Kernel Size	Kernel size determines the number of words processed in each convolution operation. For example, kernel size equal to 3 means processing 3 words at a time.

Padding	Padding is used to preserve the output image size and reduce edge information loss. **Valid**: Without padding, the output feature map will become smaller. **Same**: Pad the input data so that the output feature map is the same size as the input
Activation	The activation function introduces non-linearity into the neural network, enabling it to learn and model complex patterns in the data. Common choices include ReLU, which is computationally efficient and helps mitigate the vanishing gradient problem and sigmoid or tanh functions, which are often used in specific scenarios, such as binary classification tasks.
Pool Size (MaxPooling1D Layer)	Pooling size is used to reduce the dimension of feature map and reduce the amount of calculation.

4.6.2.5.2 CNN-Bi LSTM Hyperparameter Tuning

```
MAX_VOCAB_SIZE = len(tokenizer.word_index) + 1

# https://fasttext.cc/docs/en/english-vectors.html
fasttext_path = "crawl-300d-2M-subword.bin"
word_vectors = FastText.load_fasttext_format(fasttext_path)

embedding_matrix = np.zeros((MAX_VOCAB_SIZE, EMBEDDING_DIM))

for word, i in tokenizer.word_index.items():
    if i < MAX_VOCAB_SIZE:
        if word in word_vectors.wv:
            embedding_matrix[i] = word_vectors.wv[word]
        else:
            embedding_matrix[i] = np.random.normal(size=(EMBEDDING_DIM,))

embedding_layer = Embedding(
    input_dim=MAX_VOCAB_SIZE,
    output_dim=EMBEDDING_DIM,
    weights=[embedding_matrix],
    input_length=MAX_SEQUENCE_LENGTH,
    trainable=False
)
```

```python
def build_model(hp):
    model = Sequential()
    model.add(embedding_layer)
    # The kernel size determines the number of words processed in each convolution operation
    # for example, kernel size=3 means processing 3 words at a time
    model.add(Conv1D(filters=hp.Int("conv_filters", min_value=32, max_value=128, step=32),
                     kernel_size=hp.Int("kernel_size", min_value=3, max_value=5, step=1),
                     activation='relu',
                     padding='same'))
    # Apply 1D max pooling to downsample the feature maps
    model.add(MaxPooling1D(pool_size=hp.Int("pool_size", min_value=2, max_value=4, step=1)))

    model.add(Bidirectional(LSTM(units=hp.Int("LSTM_layer_1", min_value=64, max_value=256, step=64),return_sequences=True)))
    model.add(Dropout(hp.Float("dropout_1", min_value=0.2, max_value=0.4, step=0.1)))

    model.add(Bidirectional(LSTM(units=hp.Int("LSTM_layer_2", min_value=32, max_value=128, step=32),return_sequences=True)))
    model.add(Dropout(hp.Float("dropout_2", min_value=0.2, max_value=0.4, step=0.1)))

    model.add(Bidirectional(LSTM(units=hp.Int("LSTM_layer_3", min_value=16, max_value=64, step=16),return_sequences=False)))
    model.add(Dropout(hp.Float("dropout_3", min_value=0.2, max_value=0.4, step=0.1)))

    model.add(Dense(units=hp.Int("dense_units", min_value=16, max_value=64, step=16),
                    activation=hp.Choice("activation", ["relu", "tanh","sigmoid"]),
                    kernel_regularizer=tf.keras.regularizers.l2(hp.Float("l2_reg", min_value=0.001, max_value=0.1, step=0.01))))

    model.add(Dense(1, activation="sigmoid"))

    optimizer_choice = hp.Choice('optimizer', ['adam', 'sgd', 'adagrad'])
    learning_rate = hp.Choice("learning_rate", [0.0001, 0.003, 0.01, 0.3])

    if optimizer_choice == 'sgd':
        optimizer = SGD(learning_rate=learning_rate, momentum=0.9)
    elif optimizer_choice == 'adam':
        optimizer = Adam(learning_rate=learning_rate)
    elif optimizer_choice == 'adagrad':
        optimizer = Adagrad(learning_rate=learning_rate)

    model.compile(
        optimizer = optimizer,loss="binary_crossentropy",metrics=["accuracy"]
    )
    return model
```

```python
tuner = RandomSearch(build_model,
                     objective='val_accuracy',
                     max_trials=10,
                     executions_per_trial=1,
                     directory='my_dir_CNN-LSTM_Hate',
                     project_name='hyperparam_tuning_CNN-LSTM_1_FastText')

early_stopping = EarlyStopping(monitor="val_loss", patience=3, restore_best_weights=True)

tuner.search(x_train_resampled,y_train_resampled,
             epochs=10,
             batch_size=32,
             validation_data=(X_test,Y_test),
             callbacks=[early_stopping])

best_model = tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"""Best Hyperparameter:
      CNN  conv_filters: {best_model.get('conv_filters')}
      CNN  kernel_size: {best_model.get('kernel_size')}
      CNN  pool_size: {best_model.get('pool_size')}
      Bi-LSTM  Layer 1: {best_model.get('LSTM_layer_1')}
      Bi-LSTM  Layer 2: {best_model.get('LSTM_layer_2')}
      Bi-LSTM  Layer 3: {best_model.get('LSTM_layer_3')}
      Dropout 1: {best_model.get('dropout_1')}
      Dropout 2: {best_model.get('dropout_2')}
      Dropout 3: {best_model.get('dropout_3')}
      Dense : {best_model.get('dense_units')}
      Activation : {best_model.get('activation')}
      Learning Rate : {best_model.get('learning_rate')}
      Optimizer Choice: {best_model.get('optimizer')}
      """)

model = build_model(best_model)
history = model.fit(x_train_resampled, y_train_resampled, epochs=10, batch_size=32, validation_data=(X_test, Y_test))
```

Figure 110: CNN-LSTM Hyperparameter Tuning Code

The main reason for using a hybrid model of FastText word embedding with CNN and BiLSTM is to combine the different advantages of the two models. CNN has better performance in extracting local features, such as n-grams in text While BiLSTM is good at capturing long-distance dependencies such as contextual information in text. CNN efficiently extracts local

features through convolution kernels but is weak in capturing long-distance dependencies. BiLSTM can handle global dependencies in text data but is less efficient than CNN in extracting local features. By combining the two, the hybrid model can capture local and global information at the same time.

Table 22: CNN-BiLSTM Hyperparameter Setting

- Hyperparameter Tuning Technique: Keras Tuner "**RandomSearch"**

Hyperparameter	Range/Value
Conv1D Filters	32-128
Kenel Size	3-5
Padding	same
Pool Size	2-4
Bi-LSTM Units	16-256
Dropout	0.2-0.4
Dense Units	16-64
Activation	ReLU, Tanh, Sigmoid
Optimizer	Adam, SGD-Momentum, Adagrad
l2_reg (L2 regularization)	0.001-0.1
Learning Rate	0.0001, 0.003, 0.01, 0.3
Epochs	10
Batch Size	32

Justification of Hyperparameter Setting in Convolutional Neural Network (CNN)

The CNN-BiLSTM model is a hybrid architecture that combines Convolutional Neural Networks (CNNs) and Bidirectional LSTMs (Bi-LSTMs), with the CNN component's hyperparameter tuning following the settings proposed by (Aszemi & Dominic, 2019). In this model, a 1D Convolutional layer is used as the first hidden layer to capture the patterns in the text data. Padding is set to "same" to ensure that the output sequence length matches the input length, preserving temporal information for subsequent layers. The second pooling layer helps reduce dimensionality and prevent overfitting by undersampling the extracted features. Since text data is inherently one-dimensional, there is no need for a Flatten layer before passing the features to the next stage. Lastly, fully connected Dense layers are added at the end of the model for classification output.

Table 23: Reference Table for Hyperparameter Settings and Tuning Techniques – CNN-BiLSTM

Title	Links Refer To
RNN Hyperparameter Setting Explanation	Explanation of Hyperparameter Setting in RNN
Explanation of Hyperparameter Tuning Technique – "RandomSearch"	Explanation of Hyperparameter Tuning Technique Selection

4.6.2.5.3 CNN-Bi LSTM Best Hyperparameter

```
================================================================================================
✅ RandomSearch Object Type: <class 'keras_tuner.src.tuners.randomsearch.RandomSearch'>
================================================================================================

Best Hyperparameters:
----------------------------------------
conv_filters         : 128
kernel_size          : 4
pool_size            : 3
LSTM_layer_1         : 128
dropout_1            : 0.30000000000000004
LSTM_layer_2         : 64
dropout_2            : 0.30000000000000004
LSTM_layer_3         : 48
dropout_3            : 0.30000000000000004
dense_units          : 16
activation           : relu
l2_reg               : 0.001
optimizer            : adagrad
learning_rate        : 0.01
----------------------------------------

⏳ Best Score (Weighted F1): 0.7793
```

Figure 111: CNN-Bi LSTM Hyperparameter Tuning - Best Hyperparameter

This figure presents the best hyperparameters through Random Search tuning for the model. The optimal hyperparameter includes 128 convolutional filters, kernel size is 4 and a pool size is 3 for feature extraction. The LSTM layers are followed by 128 units in the first layer, 64 units in the second layer and 48 units in the third layer, each hidden layer followed by dropout rate of 0.3 to prevent overfitting. The dense layer consists of 16 units with a ReLU activation function and L2 regularization of 0.001. The model is optimized using the Adagrad optimizer with a learning rate of 0.01.

4.6.2.5.4 CNN-Bi LSTM Fine Tuning with the Best Parameter

```
MAX_VOCAB_SIZE = len(tokenizer.word_index) + 1
MAX_SEQUENCE_LENGTH = 100
EMBEDDING_DIM = 300

# https://fasttext.cc/docs/en/english-vectors.html
fasttext_path = "crawl-300d-2M-subword.bin"
word_vectors = FastText.load_fasttext_format(fasttext_path)

embedding_matrix = np.zeros((MAX_VOCAB_SIZE, EMBEDDING_DIM))

for word, i in tokenizer.word_index.items():
    if i < MAX_VOCAB_SIZE:
        if word in word_vectors.wv:
            embedding_matrix[i] = word_vectors.wv[word]
        else:
            embedding_matrix[i] = np.random.normal(size=(EMBEDDING_DIM,))

embedding_layer = Embedding(
    input_dim=MAX_VOCAB_SIZE,
    output_dim=EMBEDDING_DIM,
    weights=[embedding_matrix],
    input_length=MAX_SEQUENCE_LENGTH,
    trainable=False
)
```

```
# ------------------------------------------------------------------------------------------------------------------------
optimizer = Adagrad(learning_rate=0.01)

def build_model():
    model = Sequential()
    model.add(embedding_layer)

    model.add(Conv1D(filters=128, kernel_size=4, activation="relu", padding="same"))
    model.add(MaxPooling1D(pool_size=3))

    model.add(Bidirectional(LSTM(units=128, return_sequences=True)))
    model.add(Dropout(0.3))
    model.add(Bidirectional(LSTM(units=64, return_sequences=True)))
    model.add(Dropout(0.3))
    model.add(Bidirectional(LSTM(units=48, return_sequences=False)))
    model.add(Dropout(0.3))

    model.add(Dense(units=16, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)))
    model.add(Dense(1, activation="sigmoid"))

    model.compile(optimizer=optimizer, loss="binary_crossentropy", metrics=['accuracy'])
    return model

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.8,           # Decrease LR (current LR * 0.5)
    patience=1,           # If val_loss does not improve for 1 epoch, reduce LR
    min_lr=0.0001,        # Setting minimum learning rate to prevent it from being too low
    verbose=1
)
```

Figure 112: CNN-Bi LSTM Fine Tuning Code

According to the CNN-Bi LSTM hyperparameter tuning result in Evaluation Result of CNN-Bi LSTM Hyperparameter Tuning, the model's learning curve showed fluctuation. In the fine-tuning model with adding the "ReduceLROnPlateau" function is used to monitor validation loss. When the validation loss did not decrease in one epoch, this function helps to adjust the learning rate according to "factor" value setting (0.8 is the best factor value for this model), 0.8 means maintain 80% of the previous learning rate value. This method can provide a model that has stabilized training and prevents overfitting. The early stopping function is used in the model

to monitor the model and stop the model in time when the model starts overfitting and restore the model to the state before overfitting. The layer units and other hyperparameter settings used the best value from hyperparameter tuning.

4.7 Hate Speech Integrate Emotion Concept-Level Fusion Model Building

4.7.1 Bi-LSTM Integrate "SenticNet" Concept-Level Sentiment

4.7.1.1 Import Library

```
import numpy as np
import pandas as pd
import tensorflow as tf
import joblib
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from imblearn.under_sampling import RandomUnderSampler
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from nltk.tokenize import word_tokenize

# SenticNet for Emotion Features
from senticnet.senticnet import SenticNet

# Deep Learning
from gensim.models import FastText
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense, Dropout, Bidirectional, Concatenate
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.regularizers import l2

# Evaluation Metrics
from sklearn.metrics import classification_report, confusion_matrix, f1_score, precision_score, recall_score, roc_curve, auc, make_scorer,roc_auc_score
from sklearn.metrics import precision_recall_curve, average_precision_score
```

Figure 113: Import Library

Those figures show that libraries are used in the data preparation, word embedding, model building and model evaluation. "SenticNet" is imported from the "SenticNet" Python library, which is a concept-based sentiment analysis resource used for emotion feature extraction.

4.7.1.2 Data Preparation and "SenticNet" Emotion Feature Extraction

```
# Load Data
file_path = r"..\Pre_Hate_Dataset\20_Hate_Final.csv"
data = pd.read_csv(file_path)
X = data['text'].astype(str).values
Y = data['label'].values
label_encoder = LabelEncoder()
Y = label_encoder.fit_transform(Y)

# Train test Split
X_train_orig, X_test_orig, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, stratify=Y, random_state=42)

max_words = 20000
max_len = 100

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_train_orig)

# Hate Text (text for get hate speech feature)
X_train_seq = tokenizer.texts_to_sequences(X_train_orig)
X_test_seq = tokenizer.texts_to_sequences(X_test_orig)

X_train_pad = pad_sequences(X_train_seq, maxlen=max_len)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)
```

Figure 114: Load Dataset

The data preparation for the feature-level fusion model differs from that of a single model. In this model, the first step of data preparation is to split the train set and validation set before applying tokenization, sequencing, and padding. Since “SenticNet” emotion feature extraction requires raw text input, not numerical sequences, while the word embedding layer typically requires sequencing and padding.

```
# Undersampling:
    # Since the two data forms are different, use indices undersampling method and apply it to X_train_pad and X_train_orig
undersampler = RandomUnderSampler(random_state=42)
    # Generate dummy 2D indices [[0], [1], [2], ...] because fit_resample() requires 2D input
    # np.arange(len(Y_train)) creates [0, 1, 2, ..., n-1] (indices of all samples)
    # RandomUnderSampler require input 2D shape(n_samples, 1).
    # reshape(-1, 1) converts 1D [1,2,3,4] to 2D shape [[1], [2], [3], [4], [5]]
dummy_indices = np.arange(len(Y_train)).reshape(-1, 1)
train_indices_resampled, Y_train_resampled = undersampler.fit_resample(dummy_indices, Y_train)
    # Apply the undersampled indices to X_train_pad (padded sequences) for hate speech feature extraction
    # .flatten() converts [[5], [89], [79], ...] to [5, 89, 79, ...]
X_train_resampled = X_train_pad[train_indices_resampled.flatten()]
# Apply the same indices to X_train_orig (original text) for SenticNet feature extraction
# SenticNet requires raw text input, not numerical sequences
X_train_senticnet_resampled = [X_train_orig[i] for i in train_indices_resampled.flatten()]
```

Figure 115: Undersampling Code

```
=== Word Embedding Representation ===
X_train_resampled shape: (36126, 100)
Y_train_resampled shape: (36126,)
X_test_pad shape: (24621, 100)
Y_test shape: (24621,)

=== SenticNet Representation ===
x_train_sentic shape: (36126, 100, 24)
Y_train_resampled shape: (36126,)
x_test_sentic shape: (24621, 100, 24)
Y_test shape: (24621,)
```

Figure 116: Output of Data Shape

This code shows a different undersampling method while ensuring compatibility of two different feature extraction word and emotion. Undersampling samples based on the index (dummy_indices). This allows to keep two different data formats, “X_train_pad” (the padding sequence used for hate speech feature extraction) and “X_train_orig” (the original text used for SenticNet sentiment feature extraction) at the same time, ensuring that the two features correspond correctly.

Use dummy indices and reshape the indices due to “fit_resample” function requiring a two-dimensional input. Use “train_indices_resampled” to select the corresponding data points from “X_train_pad” to get final “X_train_resampled” data use for training the model. The same indices, “dummy_indices”, are also applied to X_train_orig (original training set text form) to extract the corresponding raw text samples, ensuring compatibility with SenticNet, which requires textual input rather than tokenized numerical sequences. This method can

simultaneously consider the requirements of word embedding layers and the extraction of emotion features from SenticNet.

```python
# SenticNet Feature Extraction
def extract_sentic_features(text, tokenizer, max_length):
    sn = SenticNet()
    num_sentic_features = 24  # 24-dimensional features provided by SenticNet
    sentic_matrix = np.zeros((max_length, num_sentic_features), dtype=np.float32)

    tokens = word_tokenize(text)
    token_indices = tokenizer.texts_to_sequences([tokens])[0] # Convert to index (exp: [4, 10, 7])

    for i, token_idx in enumerate(token_indices[:max_length]):
        word = tokenizer.index_word.get(token_idx, '') # Reverse search for a word based on the index (exp: 4 → "I")
        try:
            # sn.concept(word): Query the sentiment features of word word in SenticNet
            sentic_values = [float(sn.concept(word)[j]) for j in range(num_sentic_features)]
            sentic_matrix[i] = sentic_values
        except KeyError:
            pass  # If SenticNet does not have the feature of this word, write 0
    return sentic_matrix

def batch_extract_sentic(texts, tokenizer, max_length):
    sentic_features = np.array([extract_sentic_features(text, tokenizer, max_length) for text in texts])
    return sentic_features.astype(np.float32)

x_train_sentic = batch_extract_sentic(X_train_senticnet_resampled, tokenizer, max_len)
x_test_sentic = batch_extract_sentic(X_test_orig, tokenizer, max_len)
```

Figure 117: "SenticNet" Emotion Feature Extraction

The batch_extract_sentic function for extracts "SenticNet" emotion features for both training and validation sets through "extract_sentic_features" function. The "extract_sentic_features" function processes tokenizing each input text into word and then queries the 24-dimensional sentiment features corresponding to the word in "SenticNet". If a word is not found in "SenticNet", the feature vector remains zero. Finally, the extracted features are stored as "x_train_sentic" and "x_test_sentic", corresponding to the training and validation sets.

4.7.1.3 BiLSTM-SenticNet Model

```python
# Build Model
MAX_VOCAB_SIZE = len(tokenizer.word_index) + 1
MAX_SEQUENCE_LENGTH = 100
EMBEDDING_DIM = 300

def build_hybrid_model(max_sequence_length):
    NUM_SENTIC_FEATURES = 24

    text_input = Input(shape=(max_sequence_length,), name='text_input')
    sentic_input = Input(shape=(max_sequence_length, NUM_SENTIC_FEATURES), name='sentic_input')

    Embedding_Layer = Embedding(input_dim=MAX_VOCAB_SIZE, output_dim=EMBEDDING_DIM, input_length=MAX_SEQUENCE_LENGTH)(text_input)

    merged_features = Concatenate()([Embedding_Layer, sentic_input])
    # merged_features = Concatenate()([text_input, sentic_input])

    x = Bidirectional(LSTM(64, return_sequences=True))(merged_features)
    x = Dropout(0.2)(x)
    x = Bidirectional(LSTM(96, return_sequences=True))(x)
    x = Dropout(0.2)(x)
    x = Bidirectional(LSTM(48, return_sequences=False))(x)
    x = Dropout(0.2)(x)
    x = Dense(48, activation='relu', kernel_regularizer=l2(0.001))(x)
    output = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=[text_input, sentic_input], outputs=output)
    return model

model = build_hybrid_model(MAX_SEQUENCE_LENGTH)
```

```
model.compile(
    loss="binary_crossentropy",
    optimizer=Adam(learning_rate=0.003),
    metrics=["accuracy"]
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,           # Decrease LR (current LR * 0.5)
    patience=1,           # If val_loss does not improve for 1 epoch, reduce LR
    min_lr=0.0001,        # Setting minimum learning rate to prevent it from being too low
    verbose=1
)
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(
    [X_train_resampled, x_train_sentic], Y_train_resampled,
    epochs=15, batch_size=32,
    validation_data=([X_test_pad, x_test_sentic], Y_test),
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)

model.save("1_BiLSTM_SenticNet_Model.keras")
joblib.dump(history.history, "1_BiLSTM_SenticNet_History.pkl")
print("Hybrid model saved successfully!")
```

Figure 118: BiLSTM-SenticNet Model Code

SenticNet is a concept-level sentiment analysis resource that provides emotion, mood, attitude, and cognitive-related features for words and multi-word expressions. Instead of just defining words like a dictionary, SenticNet assigns 24 dimensionals numerical vectors to words, representing different emotion related attributes (*SenticNet*, n.d.). By combining traditional text features with SenticNet's emotional features, the model can have a better understanding of emotional information in the text to experiment whether combining emotional features can help improve the model's performance in detecting hate speech.

This model integrates the concept-level sentiment analysis resource “SenticNet” into this model to form a feature-level fusion model. The Concatenate function is used to merge text-based features and sentiment-based features. The hyperparameters of the Bi-LSTM layer are based on the Bi-LSTM Fine Tuning with the Best Parameter , with 15 epochs and batch size of 32. A randomly initialized trainable embedding layer is created in this model, followed by a merged feature layer. The model includes three Bi-LSTM hidden layers with 64, 96, and 48 units, each layer accompanied by a dropout layer with a rate of 0.2. The last hidden layer is fully connected dense layer with 48 units. In addition, the model also includes a "ReduceLROnPlateau" function to monitor the "val_loss", which reduces the learning rate to prevent overfitting if the validation loss does not decrease.

4.7.1.4 BiLSTM-FastText-SenticNet Model

```
def build_hybrid_model(tokenizer, word_vectors, max_sequence_length=100):
    embedding_matrix = np.zeros((MAX_VOCAB_SIZE, EMBEDDING_DIM))
    for word, i in tokenizer.word_index.items():
        if i < MAX_VOCAB_SIZE:
            if word in word_vectors.wv:
                embedding_matrix[i] = word_vectors.wv[word]
            else:
                embedding_matrix[i] = np.random.normal(size=(EMBEDDING_DIM,))

    text_input = Input(shape=(max_sequence_length,), name='text_input')
    sentic_input = Input(shape=(max_sequence_length, NUM_SENTIC_FEATURES), name='sentic_input')

    embedding_layer = Embedding(
        input_dim=MAX_VOCAB_SIZE,
        output_dim=EMBEDDING_DIM,
        weights=[embedding_matrix],
        input_length=max_sequence_length,
        trainable=False,
        mask_zero=False
    )(text_input)
    merged_features = Concatenate()([embedding_layer, sentic_input])

    x = Bidirectional(LSTM(64, return_sequences=True))(merged_features)
    x = Dropout(0.2)(x)
    x = Bidirectional(LSTM(96, return_sequences=True))(x)
    x = Dropout(0.2)(x)
    x = Bidirectional(LSTM(48, return_sequences=False))(x)
    x = Dropout(0.2)(x)
    x = Dense(48, activation='relu', kernel_regularizer=l2(0.001))(x)
    output = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=[text_input, sentic_input], outputs=output)
    return model
```

```
fasttext_path = "crawl-300d-2M-subword.bin"
word_vectors = FastText.load_fasttext_format(fasttext_path)
model = build_hybrid_model(tokenizer, word_vectors, MAX_SEQUENCE_LENGTH)

model.compile(
    loss="binary_crossentropy",
    optimizer=Adam(learning_rate=0.003),
    metrics=["accuracy"]
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,           # Decrease LR (current LR * 0.5)
    patience=1,           # If val_loss does not improve for 1 epoch, reduce LR
    min_lr=0.0001,        # Setting minimum learning rate to prevent it from being too low
    verbose=1
)
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(
    [X_train_resampled, x_train_sentic],Y_train_resampled,
    epochs=15, batch_size=32,
    validation_data=([X_test_pad, x_test_sentic], Y_test),
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)
```

Figure 119: BiLSTM-FastText-SenticNet Model Code

This model is identical to the previous BiLSTM-SenticNet model, except that the word embedding layer has been replaced with FastText for text word embedding. The model is compiled using the binary cross-entropy loss function and the Adam optimizer with a learning rate of 0.003.

4.7.1.5 BiLSTM-FastText-Focal Loss-SenticNet Model

```
def build_hybrid_model(tokenizer, word_vectors, max_sequence_length):
    embedding_matrix = np.zeros((MAX_VOCAB_SIZE, EMBEDDING_DIM))
    for word, i in tokenizer.word_index.items():
        if i < MAX_VOCAB_SIZE:
            if word in word_vectors.wv:
                embedding_matrix[i] = word_vectors.wv[word]
            else:
                embedding_matrix[i] = np.random.normal(size=(EMBEDDING_DIM,))

    text_input = Input(shape=(max_sequence_length,), name='text_input')
    sentic_input = Input(shape=(max_sequence_length, NUM_SENTIC_FEATURES), name='sentic_input')

    embedding_layer = Embedding(
        input_dim=MAX_VOCAB_SIZE,
        output_dim=EMBEDDING_DIM,
        weights=[embedding_matrix],
        input_length=max_sequence_length,
        trainable=False,
        mask_zero=False
    )(text_input)

    merged_features = Concatenate()([embedding_layer, sentic_input])

    x = Bidirectional(LSTM(64, return_sequences=True))(merged_features)
    x = Dropout(0.2)(x)
    x = Bidirectional(LSTM(96, return_sequences=True))(x)
    x = Dropout(0.2)(x)
    x = Bidirectional(LSTM(48, return_sequences=False))(x)
    x = Dropout(0.2)(x)
    x = Dense(48, activation='relu', kernel_regularizer=l2(0.001))(x)
    output = Dense(1, activation='sigmoid')(x)
    model = Model(inputs=[text_input, sentic_input], outputs=output)
    return model

fasttext_path = "crawl-300d-2M-subword.bin"
word_vectors = FastText.load_fasttext_format(fasttext_path)
model = build_hybrid_model(tokenizer, word_vectors, MAX_SEQUENCE_LENGTH)
```

```
@register_keras_serializable(package="CustomLoss")
# Function focal_loss code et from GitHub (aldi-dimara, 2018)
def focal_loss(y_true, y_pred, gamma=2.0, alpha=0.55):
    epsilon = K.epsilon()
    y_pred = K.clip(y_pred, epsilon, 1.0-epsilon)
    pt = tf.where(K.equal(y_true, 1), y_pred, 1-y_pred)
    alpha_factor = K.ones_like(y_true)*alpha
    alpha_t = tf.where(K.equal(y_true, 1), alpha_factor, 1-alpha_factor)
    cross_entropy = -K.log(pt)
    weight = alpha_t * K.pow((1-pt), gamma)
    loss = weight * cross_entropy
    loss = K.mean(loss, axis=1)
    return loss

model.compile(
    loss=focal_loss,
    optimizer=Adam(learning_rate=0.003),
    metrics=["accuracy"]
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,           # Decrease LR (current LR * 0.5)
    patience=1,           # If val_loss does not improve for 1 epoch, reduce LR
    min_lr=0.0001,        # Setting minimum learning rate to prevent it from being too low
    verbose=1
)
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(
    [X_train_resampled, x_train_sentic], Y_train_resampled,
    epochs=15, batch_size=32,
    validation_data=([X_test_pad, x_test_sentic], Y_test),
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)
```

Figure 120: BiLSTM-FastText-Focal Loss-SenticNet Model Code

This model is based on the previous BiLSTM-FastText-SenticNet model, just changing the original loss function from binary cross-entropy to focal loss. Ordinary cross-entropy can cause

large gradient fluctuations in the early stages of training, especially for unbalanced datasets or models prone to overfitting. Focal loss helps alleviate the problem of gradient fluctuations, which reduces the weight of simple examples and puts more effort on samples that are difficult to classify.

4.7.2 Bi-LSTM Integrate Emotion Dataset

4.7.2.1 Import Library and Data Preparation

```
# Import Library
import numpy as np
import pandas as pd
import tensorflow as tf
import kerastuner as kt
import joblib
from collections import Counter

# Data Preparation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from imblearn.under_sampling import RandomUnderSampler
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.utils.class_weight import compute_class_weight

# Deep Learning
from tensorflow.keras import layers
from gensim.models import FastText, KeyedVectors
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense, Dropout, Bidirectional, concatenate
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from keras_tuner import RandomSearch,Objective

# Evaluation Metrics
from sklearn.metrics import classification_report, confusion_matrix, f1_score, precision_score, recall_score, roc_curve, auc,roc_auc_score
from sklearn.preprocessing import label_binarize

# Model Performance
import seaborn as sns
import matplotlib.pyplot as plt
```

Figure 121: Import Library

```
MAX_VOCAB_SIZE = 20000
MAX_SEQUENCE_LENGTH = 100
EMBEDDING_DIM = 300

file_path_hate = r"../Pre_Hate_Dataset/20_Hate_Final.csv"
file_path_emotion = r"../Pre_Emotion_Dataset/8_Lemmatization.csv"
df_hate = pd.read_csv(file_path_hate)
df_emotion = pd.read_csv(file_path_emotion)

hate_texts = df_hate['text'].astype(str).values
hate_labels = df_hate['label'].values
emo_texts = df_emotion['text'].astype(str).values
emo_labels = df_emotion['label'].values

# Encode labels
label_encoder_hate = LabelEncoder()
hate_labels = label_encoder_hate.fit_transform(hate_labels)

label_encoder_emotion = LabelEncoder()
emo_labels = label_encoder_emotion.fit_transform(emo_labels)

# Train-test split
hate_texts_train, hate_texts_test, hate_labels_train, hate_labels_test = train_test_split(hate_texts, hate_labels, test_size=0.3, stratify=hate_labels, random_state=42)

emo_texts_train, emo_texts_test, emo_labels_train, emo_labels_test = train_test_split(emo_texts, emo_labels, test_size=0.3, stratify=emo_labels, random_state=42)
```

```
# Tokenization (Fit only on training data)
tokenizer_hate = Tokenizer(num_words=MAX_VOCAB_SIZE)
tokenizer_hate.fit_on_texts(hate_texts_train)

tokenizer_emo = Tokenizer(num_words=MAX_VOCAB_SIZE)
tokenizer_emo.fit_on_texts(emo_texts_train)

# Convert texts to sequences
hate_seq_train = tokenizer_hate.texts_to_sequences(hate_texts_train)
hate_seq_test = tokenizer_hate.texts_to_sequences(hate_texts_test)

emo_seq_train = tokenizer_emo.texts_to_sequences(emo_texts_train)
emo_seq_test = tokenizer_emo.texts_to_sequences(emo_texts_test)

# Padding
hate_padding_train = pad_sequences(hate_seq_train, maxlen=MAX_SEQUENCE_LENGTH)
hate_padding_test = pad_sequences(hate_seq_test, maxlen=MAX_SEQUENCE_LENGTH)

emo_padding_train = pad_sequences(emo_seq_train, maxlen=MAX_SEQUENCE_LENGTH)
emo_padding_test = pad_sequences(emo_seq_test, maxlen=MAX_SEQUENCE_LENGTH)

X_train_hate, X_test_hate, Y_train_hate, Y_test_hate = hate_padding_train, hate_padding_test, hate_labels_train, hate_labels_test
X_train_emo, X_test_emo, Y_train_emo, Y_test_emo = emo_padding_train, emo_padding_test, emo_labels_train, emo_labels_test
```

```
# Undersampling ------------------------------------------------------------------------------------------------------

# Train Set
undersampler_hate_train = RandomUnderSampler(random_state=42,sampling_strategy={0: 18063, 1: 18063})
undersampler_emo_train = RandomUnderSampler(random_state=42,sampling_strategy={0: 12042, 1: 12042, 2: 12042})
X_train_hate_resampled, Y_train_hate_resampled = undersampler_hate_train.fit_resample(X_train_hate, Y_train_hate)
X_train_emo_resampled, Y_train_emo_resampled = undersampler_emo_train.fit_resample(X_train_emo, Y_train_emo)

# Validation Set
rng = np.random.RandomState(42)
min_test_size = min(len(X_test_hate), len(X_test_emo))  # Find min distribution in both datasets

# Hate Validation set
# replace=False: Indicates that the same sample cannot be selected repeatedly during sampling
indices_hate = rng.choice(len(X_test_hate), min_test_size, replace=False)
X_test_hate_resample = X_test_hate[indices_hate]
Y_test_hate_resample = Y_test_hate[indices_hate]

# Emotion Validation set
indices_emo = rng.choice(len(X_test_emo), min_test_size, replace=False)
X_test_emo_resample = X_test_emo[indices_emo]
Y_test_emo_resample = Y_test_emo[indices_emo]
```

Figure 122: Data Preparation

```
📌 Before Resampling ----------------------------------------------------

Hate Train-test Split:
X_train_hate shape (57447, 100)
Y_train_hate shape (57447,)
X_test_hate shape (24621, 100)
Y_test_hate shape (24621,)

Training label distribution: Counter({0: 39384, 1: 18063})
Validation label distribution: Counter({0: 16880, 1: 7741})

Emotion Train-test Split:
X_train_emo shape (171966, 100)
Y_train_emo shape (171966,)
X_test_emo shape (73700, 100)
Y_test_emo shape (73700,)

Training label distribution: Counter({0: 98543, 1: 40060, 2: 33363})
Validation label distribution: Counter({0: 42233, 1: 17169, 2: 14298})

📌 After Resampling -----------------------------------------------------

Hate Train-test Split:
X_train_hate shape (36126, 100)
Y_train_hate shape (36126,)
X_test_hate shape (24621, 100)
Y_test_hate shape (24621,)

Training label distribution: Counter({0: 18063, 1: 18063})
Validation label distribution: Counter({0: 16880, 1: 7741})

Emotion Train-test Split:
X_train_emo shape (36126, 100)
Y_train_emo shape (36126,)
X_test_emo shape (24621, 100)
Y_test_emo shape (24621,)

Training label distribution: Counter({0: 12042, 1: 12042, 2: 12042})
Validation label distribution: Counter({0: 14158, 1: 5671, 2: 4792})
```

Figure 123: Output of Data Preparation

The first step in preparing data for the multitask model involves loading the hate speech and emotion datasets, encoding the labels for training and classification, tokenizing the words, and padding all input sequences to ensure a uniform length. Since the two datasets have different sizes, the multitask model requires both inputs to have the same train and validation shapes.

The number of labelled samples in the hate speech dataset is small compared to the sentiment dataset. When the training set and the validation set are divided at a ratio of 70%, the number of training samples labelled as “hate” in the hate speech dataset is 18,063. Since the sentiment dataset contains three labels (0, 1, and 2), based on the ratio comparison, the equivalent number of training samples for each category (18063*2/3) is equal to 12,042 rows per category.

For the validation set, random undersampling is performed according to the label distribution of the original data. The validation labels of each category are not forced to be balanced, but the number of samples in each category is reduced proportionally. In the validation set resampling process, the important thing is using the same random resampling function (“rng.choice”) instance to generate the indices for both the text samples (X_test_hate) and their corresponding labels (Y_test_hate). This ensures that each randomly selected text is correctly

matched with its original label. If using two separate "rng.choice" in text samples (X_test_hate) and corresponding labels (Y_test_hate), it could result in misaligned pairs, and the selected text samples may no longer correspond to their correct labels. This method not only retains the true distribution of the data but also ensures the consistency of the number of labels under the multi-task dual-input requirements.

4.7.2.2 Word Embedding Technique

```
# FastText
fasttext_path = "crawl-300d-2M-subword.bin"
fasttext_vectors = FastText.load_fasttext_format(fasttext_path)

embedding_matrix_fastText = np.zeros((MAX_VOCAB_SIZE_HATE, EMBEDDING_DIM))
for word, i in tokenizer_hate.word_index.items():
    if i < MAX_VOCAB_SIZE_HATE:
        if word in fasttext_vectors.wv:
            embedding_matrix_fastText[i] = fasttext_vectors.wv[word]
        else:
            embedding_matrix_fastText[i] = np.random.normal(size=(EMBEDDING_DIM,))

# Word2Vec
word2vec_path = "GoogleNews-vectors-negative300.bin"
word2vec_vectors = KeyedVectors.load_word2vec_format(word2vec_path, binary=True)

embedding_matrix_word2Vec = np.zeros((MAX_VOCAB_SIZE_EMO, EMBEDDING_DIM))
for word, i in tokenizer_emo.word_index.items():
    if i < MAX_VOCAB_SIZE_EMO:
        if word in word2vec_vectors:
            embedding_matrix_word2Vec[i] = word2vec_vectors[word]
        else:
            embedding_matrix_word2Vec[i] = np.random.normal(size=(EMBEDDING_DIM,))

hate_embedding = Embedding(
    input_dim=MAX_VOCAB_SIZE_HATE,
    output_dim=EMBEDDING_DIM,
    weights=[embedding_matrix_fastText],
    input_length=MAX_SEQUENCE_LENGTH,
    trainable=True
)(hate_input)

emo_embedding = Embedding(
    input_dim=MAX_VOCAB_SIZE_EMO,
    output_dim=EMBEDDING_DIM,
    weights=[embedding_matrix_word2Vec],
    input_length=MAX_SEQUENCE_LENGTH,
    trainable=True
)(emo_input)
```

Figure 124: Word Embedding Code

In the multi-task model, two different word embedding techniques were used. FastText was used for the hate speech dataset and Word2Vec was used for the sentiment dataset. Since hate speech data contains many out-of-vocabulary (OOV) words, using FastText is a better choice. FastText enables the model to generalize more effectively to rare or unseen words by representing words as sub-word n-grams.

4.7.2.3 Bi-LSTM Multi-Task Model

```
# hate speech Bi-LSTM
hate_lstm_1 = Bidirectional(LSTM(64, return_sequences=True))(hate_embedding)
hate_dropout_1 = Dropout(0.4)(hate_lstm_1)

hate_lstm_2 = Bidirectional(LSTM(96, return_sequences=True))(hate_dropout_1)
hate_dropout_2 = Dropout(0.4)(hate_lstm_2)

hate_lstm_3 = Bidirectional(LSTM(48, return_sequences=False))(hate_dropout_2)
hate_dropout_3 = Dropout(0.2)(hate_lstm_3)

# emotion Bi-LSTM
emo_lstm_1 = Bidirectional(LSTM(64, return_sequences=True))(emo_embedding)
emo_dropout_1 = Dropout(0.4)(emo_lstm_1)

emo_lstm_2 = Bidirectional(LSTM(96, return_sequences=True))(emo_dropout_1)
emo_dropout_2 = Dropout(0.4)(emo_lstm_2)

emo_lstm_3 = Bidirectional(LSTM(48, return_sequences=False))(emo_dropout_2)
emo_dropout_3 = Dropout(0.2)(emo_lstm_3)

# combined features
combined_features = concatenate([hate_dropout_3, emo_dropout_3])
shared_dense = Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.071))(combined_features)

# output layer
hate_output = Dense(1, activation='sigmoid', name="hate_output")(shared_dense)
emotion_output = Dense(3, activation='softmax', name="emotion_output")(shared_dense)
model = Model(inputs=[hate_input, emo_input], outputs=[hate_output, emotion_output])
```

```
early_stopping = EarlyStopping(monitor='val_hate_output_loss', mode='min',patience=3, restore_best_weights=True)

history = model.fit(
    x={
        "hate_input": X_train_hate_resampled,
        "emo_input":X_train_emo_resampled},
    y={
        "hate_output": Y_train_hate_resampled,
        "emotion_output": Y_train_emo_resampled
    },
    epochs=10,
    batch_size=32,
    validation_data=(
        {
            "hate_input": X_test_hate_resample,
            "emo_input": X_test_emo_resample
        },
        {
            "hate_output": Y_test_hate_resample,
            "emotion_output": Y_test_emo_resample
        }
    ),
    callbacks=[early_stopping],
    verbose=1
)
```

Figure 125: Multi-task Model Building

The multi-task model integrates the emotion dataset by initially training two Bi-LSTM models independently. First, using Bi-LSTM to train hate and emotion separately. After that, the features extracted from the two models are combined using a concatenation function. The merged features are passed to the dense layer and finally connected to the output layer for prediction. The hyperparameters of the model are selected based on the best performance configuration in the hate model (Bi-LSTM) with batch size 32, details written in Bi-LSTM Best Hyperparameter Tuning No 4. In addition, to prevent overfitting and improve training efficiency, the model adopts an early stopping strategy with a patience time of 3 epochs.

This model will train two different validation sets, one is the imbalance validation set and the other is the balanced validation set. In addition, no hyperparameter adjustment was performed for this model because the results of this model were very unsatisfactory, which is explained more detail in Evaluation of Multi-task Model. In addition, to solve the problem of gradient conflict, loss-weight and attention layer were included, but the results were the same as the model's performance in the imbalanced validation set.

4.8 Summary

During the data cleaning phase observed that many internets text have a lot of abbreviations, resulting in many meaningless or unrecognizable words. This poses a challenge to word embedding during model building. To address this issue, common abbreviations (such as "im", "you're" and "asap") were replaced based on unknown word bar chart. Fluctuation in model performance during model building is one of the main challenges, which may be caused by data imbalance or noise in the dataset. In addition, another key challenge in building a multi-task learning (MTL) model is to ensure that the dimensions of the two datasets match, which is crucial for training a unified model.

CHAPTER 5: RESULTS AND DISCUSSION

5.1 Introduction

This chapter presents the results of six different models, each model including basic models, hyperparameter tuning and fine-tuning. This chapter provides a comparison of the six models' performance and analyses the challenges and limitations faced by each model. The top three models are listed by comparison. Last, this chapter also shows the user interface for model deployment, as well as test results in two specific scenarios (implicit hate speech and non-hate content containing hate keywords).

5.2 Model Evaluations and Discussions

5.2.1 Model Evaluation Code

```
train_loss, train_accuracy = model.evaluate(x_train_resampled, y_train_resampled, verbose=1)
val_loss, val_accuracy = model.evaluate(X_test, Y_test, verbose=1)

y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

print(f"\n📌 Train Accuracy: {train_accuracy:.4f}")
print(f"📌 Validation Accuracy: {val_accuracy:.4f}")

f1_weighted = f1_score(Y_test, y_pred, average='weighted')
print(f"\n📌 Weighted F1 Score: {f1_weighted:.4f}")

f1_micro = f1_score(Y_test, y_pred, average='micro')
print(f"📌 Micro F1 Score: {f1_micro:.4f}")

f1_macro = f1_score(Y_test, y_pred, average='macro')
print(f"📌 Macro F1 Score: {f1_macro:.4f}")

precision = precision_score(Y_test, y_pred, average='binary')
print(f"\n📌 Precision: {precision:.4f}")

recall = recall_score(Y_test, y_pred, average='binary')
print(f"📌 Recall: {recall:.4f}")

fpr, tpr, thresholds = roc_curve(Y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)
print(f"\n📌Test ROC AUC: {roc_auc:.4f}")

print("\n📌 Classification Report:")
print(classification_report(Y_test, y_pred))

print("\n📌 Confusion Matrix:")
print(confusion_matrix(Y_test, y_pred))
```

Figure 126: Evaluation Matrix of Deep Learning Classification Models

Those models were evaluated using F1 Score, Binary Precision, Binary Recall, ROC-AUC, Train Accuracy, and Validation Accuracy. If the precision is high, it means that the false positive rate will be lower. The recall score also shows that the model successfully identifies the most positive instances. The ROC-AUC score further confirms the model's overall performance. The classification report provides detailed insights into class-wise performance, ensuring a balanced model evaluation.

Confusion Matrix Code

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

# annot & fmt: Display decimal values
sns.heatmap(cm, annot=True, fmt="d",
            xticklabels=['Non-Hate', 'Hate'],
            yticklabels=['Non-Hate', 'Hate'])


plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.title("Confusion Matrix")
plt.show()
```

Figure 127: Code of Draw Confusion Matrix

The Confusion Matrix helps evaluate how well the model performs by showing the number of correct and incorrect in prediction for each class.

- True Positives (TP) is correctly predicted in positive cases.
- True Negatives (TN) is correctly predicted in negative cases.
- False Positives (FP) is incorrectly predicted in positive cases.
- False Negatives (FN) is incorrectly predicted in negative cases.

Through confusion matrix can calculate some metrics such as accuracy, precision, recall and F1-score to better understand the model's performance. A good model's confusion matrix will have high values for True Positives (TP) and True Negatives (TN), indicating strong performance.

Learning Curve Code

```
import seaborn as sns
import matplotlib.pyplot as plt

accuracy = history.history['accuracy']      # Training accuracy
val_acc = history.history['val_accuracy']   # Validation accuracy
loss = history.history['loss']              # Training loss
val_loss = history.history['val_loss']      # Validation loss
epochs = range(1, len(accuracy) + 1 )       # Number of epochs

sns.set_style("darkgrid")
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4)) # Setting output 1 line with 2 graphs

# Accuracy Curve ----------------------------------------------------------------------------------------------------------
# "bo-": Blue, Circle Dot, Solid Line
axes[0].plot(epochs, accuracy, 'bo-', label='Train Acc', linewidth=2, markersize=5)
axes[0].plot(epochs, val_acc, 'rs-', label='Valid Acc', linewidth=2, markersize=5)
# Labels,Legend and Title
axes[0].set_xlabel('Epoch', fontsize=12)    # Label for X-axis
axes[0].set_ylabel('Accuracy', fontsize=12) # Label for Y-axis
axes[0].set_title('Model Accuracy', fontsize=14, fontweight='bold')
axes[0].legend(loc='upper left', fontsize=10)
axes[0].grid(True, linestyle='--', alpha=0.6)

#  Add accuracy values at each epoch
for i in range(len(accuracy)):           # "ha", "va" is the coordinate of setting text
    axes[0].text(epochs[i], accuracy[i], f"{accuracy[i]:.2f}", ha='right', va='bottom', fontsize=8, color='black')
    axes[0].text(epochs[i], val_acc[i], f"{val_acc[i]:.2f}", ha='right', va='top', fontsize=8, color='black')

# Loss Curve ----------------------------------------------------------------------------------------------------------
axes[1].plot(epochs, loss, 'bo-', label='Train Loss', linewidth=2, markersize=5)
axes[1].plot(epochs, val_loss, 'ro-', label='Valid Loss', linewidth=2, markersize=5)
axes[1].set_xlabel('Epoch', fontsize=12)
axes[1].set_ylabel('Loss', fontsize=12)
axes[1].set_title('Model Loss', fontsize=14, fontweight='bold')
axes[1].legend(loc='lower left', fontsize=10)
axes[1].grid(True, linestyle='--', alpha=0.6)

for i in range(len(loss)):
    axes[1].text(epochs[i], loss[i], f"{loss[i]:.2f}", ha='right', va='bottom', fontsize=8, color='black')
    axes[1].text(epochs[i], val_loss[i], f"{val_loss[i]:.2f}", ha='right', va='top', fontsize=8, color='black')

plt.tight_layout()
plt.show()
```

Figure 128: Code of Plot Learning Curve

The learning curve includes training and validation accuracy and loss to visualize the model's performance over epochs. Help to identify underfitting, overfitting, and convergence trends, ensuring optimal training progress.

ROC-AUC Curve

```
import matplotlib.pyplot as plt

y_pred_prob = model.predict(X_test) # Get predicted probabilities
y_pred = (y_pred_prob > 0.5).astype(int)

# ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(Y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)
print(f"\n📌 ROC AUC: {roc_auc:.4f}")

plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.4f})') # lw = linewidth
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--',label='Random Guess')
plt.xlim([-0.01, 1.0])
plt.ylim([0.0, 1.02])

# Labels and title
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Add legend
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

Figure 129: Code of Plot ROC Curve

The ROC (Receiver Operating Characteristic) curve illustrates the model's ability to distinguish between categories by plotting the True Positive Rate (TPR) and the False Positive Rate (FPR) under different test thresholds. The closer the curve is to the upper left corner, the better the model's ability to make accurate predictions. The higher the AUC (Area Under the Curve), the better the model performs. Conversely, the closer the curve is to the baseline, the weaker the predictive ability.

Precision-Recall Curve

```
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, average_precision_score

precision, recall, _ = precision_recall_curve(Y_test, y_pred_prob)
pr_auc = average_precision_score(Y_test, y_pred_prob)

print(f"\n📌 Precision-Recall AUC: {pr_auc:.4f}")

plt.figure()
plt.plot(recall, precision, color='blue', lw=2, label=f'PR Curve (AUC = {pr_auc:.4f})')

baseline = sum(Y_test) / len(Y_test)
plt.plot([0, 1], [baseline, baseline], color='navy', lw=2, linestyle='--', label='Random Guess')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.02])

plt.xlabel('Recall')
plt.ylabel('Precision')

plt.title('Precision-Recall Curve')
plt.legend(loc="upper right")
plt.show()
```

Figure 130: Code of Precision-Recall Curve

Sometimes, the ROC-AUC score may appear high, indicating that the model performs well, but sometimes ROC-AUC score may be affected by the majority class (non-hate). The PR curve is more suitable for class imbalance problems than the ROC curve. PR-AUC score evaluates model performance by applying different probability thresholds to determine how many positive samples are correctly identified (true positives) and how many are incorrectly classified (false positives), focusing on the precision and recall for the positive class (hate label).

For example, suppose there are 20 examples of hate speech and 10,000 examples of non-hate speech in a dataset. Initially, if 50 examples of non-hate speech are misclassified, the false positive rate is only $\frac{50}{(50+9950)} = 0.005$, which is a small change on the ROC curve. However, there will be a significant drop on the PR curve's $precision = \frac{20}{(20+50)} = 0.286$, so using the PR curve can better see the performance of the model when facing data imbalance.

5.2.2 Evaluation of XGBoost Hate Model

5.2.2.1 Evaluation Result of XGBoost Base Model

XGBoost Base Model default hyperparameter refers to table XGBDefaultValueTable.

```
📌 Train Accuracy: 0.8006
📌 Validation Accuracy: 0.7821

📌 Weighted F1 Score: 0.7855
📌 Micro F1 Score: 0.7821
📌 Macro F1 Score: 0.7560

📌 Precision: 0.6362
📌 Recall: 0.7214

📌 Test ROC AUC: 0.8394
Cross-validation F1 score: [0.75335934 0.74762253 0.74686498 0.74795322 0.75040117]
Average F1 Score: 0.7492402461685435

📌 Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.81      0.84     16860
           1       0.64      0.72      0.68      7761

    accuracy                           0.78     24621
   macro avg       0.75      0.77      0.76     24621
weighted avg       0.79      0.78      0.79     24621
```

Figure 131: XGB Base Model Evaluation Result

The XGBoost model achieved a training accuracy of 80% and validation accuracy of 78% using default hyperparameters. The weighted F1 score reached 0.7855 and the micro and macro F1 scores are 0.7821 and 0.7560. In addition, the cross-validation result of the 5-fold F1 score was 0.7492. ROC-AUC score of 74% represents the overall performance of the model. 74% means that the model is not very good at predicting hate speech. Generally, a better model will have a ROC-AUC score of more than 80%.

The classification report shows that the model performs worse at predicting class 1 (hate) compared to class 0. The low precision of 64% indicates a high false positive rate, while the low recall of 72% indicates a high false negative rate.

Confusion Matrix

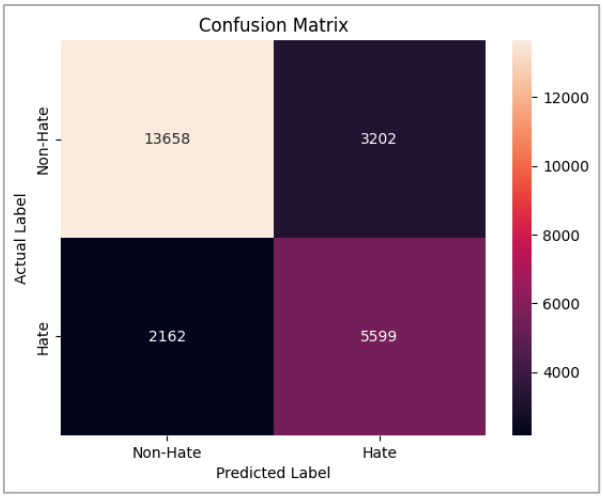


Figure 132: XGB Base Model's Confusion Matrix

XGBoost model with using default hyperparameter correctly classified 13658 rows of data are non-hate speech and 5599 rows of hate. False positive rate is 18.99% and false negative rate is 27.85%.

Learning Curve

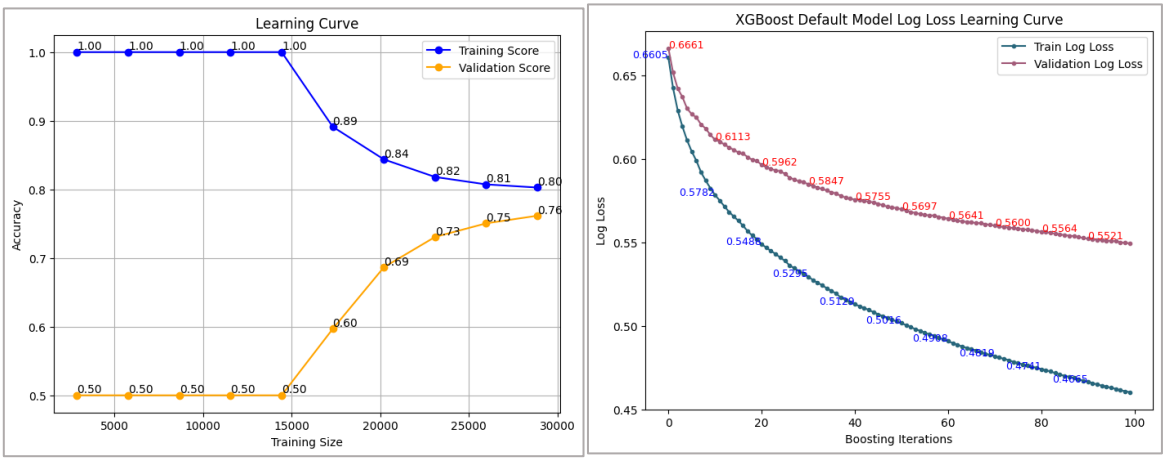


Figure 133: XGB Base Model's Accuracy Curve and Log Loss Curve

The accuracy learning curves show the model's accuracy on the training and validation sets when the training size increases. Initially, the training accuracy remains at 100%, and the validation accuracy remains at 50%. As more data is added, the training accuracy decreases, and the validation accuracy increases, indicating better generalization. Eventually, the train and validation accuracy gap stabilize at around 4%, indicating slight overfitting. The training log loss decreases more rapidly than the validation log loss, indicating overfit through observation of the gap between the two curves.

ROC and Precision-Recall Curve

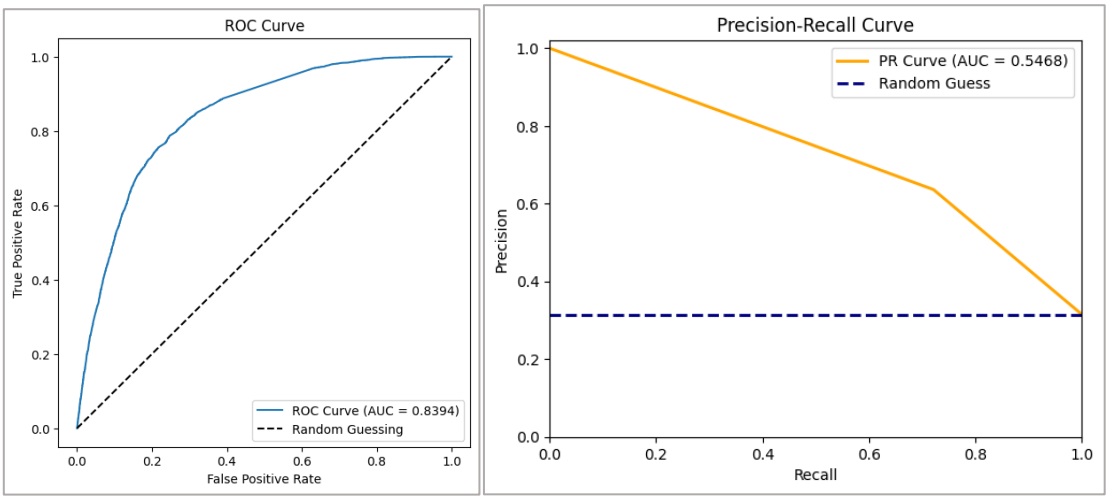


Figure 134: XGB Base Model's ROC Curve (Left)and PR Curve (Right)

The ROC curve shows an AUC of 0.8394, indicating good overall model performance. But Precision-Recall (PR) curve AUC only 0.5468, which is relatively low. This means that the model is difficult to predict with class imbalance or has difficulty maintaining high precision when recall increases. The PR curve is above the random baseline for most recall values but drops sharply at higher recall levels, indicating challenges in handling false positives effectively.

5.2.2.2 Evaluation Result of XGBoost Hyperparameter Tuning

```
Train Accuracy: 0.8460
Validation Accuracy: 0.7646

Weighted F1 Score: 0.7704
Micro F1 Score: 0.7646
Macro F1 Score: 0.7427

Precision: 0.6014
Recall: 0.7502

ROC AUC: 0.8352
```

Figure 135: Evaluated Result of XGB Hyperparameter Tuning

This figure showing the result of hyperparameter tuning, hyperparameter setting can refer to table XGBHyperparameter Tuning Values Setting. The XGBoost hyperparameter tuning optimized using RandomizedSearchCV with 5-fold cross-validation, achieved the training accuracy of 84% and validation accuracy of 76%. The weighted F1-score is 0.7704, the micro F1-score is 0.7646 and the macro F1-score is 0.7427. The model attained a precision of 0.6014,

a recall of 0.7502, and an AUC score of 0.8352. Lower precision indicates a higher likelihood of incorrectly predicting non-hate speech as hate speech.

```
📌 Classification Report:
                precision    recall  f1-score   support

           0       0.87      0.77      0.82     16860
           1       0.60      0.75      0.67      7761

    accuracy                           0.76     24621
   macro avg       0.74      0.76      0.74     24621
weighted avg       0.79      0.76      0.77     24621
```

Figure 136: XGB Hyperparameter Tuning Classification Report

This classification report shows that the overall validation accuracy is 76%. For class 1 (hate speech), the recall is 75% and the precision is 60%. While the model is effective at capturing hate speech, lower precision may lead to more misclassifications of non-hate speech as hate speech. For label 0 (non-hate) has more data, the overall F1 score is over 82.

Confusion Matrix

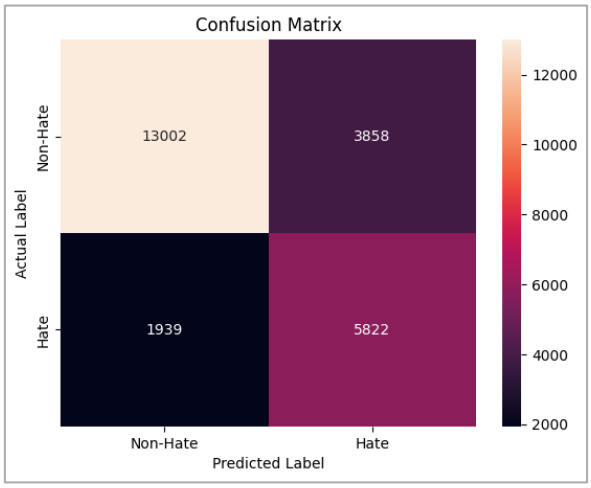


Figure 137: XGB Hyperparameter Tuning Confusion Matrix

Using the "randomSearchCV" hyperparameter tuning method with 5-fold cross-validation, the XGBoost model correctly classified 13002 rows of data as non-hate speech and 5822 rows as hate speech. The false positive rate is 22.88%, and the false negative rate is 24.98%.

Learning Curve

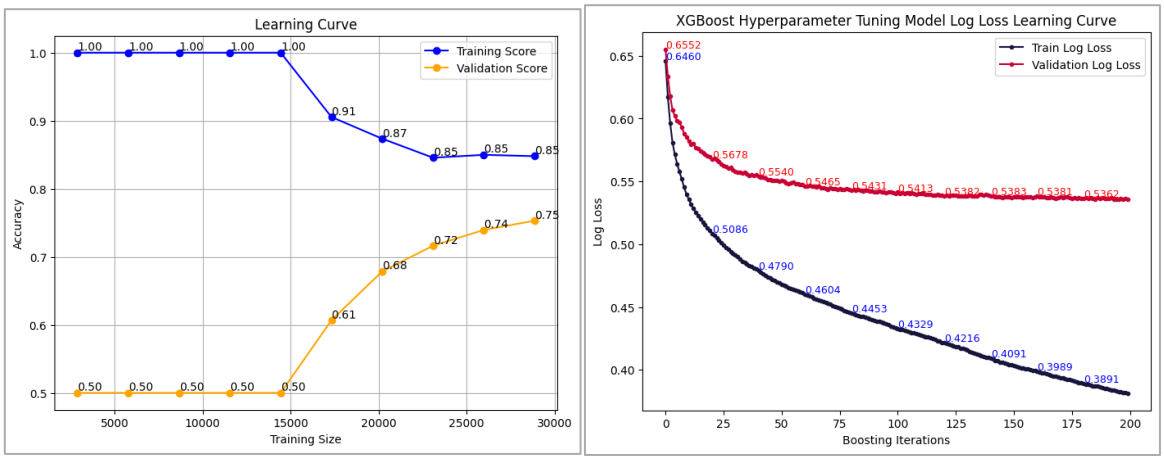


Figure 138: XGB Hyperparameter Tuning Accuracy Curve (Left) and Log Loss Curve (Right)

After applying the best hyperparameter to the model, as the training size increases, the training accuracy stabilizes at 85%, while the validation accuracy improves to 75%. The 10% gap between training and validation accuracy, along with the noticeable difference between the training and validation log loss curves, showed that there was overfitting in the model.

ROC Curve and Precision-Recall Curve

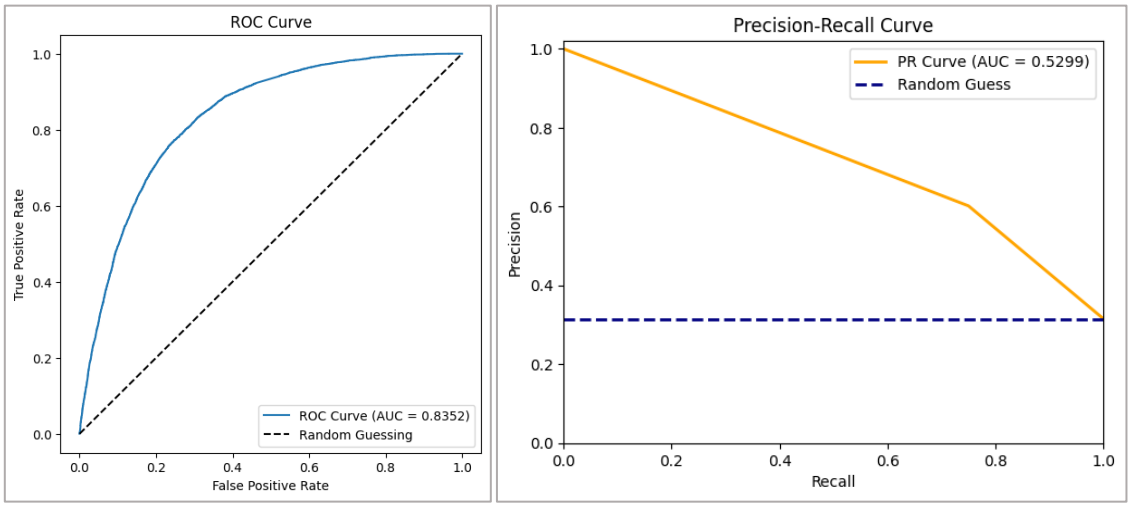


Figure 139: XGB Hyperparameter Tuning ROC Curve (Left) and PR Curve (Right)

These two figures show that although the ROC-AUC score exceeds 80% (83.52%), the PR-AUC score only reaches 52.99%. This indicates that the overall performance of the model is biased towards the majority label (non-hate) and ignores the minority hate labels.

5.2.2.3 Comparison Between XGBoost Hyperparameter Tuning and Base Model

Table 24: Comparison Between Hyperparameter Tuning and Base Model

Model / Evaluation Matrix	Base Model (Default Value)	Hyperparameter Tuning Model
Train Accuracy	0.8006	0.8460
Validation Accuracy	0.7821	0.7646
Weighted F1 Score	0.7855	0.7704
Micro F1 Score	0.7821	0.7646
Macro F1 Score	0.7560	0.7427
Precision	0.6362	0.6014
Recall	0.7214	0.7502
ROC-AUC	0.8394	0.8352
PR-AUC Score	0.5468	0.5299
False Positive Rate	18.99%	22.88%
False Negative Rate	27.85%	24.98%
Accuracy Gap	4%	10%

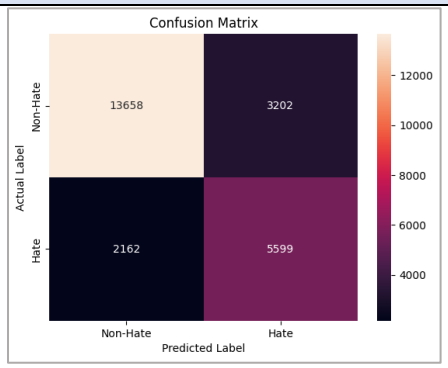
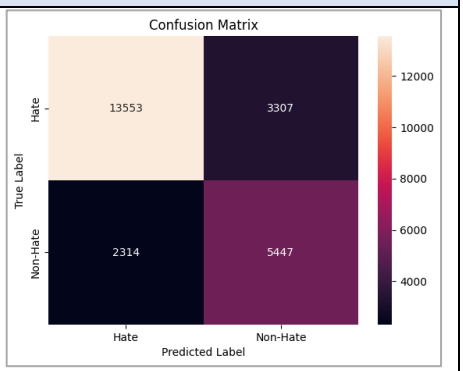
Comparing the two models, the XGB model with default values performs better than hyperparameter tuning. Both models have difficulty in detecting the minority class (hate label), with PR-AUC Score only achieving around 50%. Additionally, the hyperparameter-tuned model shows signs of severe overfitting, with a larger accuracy gap is 10% and a higher false positive rate of 22.88%. This suggests that hyperparameter tuning led to over-optimization of the training data.

5.2.2.4 Comparison Between XGBoost Base Model and Fine-Tuning Model

Table 25: Comparison Between Base Model and Fine-Tuning Model

Model / Evaluation Matrix	Base Model (Default Value)	Fine-Tuning Model
Train Accuracy	0.8006	0.7769
Validation Accuracy	0.7821	0.7717
Weighted F1 Score	0.7855	0.7751
Micro F1 Score	0.7821	0.7717
Macro F1 Score	0.7560	0.7439
Precision	0.6362	0.6222
Recall	0.7214	0.7018
ROC-AUC	0.8394	0.8277
PR-AUC Score	0.5468	0.5307
Accuracy Gap	4%	3%

Table 26: XGB Comparison of Base Model and Fine-Tuning Model (FPR & FNR)

	Base Model (Default Value)	Fine Tuning Model
		
False Positive Rate (FPR)	18.99%	19.61%

False Negative Rate (FNR)	27.85%	29.82%

To prevent the base model from overfitting, different hyperparameters were adjusted, but the results were similar, with the accuracy gap between the training set and the validation set between 2% and 4%. On the contrary, adding "reg_lambd" and "reg_alpha" to the fine-tuning model resulted in higher false positive rate of 19.61% and false negative rate of 29.82%. The PR AUC score of the fine-tuned model and the base model dropped by 1% from 54% to 53%, indicating that the model performs very poorly in detecting hate speech.

XGB Fine Tuning Learning Curve and Log Loss Curve

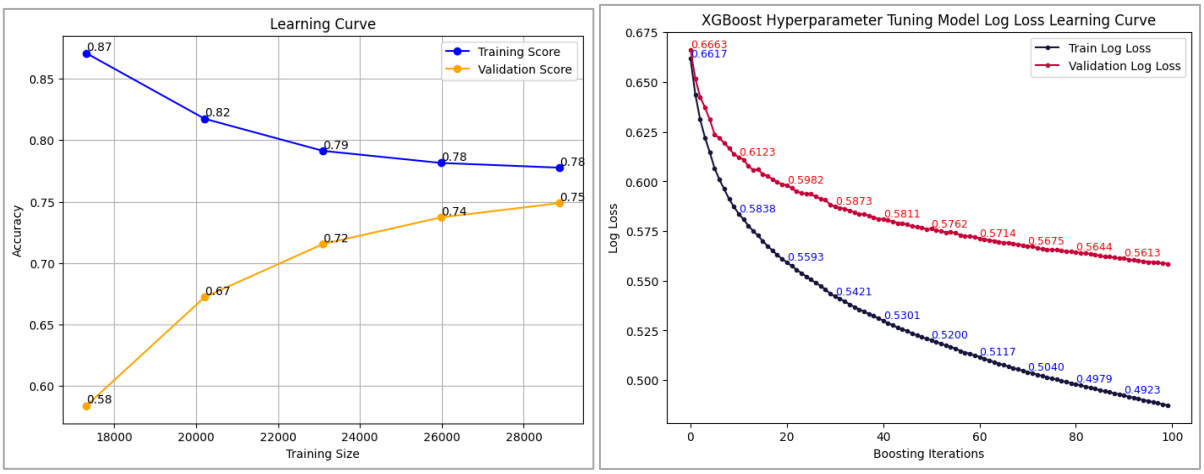


Figure 140: XGB Fine Tuning Accuracy Curve (Left) and Log Loss Curve (Right)

The base model has a log loss difference of 0.086 and an accuracy gap of 4%. After fine-tuning, the train-validation log loss difference is reduced to 0.069, and the train-validation accuracy gap decreases to 3%.

Overall, the performance of the base XGBoost and fine-tuned models on complex text datasets is very weak, PR-AUC score only 0.55 and 0.53, and the fine-tuning result is not better than the base model result, although the train-validation accuracy gap of the fine-tuning model is 1% less than that of the base model.

5.2.3 Evaluation of Bi-LSTM Hate Model

5.2.3.1 Evaluation Result of Bi-LSTM Base Model

Bi-LSTM Base Model default value of hyperparameter refers to the table BiLSTM Model Default Hyperparameter.

```
📌 Train Accuracy: 0.8926
📌 Validation Accuracy: 0.7804

📌 Weighted F1 Score: 0.7876
📌 Precision: 0.6076
📌 Recall: 0.8512

📌 ROC AUC: 0.8738
```

Figure 141: Evaluation Result of Bi-LSTM Base Model

```
📌 Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.75      0.82     16880
           1       0.61      0.85      0.71      7741

    accuracy                           0.78     24621
   macro avg       0.76      0.80      0.77     24621
weighted avg       0.82      0.78      0.79     24621
```

Figure 142: Bi-LSTM Base Model Classification Report

Using the default hyperparameter settings, the model has validation accuracy of 78%, weighted F1 score of 79%, binary precision is 60%, binary recall of 85%, and ROC-AUC of 87%. There is an 11% difference between training and validation accuracy, which indicates that the model is not able to predict well in new input of data. In addition, the precision is lower than the recall, with a precision of only 60%, which means that many positive examples are predicted incorrectly. However, ROC-AUC 87% shows that the model still has a good overall ability to distinguish between categories.

The classification report shows that for class 0 (non-hate) with high precision 92% but lower recall 75%. For class 1, recall is strong 85%, but precision is relatively low 61%.

Confusion Matrix

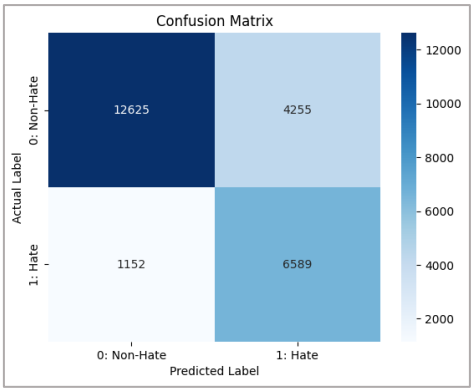


Figure 143: Bi-LSTM Base Model Confusion Matrix

The confusion matrix shows the model correctly classified 12625 rows of data are non-hate speech and 6,589 rows of data are hate speech. The model misclassified 4255 rows of non-hate speech data as hate speech (false positive) and 1152 rows of hate speech data as non-hate (false negative). This shows that although the model has a high precision (92%) for class 0 can effectively detect non-hate speech, but still has 25% of false positive problems, which means that some non-hate speech is mistakenly labelled as hate speech. For class 1 (hate), the false negative rate is 14% ($\frac{152}{152+6589} * 100$), for class 0 (non-hate), the false positive rate is 25% ($\frac{4255}{4255+12625} * 100$).

Learning Curve

Figure 144: Bi-LSTM Base Model Learning Curve

The learning curve shows the accuracy and loss when evaluating the model at each epoch. The difference between the training loss and the validation loss at epoch 5 is 0.67, and the difference between the training accuracy and the validation accuracy at epoch 5 is 17%, indicating that the model is overfitting when using the default values and without any dropout.

ROC-AUC Curve & Precision-Recall Curve

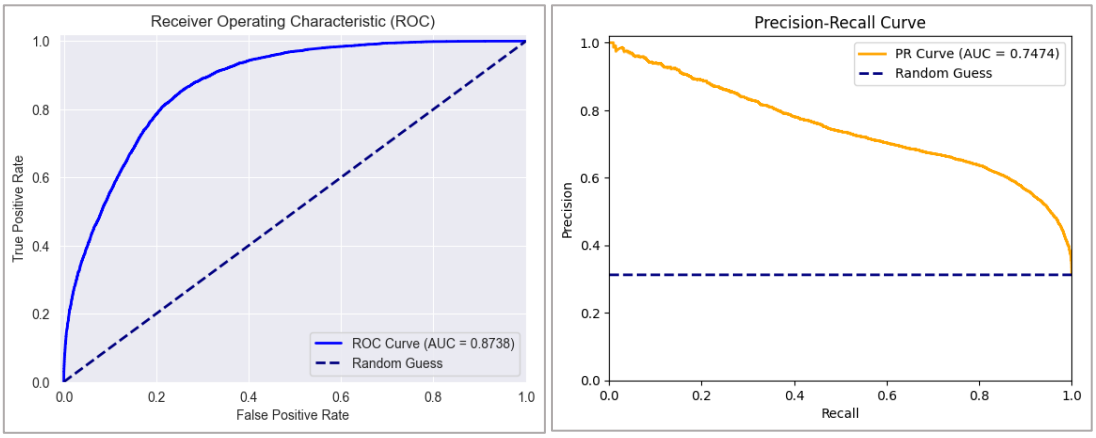


Figure 145: Bi-LSTM Base Model ROC Curve (Left) and PR Curve (Right)

The model's ROC-AUC score is 87% has a good overall performance, but the PR-AUC score is 74%, which is lower than 13% of the ROC-AUC score, indicating ROC-AUC score is biased towards non-hate speech due to class imbalance. The ROC curve remains high even with some false positives, while the PR curve shows precision drop when recall increases.

5.2.3.2 Comparison Bi-LSTM Hyperparameter Tuning and Base Model

Hyperparameter tuning setting in detail is shown in table Bi-LSTM Best Hyperparameter Tuning.

Table 27: Evaluated Result and Comparison Between Hyperparameter Tuning and Base Model

Model / Evaluation Matrix	1st HP Tuning	2nd HP Tuning	3rd HP Tuning	4th HP Tuning	5th HP Tuning	Base Model (Default Value)
Train Accuracy	0.8621	0.8775	0.9150	0.8598	0.9279	0.8926
Validation Accuracy	**0.8379**	0.7818	0.7632	**0.7964**	0.7878	0.7804
Weighted F1 Score	**0.8426**	0.7889	0.7709	**0.8030**	0.7932	0.7876
Micro F1 Score	**0.8379**	0.7818	0.7632	**0.7964**	0.7878	0.7804

Macro F1 Score	**0.8254**	0.7672	0.7475	**0.7828**	0.7680	0.7663
Binary Precision	**0.6833**	0.6107	0.5889	**0.6273**	0.6299	0.6076
Binary Recall	**0.9053**	0.8442	0.8176	**0.8684**	0.7885	0.8512
ROC-AUC Score	**0.9094**	0.8674	0.8268	**0.8911**	0.8658	0.8738
PR-AUC Score	0.6062	**0.7499**	0.5924	**0.7744**	0.7298	0.7474
Learning Curve (in 10 Epoch)						
Train Loss	0.95	0.50	0.59	0.44	0.61	0.82
Validation Loss	0.12	0.36	0.29	0.36	0.24	0.15
Loss Gap	0.83	0.14	0.3	**0.08**	0.37	0.67

** The bolded results are ranked first and second, and the red words represent bad results.*

According to the table above, the first hyperparameter tuning has the highest overall validation accuracy, as well as the highest binary precision and recall. However, the PR score is only 60%, which is lower than the general standard of 80%. This shows that although the 1st tuning model performs well in validation accuracy, the model has difficulty detecting hate speech in an unbalanced dataset. Therefore, considering that the model is designed to focus on detecting hate speech (class 1), the fourth hyperparameter tuning shows the best performance with the highest PR score 77%. The difference between training and validation loss is 0.08, which is the smallest difference among the 4 times of hyperparameter tunings, indicating that the learning curve is balanced, and the difference in training and validation accuracy is 0.06, confirming that the model is neither overfitting nor underfitting.

According to the comparison results, the fifth hyperparameter tuning used Word2Vec word embedding technology, and the performance of the trained model was not as good as the model based on FastText. Of course, the performance difference cannot be attributed solely to word embeddings, as all hyperparameters were selected through random search, making the results subject to variation.

4th Hyperparameter Tuning Classification Report

📌 Classification Report:

	precision	recall	f1-score	support
0	0.93	0.76	0.84	16880
1	0.63	0.87	0.73	7741
accuracy			0.80	24621
macro avg	0.78	0.82	0.78	24621
weighted avg	0.83	0.80	0.80	24621

Figure 146: 4th Bi-LSTM Hyperparameter Tuning Classification Report

The classification report shows that the model has an overall accuracy of 80% and a weighted F1 score of 80%. For class 0 non-hate, the accuracy is 93% and the recall is 76%. For class 1 hate labels, the recall is 87%, which means that most instances of hate speech are correctly identified, with an accuracy of 63%, resulting in a high false positive rate.

4th Hyperparameter Tuning Confusion Matrix

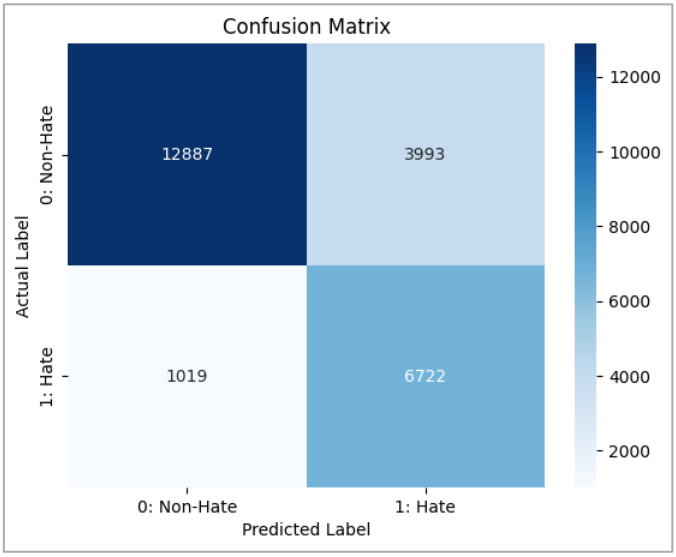


Figure 147: 4th Bi-LSTM Hyperparameter Tuning Confusion Matrix

The fourth tuning result shows a false positive rate (FPR) of 23.66% and a false negative rate (FNR) of 13.16%.

4th Hyperparameter Tuning Learning Curve

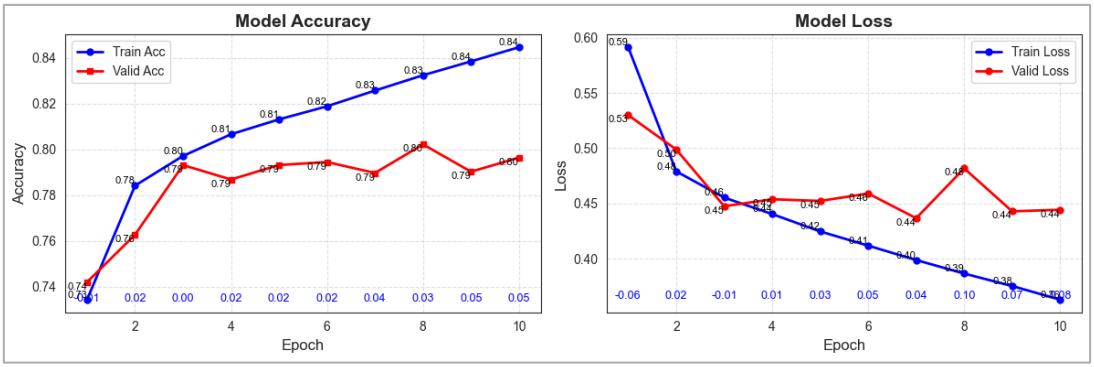


Figure 148: 4th Bi-LSTM Hyperparameter Tuning Learning Curve

This figure shows that the fluctuation of the learning curve is still a bit large, although the overall accuracy and loss gap are not very large. The fourth hyperparameter tuning would perform the final fine-tuning and try to prevent overfitting and increase the validation accuracy, f1 score and make the validation curve more stable.

5.2.3.3 Comparison Bi-LSTM Hyperparameter Tuning and Fine-Tuning Model

```
# When saving the model, Keras will automatically recognize my_custom_loss and load it correctly
@register_keras_serializable(package="CustomLoss")
# Function focal_loss code et from GitHub (aldi-dimara, 2018)
# Gamma, Alpha default values recommended by the paper (Lin et al., 2018) # https://arxiv.org/pdf/1708.02002v
def focal_loss(y_true, y_pred, gamma=2.0, alpha=0.55): # 1.5,2.0 # 0.25,0.75,0.55,0.51
    epsilon = K.epsilon()
    y_pred = K.clip(y_pred, epsilon, 1.0-epsilon)
    # Predicted probability of the correct class
    pt = tf.where(K.equal(y_true, 1), y_pred, 1-y_pred)
    # Alpha factor for balancing class imbalance.
    alpha_factor = K.ones_like(y_true)*alpha
    # Dynamically assigns different alpha values to different classes
    alpha_t = tf.where(K.equal(y_true, 1), alpha_factor, 1-alpha_factor)
    # Cross-entropy loss
    cross_entropy = -K.log(pt)
    # Alpha > 0.5, then class 1 weigth > class 0 weigth. Alpha < 0.5, then class 0 weigth > class 1 weigth
    weight = alpha_t * K.pow((1-pt), gamma) #alpha_t*(1 - pt) ^ gamma
    loss = weight * cross_entropy
    # loss = K.sum(loss, axis=1)
    loss = K.mean(loss, axis=1)
    return loss
```

Figure 149: Bi-LSTM Fine Tuning Code Added 1.0

```
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,           # Decrease LR (current LR * 0.5)
    patience=1,           # If val_loss does not improve for 1 epoch, reduce LR
    min_lr=0.0001,        # Setting minimum learning rate to prevent it from being too low
    verbose=1
)
```

Figure 150: Bi-LSTM Fine Tuning Code Added 2.0

In the fine-tuning step, different factor values (0.9, 0.8, 0.5, and 0.2) were tested for the "ReduceLROnPlateau" function. The results were quite similar across all values, but 0.5 yielded the best performance. In Focal Loss function try different alpha values (0.25, 0.27, 0.55, 0.51) and gamma values (1.5, 2.0). Alpha value chosen is 0.55, and Gamma value chosen is 2.0 model has a better performance.

Table 28: Bi-LSTM Comparison 2.0

Model / Evaluation Matrix	4th HP Tuning	Fine Tuning
Train Accuracy	**0.8598**	0.8371
Validation Accuracy	0.7964	**0.7969**
Weighted F1 Score	0.8030	0.8030
Micro F1 Score	0.7964	**0.7969**
Macro F1 Score	**0.7828**	0.7813
Binary Precision	0.6273	**0.6330**
Binary Recall	**0.8684**	0.8425
ROC-AUC Score	**0.8911**	0.8845
PR-AUC Score	**0.7744**	0.7558

📌 Classification Report:

	precision	recall	f1-score	support
0	0.91	0.78	0.84	16880
1	0.63	0.84	0.72	7741
accuracy			0.80	24621
macro avg	0.77	0.81	0.78	24621
weighted avg	0.83	0.80	0.80	24621

Figure 151: Bi-LSTM Fine-Tuning Classification Report

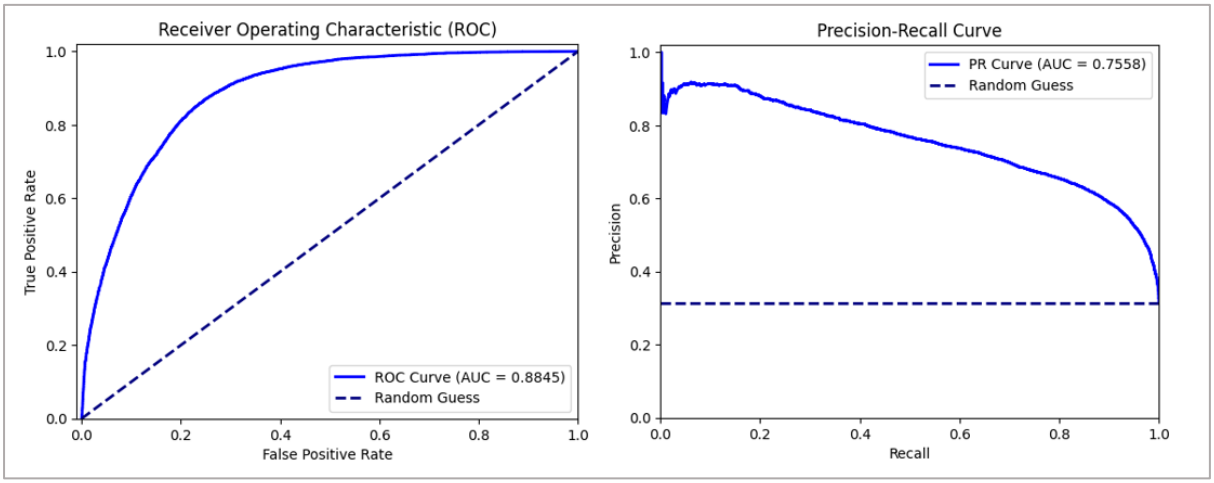
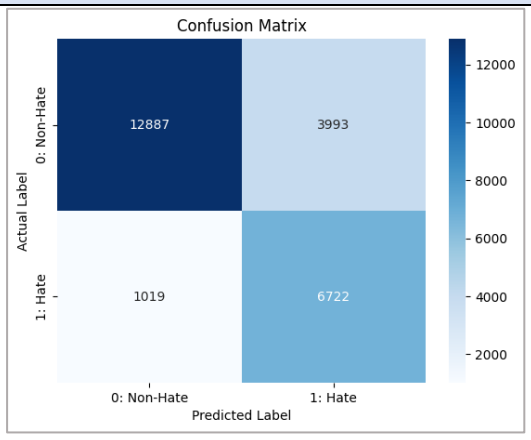
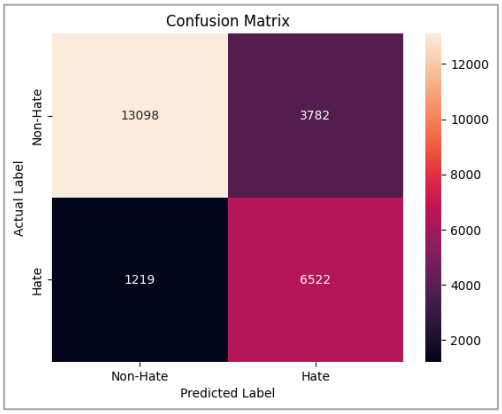


Figure 152: Bi-LSTM Fine Tuning ROC Curve (Left) and PR Curve (Right)

The fine-tuning process resulted in no overfit model performance compared to the 4th hyperparameter tuning. Train accuracy dropped from 85.98% to 83.71%. However, validation accuracy remained nearly the same at 79.64% and 79.69%. The weighted and micro F1 scores stayed constant at 0.8030, while the macro F1 score had a minor decrease (0.7828 to 0.7813). Binary precision improved from 0.6273 to 0.6330, decreasing the false positives rate, but binary recall decreased (0.8684 to 0.8425), showing more false negatives. The ROC-AUC score of 0.8911 was reduced to 0.8845 and the PR-AUC score from 0.7744 to 0.7558, indicating a small drop in overall classification performance.

Table 29: Bi-LSTM Comparison of HP Tuning Model and Fine-Tuning Model (FPR & FNR)

	4th HP Tuning	**Fine Tuning**
		
FPR	23.66%	**22.41%**
FNR	**13.16%**	15.75%

The fine-tuning model's false positive rate (FPR) decreased from 23.66% to 22.41%, with some improvement in reducing misclassified non-hate speech. However, the false negative rate (FNR) increased from 13.16% to 15.75%.

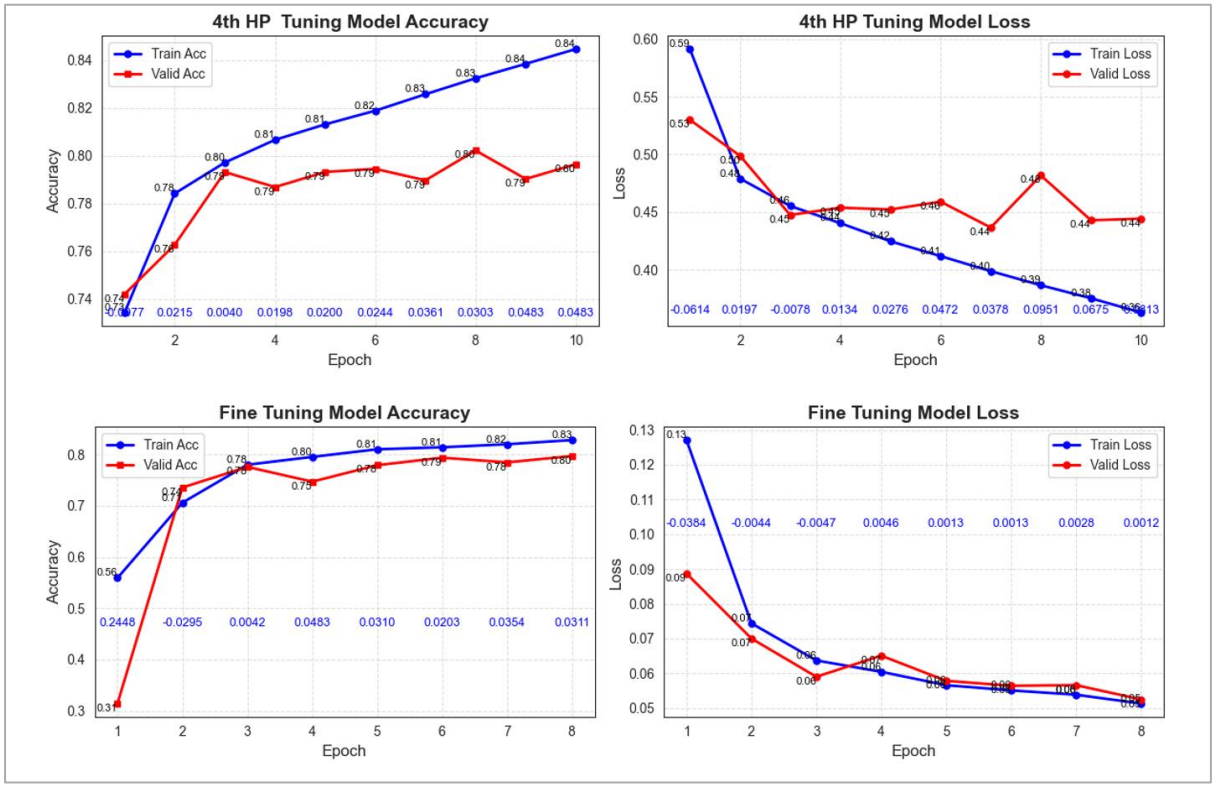


Figure 153: Bi-LSTM Hyperparameter Tuning & Fine-Tuning Learning Curve

According to the table of Bi-LSTM Comparison 2.0, the two results are similar 4th hyperparameter evaluation matrix performance is better than fine-tuning. Through observing the learning rate of the fourth hyperparameter tuning a bit overfitting while fine-tuning the learning curve is more stable than 4th hyperparameter tuning.

5.2.4 Evaluation of Bi-GRU Hate Model

5.2.4.1 Evaluation Result of Bi-GRU Base Model

```
📌 Train Accuracy: 0.8942
📌 Validation Accuracy: 0.7800

📌 Weighted F1 Score: 0.7874
📌 Micro F1 Score: 0.7800
📌 Macro F1 Score: 0.7671

📌 Precision: 0.6049
📌 Recall: 0.8658

📌 ROC AUC: 0.8774
```

Figure 154: Evaluation Result of Bi-GRU Base Model

Using the default hyperparameter settings, the model achieves validation accuracy of 78%, a weighted F1 score of 79%, a binary precision of 60%, and a binary recall of 87%. The ROC-AUC score of 88% indicates that the model still has a strong overall ability to distinguish between classes.

```
📌 Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.74      0.82     16880
           1       0.60      0.87      0.71      7741

    accuracy                           0.78     24621
   macro avg       0.76      0.80      0.77     24621
weighted avg       0.82      0.78      0.79     24621
```

Figure 155: Bi-GRU Base Model Classification Report

The classification report shows that the model has a high precision of 92% for category 0 (non-hate), which means that the model can correctly identify most non-hate samples, but the recall rate is low (74%), indicating that it missed some actual non-hate instances. On the contrary, for category 1 (hate), the model successfully identified most hate speech instances with a high recall rate of 87%, but the precision rate was 60%, indicating that the false positive rate of hate speech detection is high.

Confusion Matrix

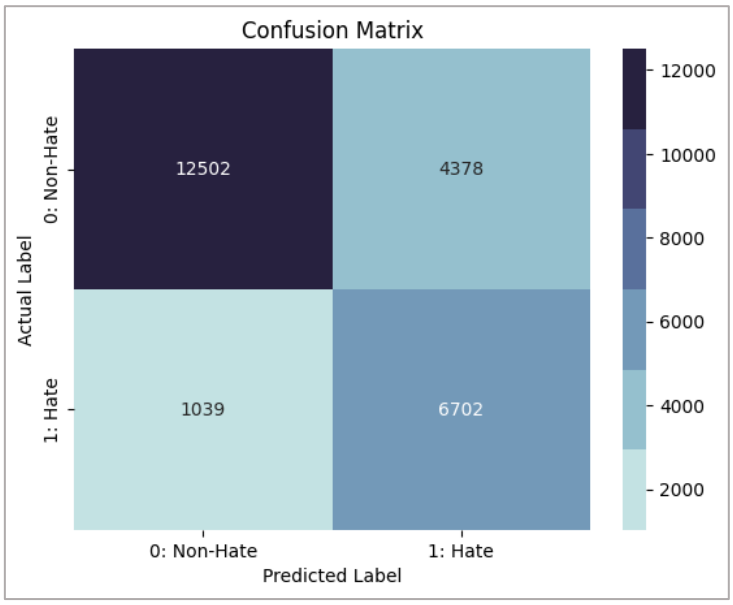


Figure 156: Bi-GRU Base Model Confusion Matrix

Bi-GRU model with using default hyperparameter correctly classified 12502 rows of data are non-hate speech and 6702 rows of hate. False positive rate is 25% and false negative rate is 13.42%. This dataset has large proportion of non-hate labels, but in this Bi-GRU base model can detect hate speech well with a low false negative rate. However, the false positive rate is very high, exceeding 20%. This causes many non-hate data to be misclassified as hate, which affects the reliability of the model.

Learning Curve

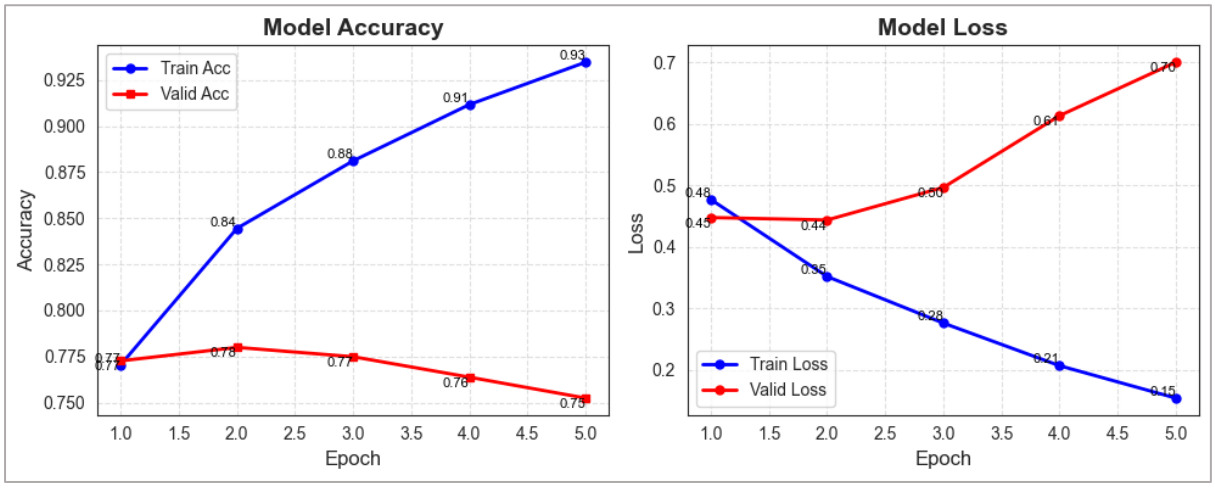


Figure 157: Bi-GRU Base Model Learning Curve

The model is severely overfitting, with a train and validation loss gap of 0.55. While the model performs exceptionally well on the training dataset, the accuracy on the validation set continues to drop, indicating that it is not effective in detecting unknown data. This suggests that the

model is learning patterns that are too specific to the training data, rather than capturing hate speech features.

ROC and Precision-Recall Curve

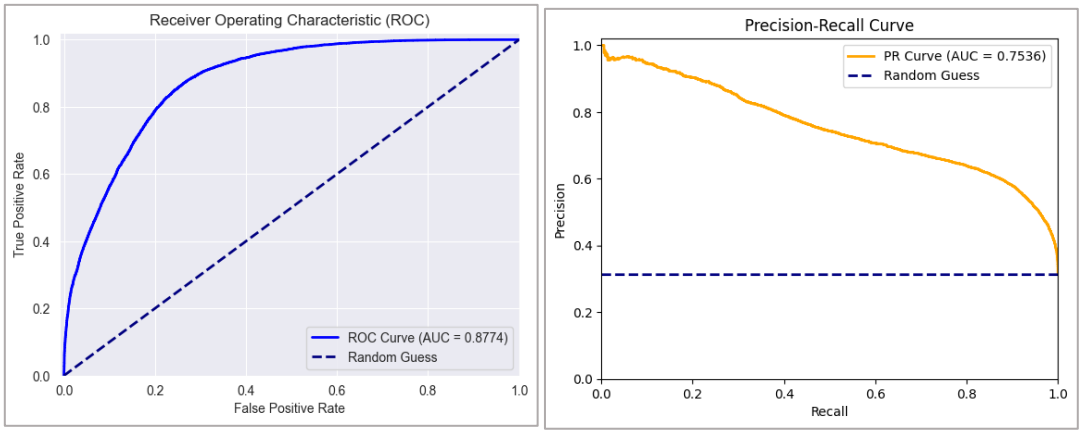


Figure 158: Bi-GRU Base Model ROC Curve (Left) and PR Curve (Right)

The ROC-AUC score is 0.8774, indicating that the model performs well in distinguishing the categories. However, the AUC score of the precision-recall (PR) curve is 0.7536, which means that the model has low accuracy in detecting the minority class (hate speech). These results indicate that the model is biased towards the majority class (non-hate).

5.2.4.2 Evaluation Result of Bi-GRU Hyperparameter Tuning

```
Train Accuracy: 0.8543
Validation Accuracy: 0.7797

Weighted F1 Score: 0.7872
Micro F1 Score: 0.7797
Macro F1 Score: 0.7671

Precision: 0.6039
Recall: 0.8702

ROC AUC: 0.8809
```

Figure 159: Evaluation Result of Bi-GRU Hyperparameter Tuning

After hyperparameter tuning find the best hyperparameter, the model achieves a validation accuracy of 77%, a weighted F1 score of 79%, a binary precision of 60%, and a binary recall of 87%. The ROC-AUC score of 88%.

Classification Report

Classification Report:

	precision	recall	f1-score	support
0	0.93	0.74	0.82	16880
1	0.60	0.87	0.71	7741
accuracy			0.78	24621
macro avg	0.76	0.80	0.77	24621
weighted avg	0.82	0.78	0.79	24621

Figure 160: Bi-GRU Hyperparameter Tuning Classification Report

The classification report shows that the model has a high precision of 93% and a recall low of 74% for category 0 (non-hate). On the contrary, for category 1 (hate), high recall rate of 87%, but the precision rate was 60%.

Confusion Matrix

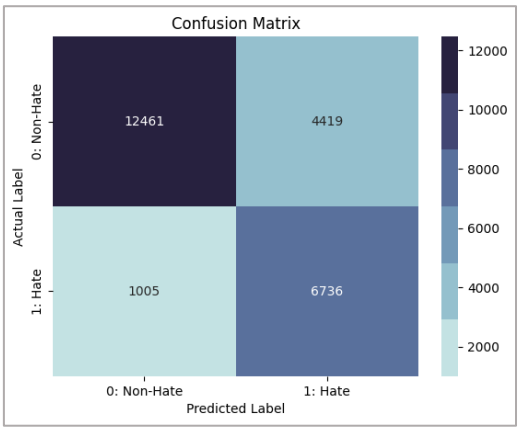


Figure 161: Bi-GRU Hyperparameter Tuning Confusion Matrix

Bi-GRU model with using default hyperparameter correctly classified 12461 rows of data are non-hate speech and 6736 rows of hate. False positive rate is 26.17% and false negative rate is 12.98%.

Learning Curve

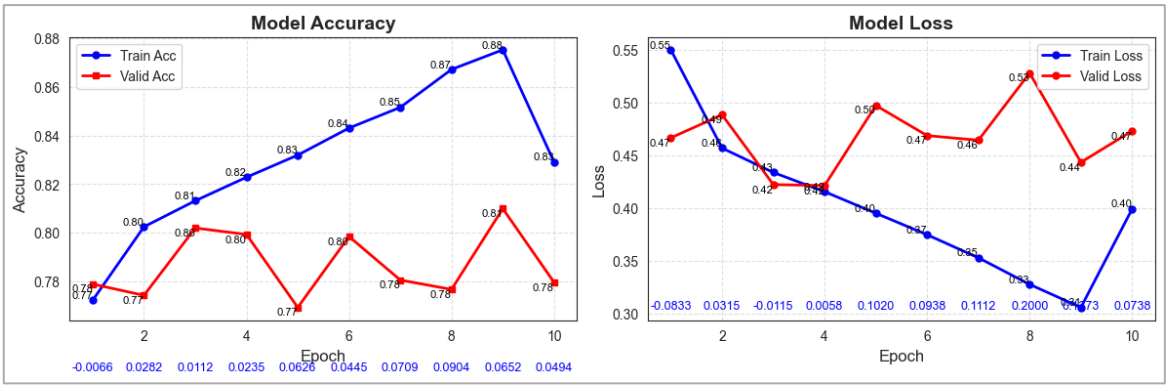


Figure 162: Bi-GRU Hyperparameter Tuning Learning Curve

From the chart, showing that the accuracy of the training set keeps increasing, while the accuracy of the validation set fluctuates greatly. It does not increase synchronously with the training set but has a downward trend. The training loss continues to decrease, but the valid loss fluctuates and tends to increase overall. The accuracy rate sharp drop and loss rate sharp increase in the last epoch may be due to problems such as overfitting due to too high learning rate and gradient explosion.

ROC and Precision-Recall Curves

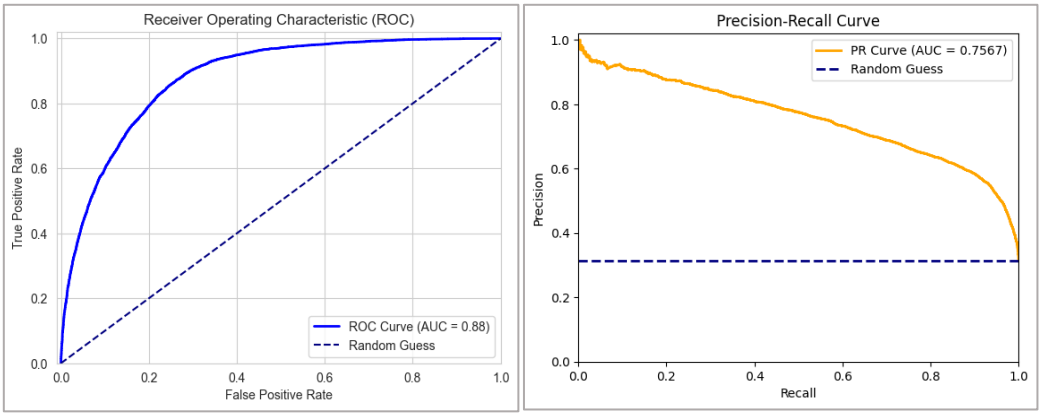


Figure 163: Bi-GRU Hyperparameter Tuning ROC Curve (Left) and PR Curve (Right)

The model performs well in a general classification sense but exhibits imbalance when detecting the minority class.

5.2.4.3 Comparison Between Bi-GRU Hyperparameter Tuning and Base Model

Table 30: Comparison Between Bi-GRU Hyperparameter Tuning and Base Model

Model / Evaluation Matrix	Default Value	Hyperparameter Tuning
Train Accuracy	0.8942	0.8543
Validation Accuracy	0.7800	0.7797
Weighted F1 Score	0.7874	0.7872
Micro F1 Score	0.7800	0.7797
Macro F1 Score	0.7671	0.7671
Binary Precision	0.6049	0.6039
Binary Recall	0.8658	0.8702
ROC-AUC Score	0.8774	0.8809
PR-AUC Score	0.7536	0.7567
FPR	25.93%	26.17%
FNR	13.42	12.98%
Train Loss	0.7	Not stable
Valid Loss	0.15	Not stable
Loss Gap	0.55	-

The current hyperparameter tuning may not have found the best combination. Although the model loss has decreased (Figure: Bi-GRU Hyperparameter Tuning Learning Curve), it is precarious.

5.2.4.4 Comparison Bi-GRU Hyperparameter Tuning and Fine-Tuning Model

Table 31: Comparison Between Bi-GRU Hyperparameter Tuning and Fine-Tuning Model

Model / Evaluation Matrix	Hyperparameter Tuning	Fine Tuning
Train Accuracy	**0.8543**	0.8283
Validation Accuracy	0.7797	**0.7993**
Weighted F1 Score	0.7872	**0.8054**
Micro F1 Score	0.7797	**0.7993**
Macro F1 Score	0.7671	**0.7841**
Binary Precision	0.6039	**0.6353**
Binary Recall	**0.8702**	0.8490
ROC-AUC Score	0.8809	**0.8865**
PR-AUC Score	0.7567	**0.7627**

📌 Classification Report:

	precision	recall	f1-score	support
0	0.92	0.78	0.84	16880
1	0.64	0.85	0.73	7741
accuracy			0.80	24621
macro avg	0.78	0.81	0.78	24621
weighted avg	0.83	0.80	0.81	24621

Figure 164: Bi-GRU Fine-Tuning Model Classification Report

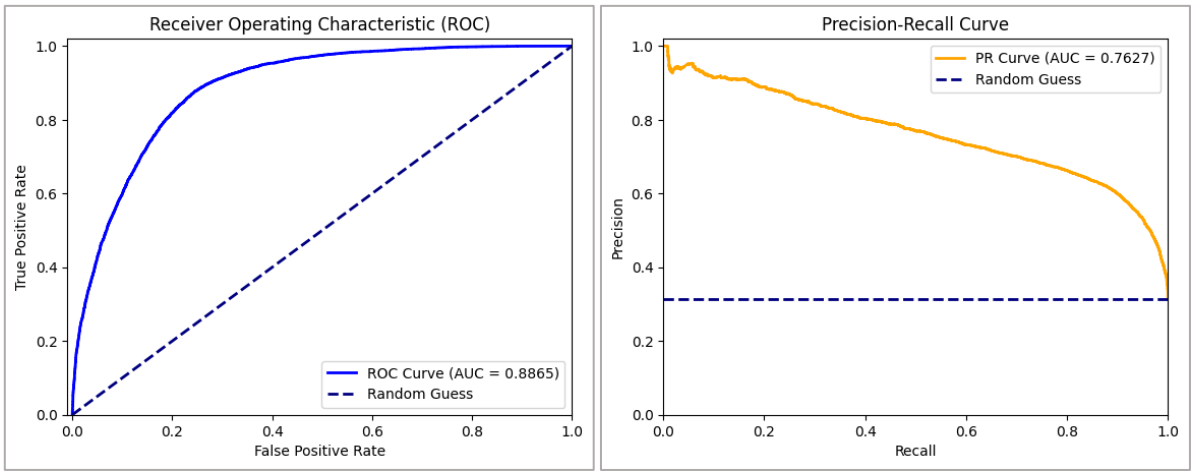
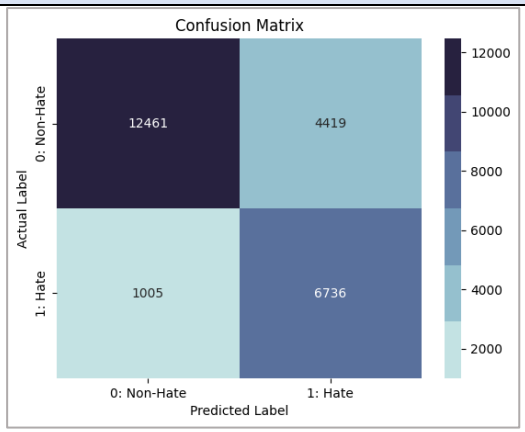
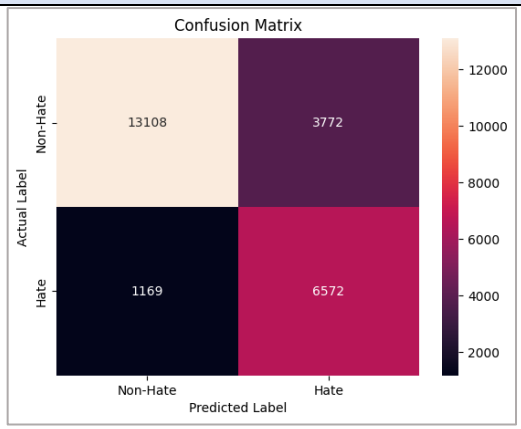


Figure 165: Bi-GRU Fine-Tuning Model ROC Curve and PR Curve

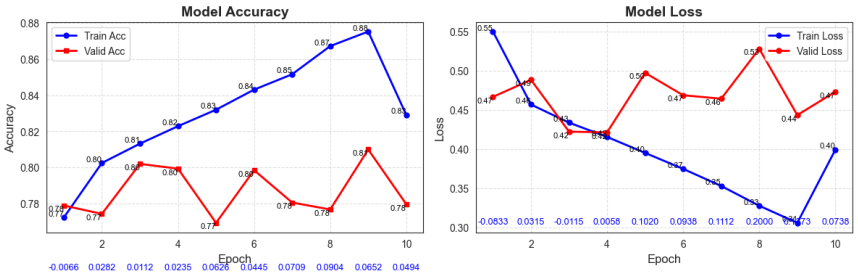
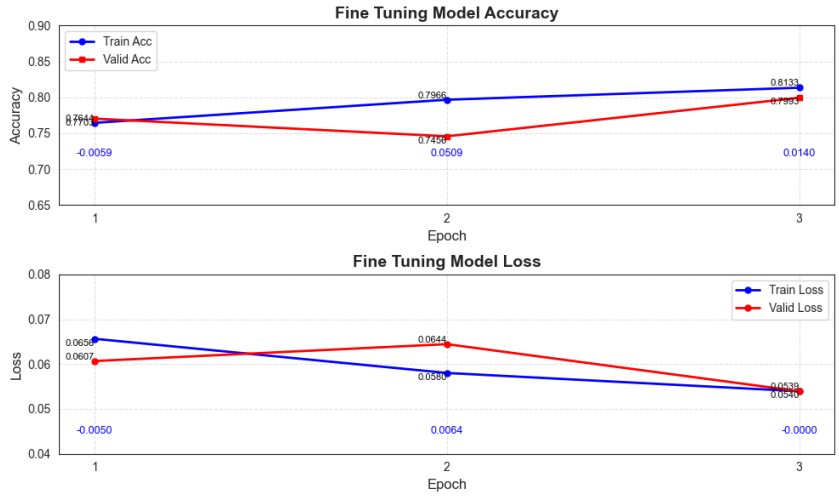
Compared to the results of hyperparameter tuning, the fine-tuning stage yielded improved performance across all key evaluation metrics. Validation accuracy increased by 2%, reaching 0.7993, while the F1 score also rose by 2% to 0.8054. In addition, the model achieved ROC-AUC score of 0.8865 and a PR-AUC score of 0.7627. Fine-tuning also addressed the issue of imbalanced binary classification performance, where previously the recall for the positive class was excessively high and the precision was relatively low. After fine-tuning, the model showed a more balanced outcome. In the classification report, class 1 achieved a recall of 85% and a precision of 64%, resulting in an overall F1 score of 80%.

Table 32: Bi-GRU Comparison of HP Tuning Model and Fine-Tuning Model (FPR & FNR)

	Hyperparameter Tuning	**Fine Tuning**
		
FPR	26.17%	**22.35%**
FNR	**12.98%**	15.10%

Compared the false positive rate (FPR) and false negative rate (FNR) between the hyperparameter tuning and fine-tuning stages. The FPR decreased significantly from 26.17% to 22.35%, indicating that the fine-tuned model made fewer incorrect positive predictions. This suggests an improvement in model precision. FNR slightly increased from 12.98% to 15.10%, meaning the fine-tuned model missed slightly more true positive cases.

Table 33: Bi-GRU Comparison of HP Tuning Model and Fine-Tuning Model (Learning Curve)

	Learning Curve
Hyperparameter Tuning	
Fine Tuning	

When comparing the learning curves from both figures. In the fine-tuning phase, the early stopping patience parameter was reduced from 3 to 2, allowing the training process to halt promptly when the model began to exhibit instability. During fine-tuning, there was a slight fluctuation in the second epoch, where the validation loss increased by 0.004 and validation accuracy dropped by 2%, the model quickly recovered by epoch 3. This temporary dip had minimal impact on the overall performance and stability of the model. Overall, the fine-tuning model makes the model more stable and the analysis results more consistent and efficient.

5.2.5 Evaluation of CNN-Bi LSTM Hate Model

5.2.5.1 Evaluation Result of CNN-Bi LSTM Hyperparameter Tuning

```
📌 Train Accuracy: 0.8129
📌 Validation Accuracy: 0.7887

📌 Weighted F1 Score: 0.7932
📌 Micro F1 Score: 0.7887
📌 Macro F1 Score: 0.7665

📌 Precision: 0.6368
📌 Recall: 0.7635

📌 ROC AUC: 0.8516
```

Figure 166: Evaluation Result of CNN-Bi LSTM Hyperparameter Tuning

The result of hyperparameter tuning is that the model achieves a validation accuracy of 78.87%, a weighted F1 score of 79.32%, a binary precision of 63.68%, and a binary recall of 76.35%. The ROC-AUC score of 85%.

Classification Report

```
📌 Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.80      0.84     16880
           1       0.64      0.76      0.69      7741

    accuracy                           0.79     24621
   macro avg       0.76      0.78      0.77     24621
weighted avg       0.80      0.79      0.79     24621
```

Figure 167: CNN-Bi LSTM Hyperparameter Tuning Classification Report

The classification report shows that the model achieved the accuracy of 79% and macro F1 score of 77%. The model performed well in detecting non-hate speech, with a precision of 88% and recall of 80%. For hate speech, recall is 76%, and the lower precision of 64%, indicating the high false positive rate.

Confusion Matrix

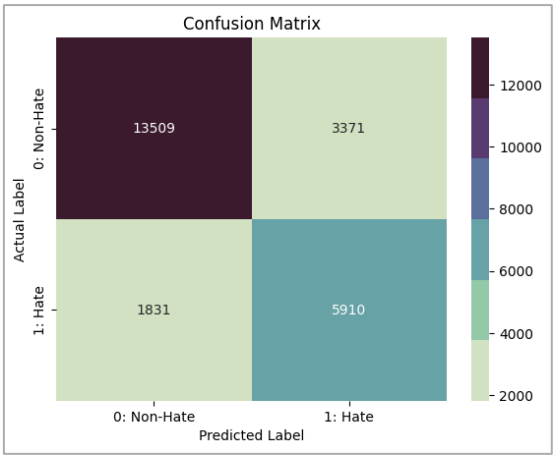


Figure 168: CNN-Bi LSTM Hyperparameter Tuning Confusion Matrix

CNN Bi-LSTM hybrid model correctly classified 13509 rows of data are non-hate speech and 5910 rows of hate. The false positive rate is 19.97% and the false negative rate is 23.65%.

Learning Curve

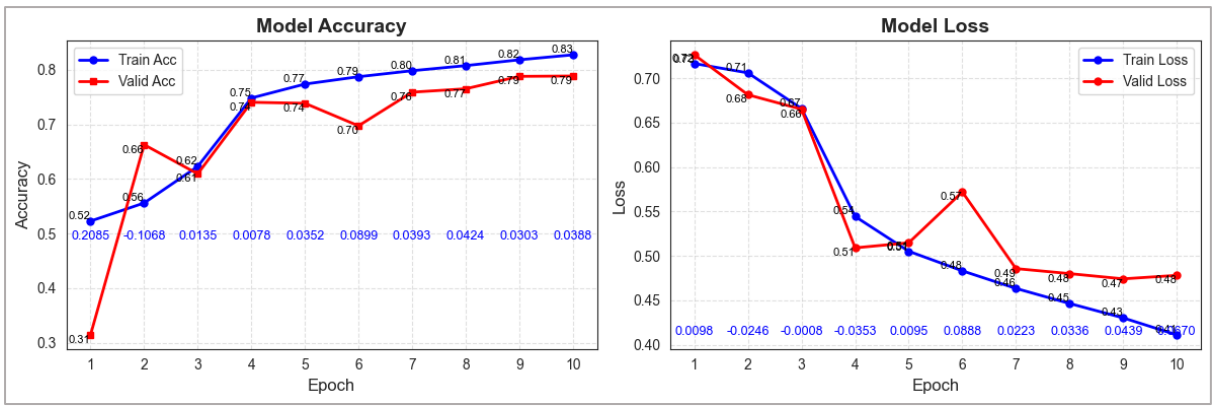


Figure 169: CNN-Bi LSTM Hyperparameter Tuning Learning Curve

This learning curve shows that before the 7th epoch, the training and validation accuracy and loss fluctuated greatly. After the 7th epoch, the model began to stabilize and slightly overfitted. As the epochs increased, the valid loss began to increase slowly. Although the hyperparameter tuning found the best parameters but caused the model to overfit slightly.

ROC and Precision-Recall Curve

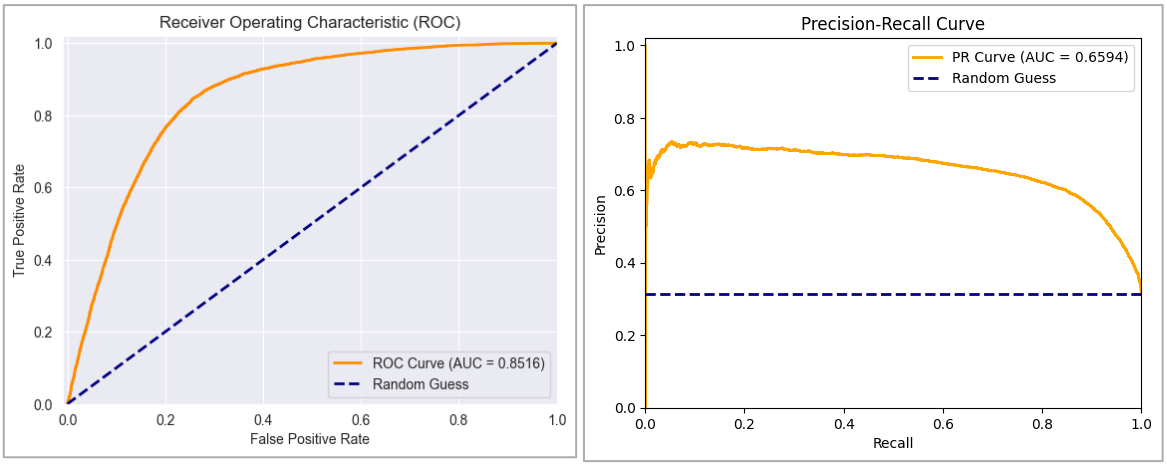


Figure 170: CNN-Bi LSTM Hyperparameter Tuning ROC Curve (Left) and PR Curve (Right)

The model performs well overall, but in PR-AUC score is 65% means that performs poorly when detecting minority classes (hate labels), proving that the overall performance of the model depends on non-hate speech.

5.2.5.2 Evaluation Result of CNN-BiLSTM Fine Tuning

```
Train Accuracy: 0.8057
Validation Accuracy: 0.7691

Weighted F1 Score: 0.7766
Micro F1 Score: 0.7691
Macro F1 Score: 0.7537

Precision: 0.5959
Recall: 0.8255

ROC AUC: 0.8414
```

Figure 171: CNN-Bi LSTM Fine Tuning Evaluation

The fine-tuning model achieved a train accuracy of 80.57% and validation accuracy of 76.91%. The weighted F1 score is 77.66%, showing balanced overall performance, while the macro F1 score of 75.37%, binary precision is 59.59%, recall is 82.55%, meaning the model captures most actual positive cases but at the cost of false positives. The ROC AUC of 84.14% suggests that the model has a good overall model performance.

Classification Report

```
📌 Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.74      0.82     16880
           1       0.60      0.83      0.69      7741

    accuracy                           0.77     24621
   macro avg       0.75      0.78      0.75     24621
weighted avg       0.81      0.77      0.78     24621
```

Figure 172: CNN-Bi LSTM Fine-Tuning Classification Report

From this classification report, the recall of category 1 (hate speech) is 83% and the precision is 60%, which shows that the model effectively captures most instances of hate speech, but at the cost of a high false positive rate. In hate speech detection, the recall of category 1 (hate speech) is more critical, model focus on improving the recall. However, the low precision (60%) means that a large number of non-hate speech samples are incorrectly classified as hate speech and causing the false positive rate to be high. Overall, the validation accuracy is 77%, which shows a moderate level of performance.

Confusion Matrix

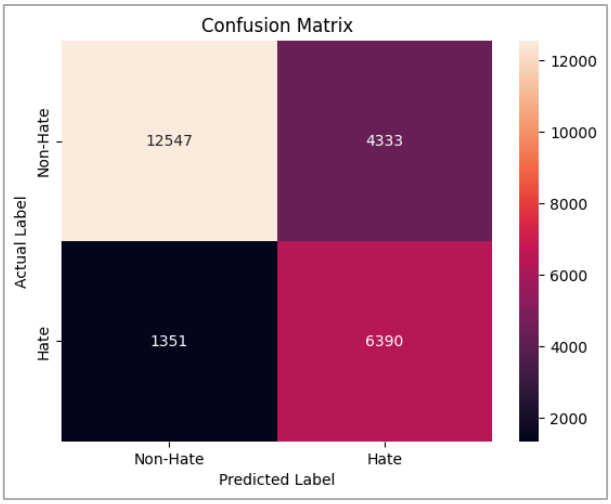


Figure 173: CNN-Bi LSTM Fine Tuning Confusion Matrix

The model correct classification 12547 non-hate speech data and 6390 hate speech data. False positive rate is 25.67% and 17.45% of false negative rate. The model has a strong recall for hate speech detection, resulting in a relatively high number of false positives.

Learning Curve

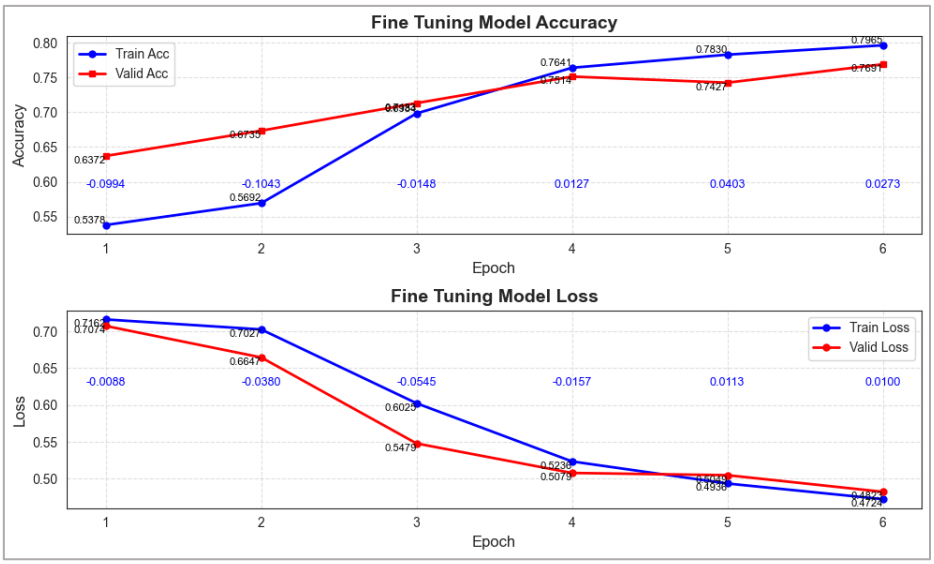


Figure 174: CNN-Bi LSTM Fine-Tuning Learning Curve

The fine-tuning learning curve shows that the train and valid accuracy continue to improve, reaching 79.6% and 76.9% respectively at the sixth epoch, with a difference of 2.7% in training and validation accuracy. The loss curve shows that both training and validation loss are stable decreasing without fluctuations, indicating that the model is stable with low noise and no overfitting.

ROC Curve and Precision-Recall Curve

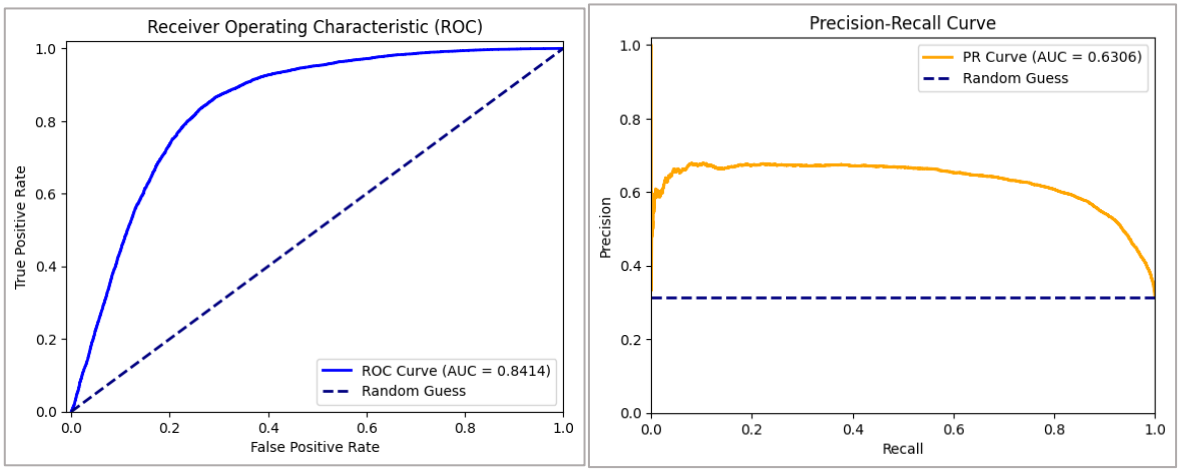


Figure 175: CNN-Bi LSTM Fine Tuning ROC Curve (Left) and PR Curve (Right)

The ROC curve shows a strong classification performance with an AUC of 0.8414. However, the Precision-Recall (PR) AUC is 0.6306, which suggests that precision drops as recall increases. This highlighted the challenge of maintaining a low false positive rate while ensuring high recall in hate speech detection.

5.2.5.3 Comparison Between CNN-Bi LSTM Hyperparameter Tuning and Fine-Tuning

Table 34: Comparison Between CNN-Bi LSTM Hyperparameter Tuning and Fine-Tuning Model

Model / Evaluation Matrix	Hyperparameter Tuning	Fine-Tuning
Train Accuracy	0.8129	0.8057
Validation Accuracy	0.7887	0.7691
Weighted F1 Score	0.7932	0.7766
Micro F1 Score	0.7887	0.7691
Macro F1 Score	0.7665	0.7537
Binary Precision	0.6368	0.5959
Binary Recall	0.7635	0.8255
ROC-AUC Score	0.8516	0.8414
PR-AUC Score	0.8516	0.8414
FPR	19.97%	25.67%
FNR	23.65%	17.45%

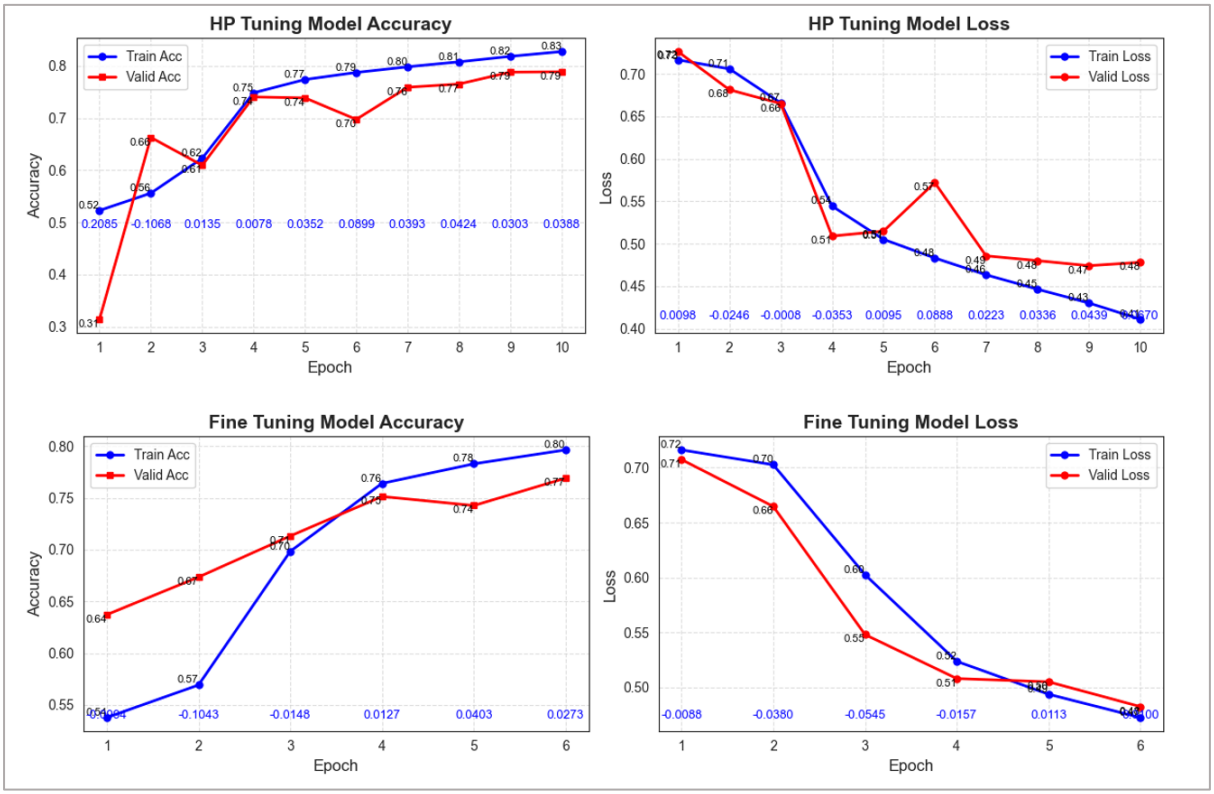


Figure 176: CNN-Bi LSTM Fine Tuning Learning Curve Compare with Hyperparameter Tuning

From this comparison table, that most of the metrics for hyperparameter tuning outperform fine tuning. The main difference between the two models is that fine tuning achieves a better recall for class 1, but at the cost of a higher false positive rate (FPR). Overall, the model performance suggests that hyperparameter tuning is superior to fine tuning. However, when comparing the learning curves, fine tuning shows a clear advantage, fine tuning model are more stable and the valid loss continues to decrease, indicating a better generalization. Although the validation accuracy, F1 score, and PR-AUC score of fine-tuning are not as good as those of the hyperparameter tuning model, choose fine-tuning because the learning curve of fine-tuning is more stable, the valid loss continues to decrease, and the higher recall is more important for class 1 in hate speech detection.

5.2.6 Evaluation of Bi-LSTM Integration "SenticNet" Concept-Level Sentiment Comparison between BiLSTM-SenticNet, BiLSTM-FastText-SenticNet and BiLSTM-FastText-Focal Loss-SenticNet Model

Table 35: Comparison of Three Feature-Level Fusion Models

Feature-Level Fusion Model / Evaluation Matrix	BiLSTM-SenticNet	BiLSTM-FastText-SenticNet	BiLSTM-FastText-Focal Loss-SenticNet
Train Accuracy	**0.8791**	0.8180	0.8375
Validation Accuracy	0.7745	**0.7903**	0.7871
Weighted F1 Score	0.7822	**0.7967**	0.7944
Micro F1 Score	0.7745	**0.7903**	0.7871
Macro F1 Score	0.7621	0.7746	**0.7751**
Binary Precision	0.5970	**0.6239**	0.6116
Binary Recall	0.8698	0.8380	**0.8846**
ROC-AUC Score	0.8705	0.8758	**0.8878**
PR-AUC Score	0.7363	0.7366	**0.7725**

Table 36: Compare Classification Report of Three Feature-Level Fusion Models

	Classification Report
BiLSTM-SenticNet	Classification Report: precision recall f1-score support 0 0.92 0.73 0.82 16880 1 0.60 0.87 0.71 7741 accuracy 0.77 24621 macro avg 0.76 0.80 0.76 24621 weighted avg 0.82 0.77 0.78 24621
BiLSTM-FastText-SenticNet	Classification Report: precision recall f1-score support 0 0.91 0.77 0.83 16880 1 0.62 0.84 0.72 7741 accuracy 0.79 24621 macro avg 0.77 0.80 0.77 24621 weighted avg 0.82 0.79 0.80 24621
BiLSTM-FastText-Focal Loss-SenticNet	Classification Report: precision recall f1-score support 0 0.93 0.74 0.83 16880 1 0.61 0.88 0.72 7741 accuracy 0.79 24621 macro avg 0.77 0.81 0.78 24621 weighted avg 0.83 0.79 0.79 24621

These two tables compare three variants of BiLSTM-based models for hate speech detection, each incorporating different combinations of SenticNet, FastText-SenticNet and FastText-Focal Loss-SenticNet.

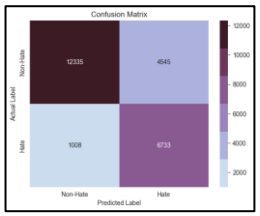
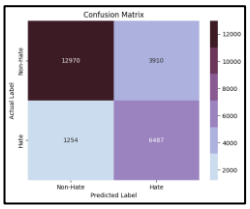
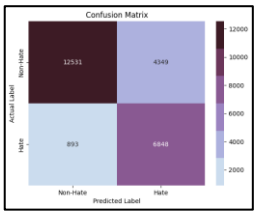
The **BiLSTM-SenticNet model** has a training accuracy of 87.91% and validation accuracy of 77.45%. This model has 10% gap between the training and validation accuracy, indicating overfitting. Although the BiLSTM-SenticNet model has a high binary recall of 86.98%, but the binary precision only 59.70% indicating higher false positives rate and this model is not suitable for actual deployment.

The **BiLSTM-FastText-SenticNet model** shows better generalization ability, with a higher validation accuracy of 79.03% and weighted F1 of 79.67%, and precision of 2.69% (62.39%) compared to the BiLSTM-SenticNet model. The ROC-AUC score and PR-AUC score are not much different, both maintained at 87% and 73%, indicating that incorporating fastText as the

word embedding method improves the model's ability to distinguish between classes compared to a randomly initialized trainable embedding.

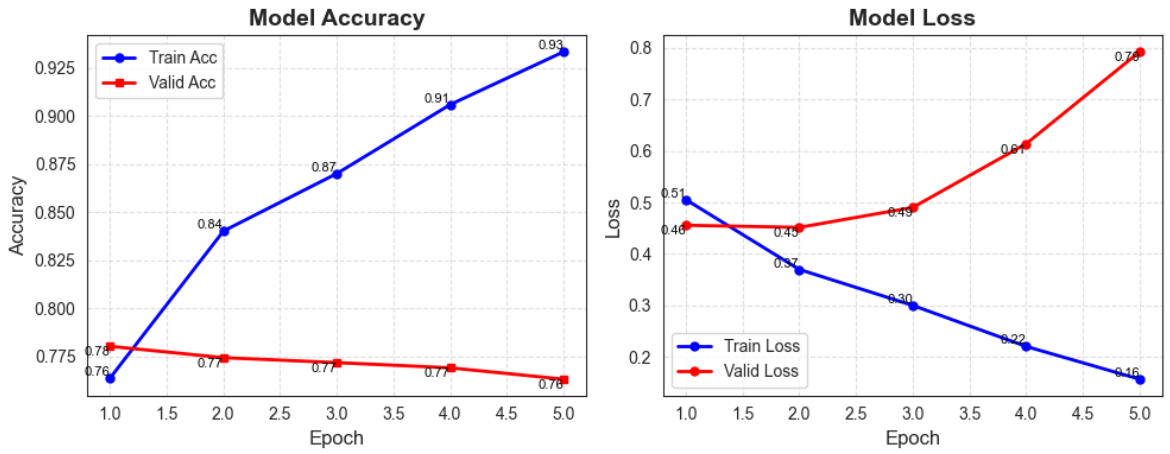
The **BiLSTM-FastText-Focal Loss-SenticNet model** performs better than another two model, although the validation accuracy and f1-score drop slightly to 78.71% and 79.44%. But this model has the highest recall rate of 88.46%, ROC-AUC score of 88.78% and PR-AUC score of 77.25%, indicating that the model has a strong ability to identify true hate speech text. In scenarios where minimizing false positives is critical, the BiLSTM-FastText-SenticNet model is preferable, but using Focal Loss is the best choice for hate speech detection because it can maximize the detection of hate speech, and the validation accuracy is still within an acceptable range.

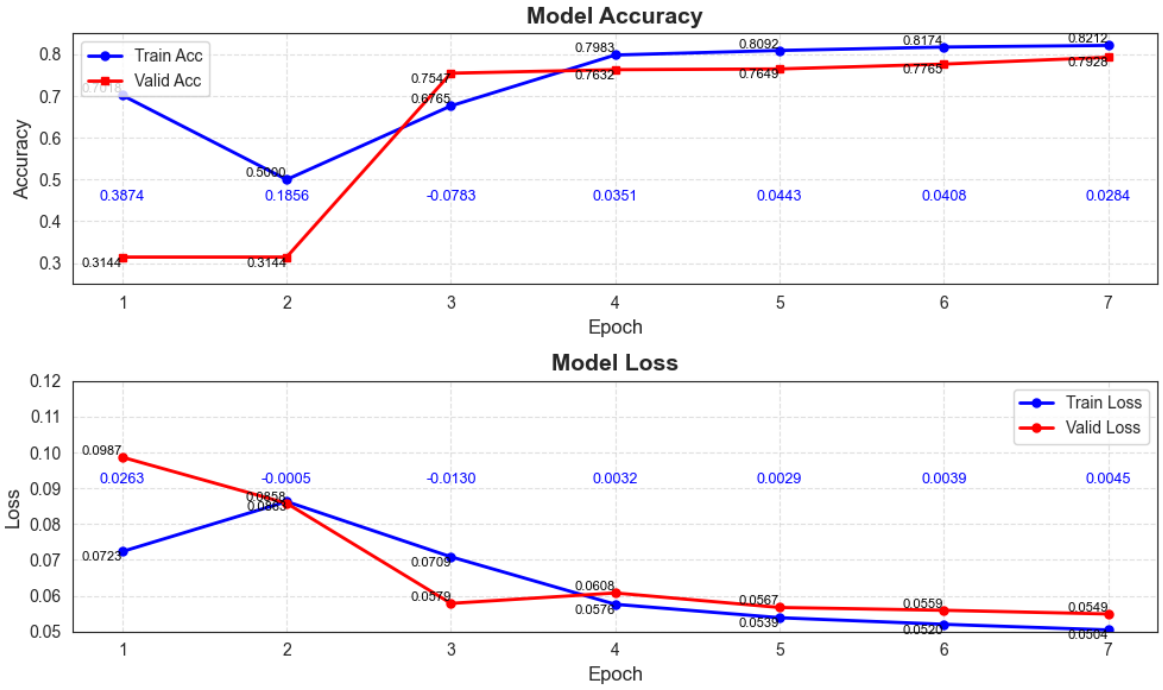
Table 37: Compare Confusion Matrix of Three Feature-Level Fusion Models

Feature-Level Fusion Model / Evaluation Matrix	BiLSTM-SenticNet	BiLSTM-FastText-SenticNet	BiLSTM-FastText-Focal Loss-SenticNet
Confusion Matrix			
FPR	26.92%	**23.16%**	25.76%
FNR	13.02%	16.20%	**11.54%**

From the perspective of this confusion matrix, BiLSTM-SenticNet is still the least suitable for detecting hate speech because this model has the highest false positive rate and the false negative rate is not the lowest. The BiLSTM-FastText-SenticNet model and the BiLSTM-FastText-Focal Loss-SenticNet model are similar, correctly detecting 19457 (true positive plus true negative) and 19379 data samples respectively, but BiLSTM-FastText-FocalLoss-SenticNet model is stronger in detecting hate speech because this model has the lowest false negative rate 11.54%.

Table 38: Compare Learning Curve of Three Feature-Level Fusion Models

	Learning Curve
BiLSTM-SenticNet	
BiLSTM-FastText-SenticNet	

BiLSTM-FastText-Focal Loss-SenticNet	

From the learning curves of the three models, the first BiLSTM-SenticNet model has a serious overfit problem. The validation accuracy starts to go down from the second epoch, and the validation loss starts to rise rapidly from the third epoch.

The second BiLSTM-FastText-SenticNet model does not have an obvious overfit phenomenon, but the learning curve is greatly affected by noise, and the curve fluctuates greatly, sometimes rising and sometimes falling.

The third BiLSTM-FastText-Focal Loss-SenticNet model was not stable in the first three epochs. From epoch 4, the model gradually stabilized and the validation accuracy slowly increased to 79.28%, and the validation loss finally dropped to 0.0549. The Focal Loss version dynamically adjusts the sample weights, reduces the contribution of easy-to-classify samples, focuses on difficult samples, and offsets the impact of some noise on the model, thereby showing a more stable convergence. The train and validation accuracy in epoch 7 gap is 2.89%, indicating that the model is not overfitting.

After comparing all the classification reports, evaluation matrix, confusion matrix and learning curve, the third BiLSTM-FastText-Focal Loss-SenticNet model performed the best and most stable.

5.2.7 Evaluation of Multi-task Model

The following section presents a comparison between two results obtained using the same multi-task learning (MTL) model. The first result is based on an imbalanced validation set, where the original label distribution is preserved. The second result is derived from a balanced validation set, where the label distribution has been equalized. This comparison aims to evaluate how validation set balancing impacts the model's performance under the same architecture and training conditions.

Table 39: MTL Model Comparison Between Balanced and Imbalanced Validation Set

	MTL Model 1: imbalanced validation set		**MTL Model 2:** balanced validation set	
	hate result	**emotion result**	**hate result**	**emotion result**
Train Accuracy	94.15%	97.67%	94.13%	98.06%
Validation Accuracy	63.83%	97.37%	91.67%	97.29%
Weighted F1 Score	0.6353	0.9738	0.9167	0.9729
Micro F1 Score	0.6383	0.9737	0.9167	0.9729
Macro F1 Score	0.5737	0.9646	0.9167	0.9729
Binary Precision	0.4202	weighted precision: 0.97	0.9066	weighted precision: 0.97
Binary Recall	0.3961	weighted recall: 0.97	0.9292	weighted recall: 0.97
ROC-AUC Score	0.6595	Three classes ROC-AUC score: 0.99	0.9785	Three classes ROC-AUC score: 0.99
PR-AUC Score	0.3143	-	0.3148	-

The result shown in the table shows a substantial performance gap between the results of the imbalanced and balanced validation sets under the same MTL model. For the hate speech task, the validation accuracy increased from 63.83% to 91.67% after balancing the validation set, with similar improvements observed across all F1 scores (weighted F1 increased from 0.6353 to 0.9167, macro F1 from 0.5737 to 0.9167, and ROC-AUC from 0.6595 to 0.9785). Binary precision and recall also showed large improvements, rising from 0.4202 and 0.3961 to 0.9066 and 0.9292.

Although using a balanced validation set for testing can improve accuracy, this result does not reflect the actual performance of the model in real scenarios. The PR-AUC score of 0.3148 (MTL Model 2) shows that a forced balanced validation set will cause the evaluation score to be falsely high, making it difficult for the model to handle unbalanced data in reality. In contrast, an unbalanced validation set can more realistically and reliably reflect the performance of the model under the actual data distribution.

	Classification Report & Confusion Matrix
MTL Model 1 (Hate Result): imbalanced validation set	📌 Classification Report: precision recall f1-score support 0 0.73 0.75 0.74 16880 1 0.42 0.40 0.41 7741 accuracy 0.64 24621 macro avg 0.58 0.57 0.57 24621 weighted avg 0.63 0.64 0.64 24621 📌 Confusion Matrix: [[12650 4230] [4675 3066]]
MTL Model 2 (Hate Result): balanced validation set	📌 Classification Report: precision recall f1-score support 0 0.93 0.90 0.92 7740 1 0.91 0.93 0.92 7740 accuracy 0.92 15480 macro avg 0.92 0.92 0.92 15480 weighted avg 0.92 0.92 0.92 15480 📌 Confusion Matrix: [[6999 741] [548 7192]]

From the confusion matrix of the model evaluated on the imbalanced validation set, the false positive rate (FPR) is 25%, while the false negative rate (FNR) is 60%. There are 60% of the actual hate speech instances that were incorrectly classified as non-hate, indicating a serious issue with the model's ability to identify hate speech samples.

Challenges of Multi-task Models

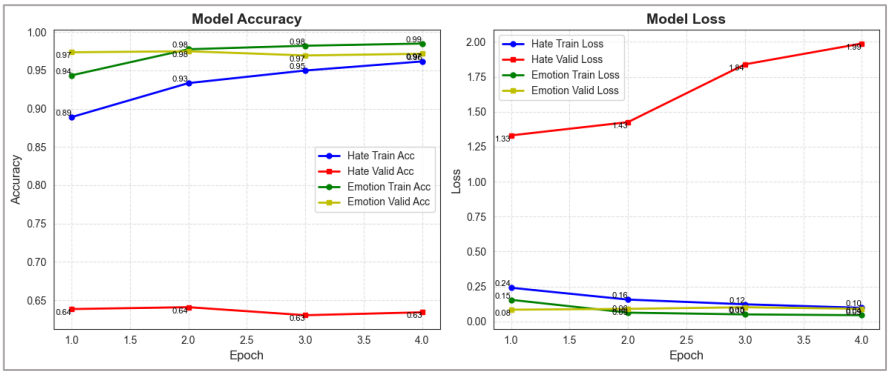


Figure 177: MTL Model 1- Imbalanced Validation Set Learning Curve

From this table MTL Model Comparison and the graph of the learning curve, it is obvious that in addition to the dataset imbalance problem, there is also a gradient conflict problem. Gradient conflict can be identified by observing the validation loss curves of hate and emotion. The validation loss of emotion has been decreasing while the loss of hate speech has been increasing. This shows that the gradient directions of hate and emotion tasks are opposite. When the model optimizes one task, the other task will be negatively affected. In other words, there is a lack of correlation between the two datasets, and the emotion task is more dominant. The model will prioritize optimizing emotion, resulting in good performance of emotion prediction on the imbalance validation set, while poor performance on the hate task.

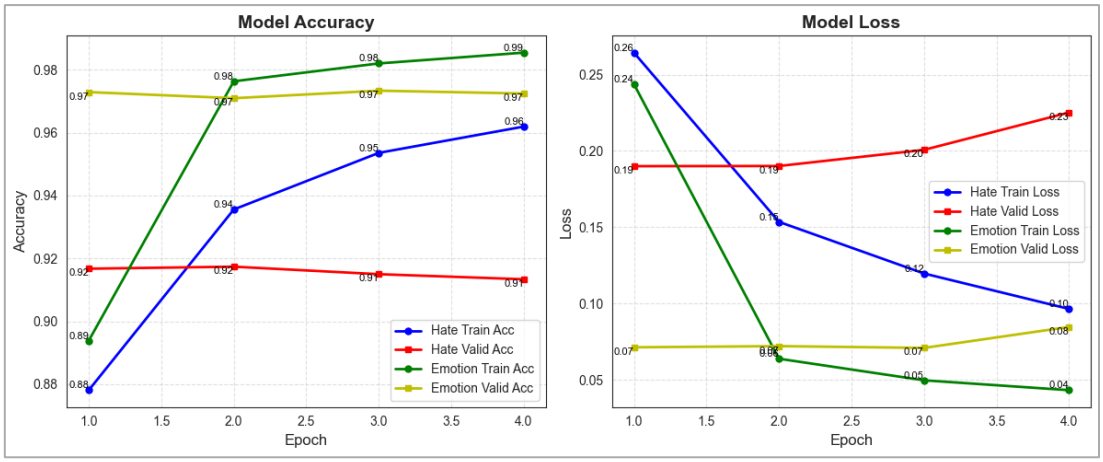


Figure 178: MTL Model 2- Balanced Validation Set Learning Curve

From the learning curve of model 2, showed that balanced validation set makes the accuracy look good, the validation curve reveals the problem. The validation accuracy of the model has not actually improved at all, and the validation loss has slowly increased. This is a typical overfitting performance. The training accuracy continues to rise, but the validation set has made no improvement. This shows that the model is primarily memorizing the training data rather than learning meaningful or generalizable patterns. So, seemingly high accuracy on a balanced validation set might lead one to believe that the model is performing well, when in reality, the model does not perform well on unseen data.

5.2.8 Comparison of All Models' Performance

Table 40: Comparison of All Model Performances

Model / Evaluation Matrix	XGBoost Base Model	Bi-LSTM Fine Tuning	Bi-GRU Fine Tuning	CNN-BiLSTM Fine Tuning	BiLSTM-SenticNet	MTL Model 1: imbalanced validation set
	-	FastText Focal Loss ReduceLROnPlateau	FastText Focal Loss ReduceLROnPlateau	FastText ReduceLROnPlateau	FastText Focal Loss ReduceLROnPlateau	FastText Word2Vec
Train Accuracy	0.8006	0.8371	0.8283	0.8057	0.8375	**0.9415**
Validation Accuracy	0.7821	0.7969	**0.7993**	0.7691	0.7871	0.6383
Weighted F1 Score	0.7855	0.8030	**0.8054**	0.7766	0.7944	0.6353
Micro F1 Score	0.7821	0.7969	**0.7993**	0.7691	0.7871	0.6383
Macro F1 Score	0.7560	0.7813	**0.7841**	0.7537	0.7751	0.5737

Binary Precision	**0.6362**	0.6330	0.6353	0.5959	0.6116	0.4202
Binary Recall	0.7214	0.8425	0.8490	0.8255	**0.8846**	0.3961
ROC-AUC Score	0.8394	0.8845	0.8865	0.8414	**0.8878**	0.6595
PR-AUC Score	0.5468	0.7558	0.7627	**0.8414**	0.7725	0.3143

In this project, six distinct models were built, including the XGBoost machine learning model and deep learning models, such as Bi-LSTM, Bi-GRU, the CNN-BiLSTM hybrid model, the Bi-LSTM-SenticNet concept-level fusion model and the Bi-LSTM multi-task model (integrated with emotion dataset). The evaluation focused on five key metrics: PR-AUC score, weighted F1-score, validation accuracy, binary recall, and binary precision. Although the XGBoost model has good results on all evaluation metrics, it has a very low PR-AUC score of 0.5468, which indicates that it performs poorly in minority class recognition and is therefore not suitable for deployment in real applications.

In addition, by comparing the baseline hate speech models (Bi-GRU and Bi-LSTM) with the Bi-LSTM multi-task model and the Bi-LSTM-SenticNet concept-level fusion model, it can be found that the concept-level fusion method faces fewer challenges than the multi-task learning method. The multi-task model faces problems, including low correlation between tasks (dataset) and gradient conflicts, resulting in poor model performance, with most evaluation scores below 50% and a PR-AUC score of only 31%. In contrast, the concept-level fusion model (Bi-LSTM-SenticNet) is relatively simple and less challenging because the concept-level fusion model uses the same hate speech dataset for word vector construction and emotion concept-level analysis, and the concept-level fusion model only has one output, so the model does not face the challenge of gradient conflicts.

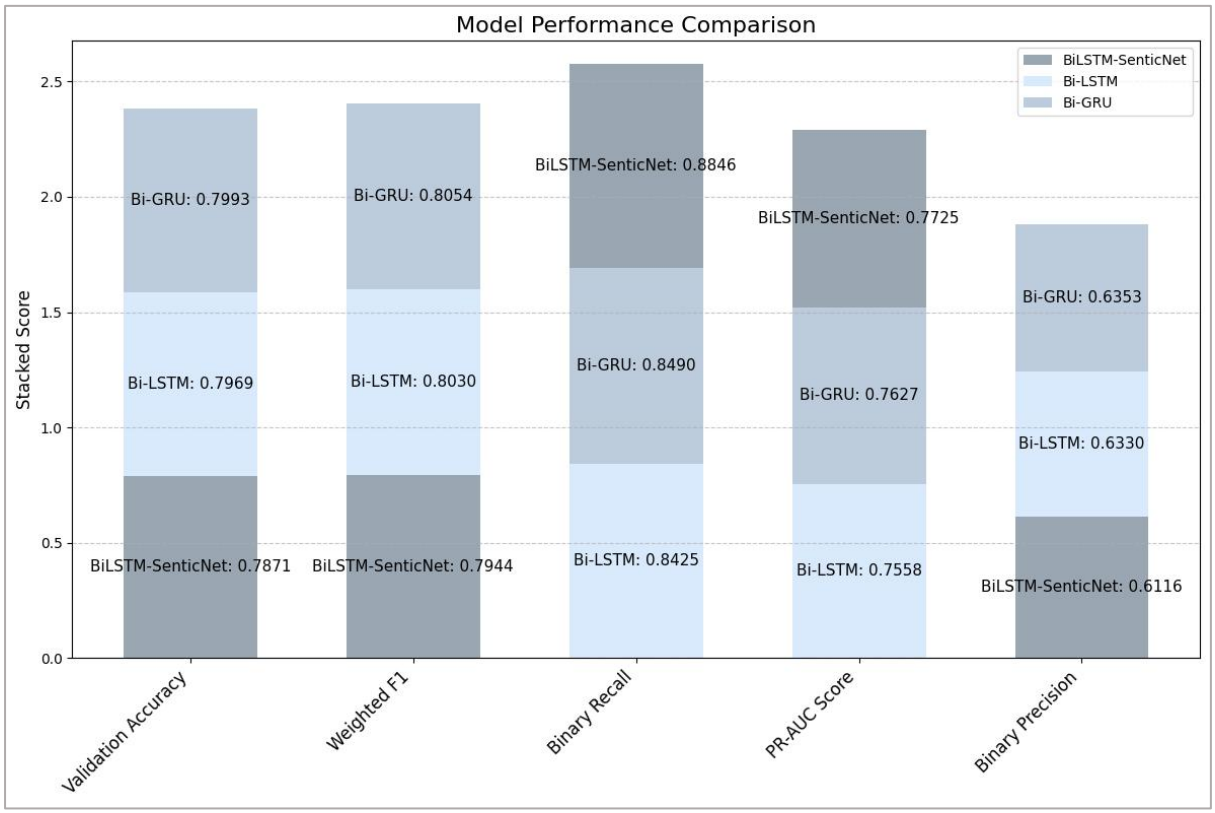


Figure 179: Top 3 Best Model Performance Bar Chart

Based on the comparison table, the top three performing models were identified, which is Bi-GRU, BiLSTM-SenticNet and Bi-LSTM. Among all the baselines hate speech models, Bi-GRU showed the best overall performance. The BiLSTM-SenticNet model performs at a similar level to the Bi-GRU model but is better at detecting hateful tags, with the highest binary recall of 88% and PR-AUC score of 77%. However, this model faced a higher false positive rate problem, with the lowest binary precision of 61% among the three. In contrast, the Bi-GRU model demonstrates a more balanced performance across all metrics and shows greater overall stability. Bi-GRU achieves the highest validation accuracy is 80%, the top weighted F1-score is 81%, and the highest binary precision is 64%. Additionally, Bi-GRU ranks second in both binary recall 85% and PR-AUC score 76%. The Bi-LSTM model ranks third, and its overall performance lags behind the other two models.

In summary, integrating sentiment-based concept-level features can help the model detect hate speech better, and the model will pay more attention to the minority category (hate). The Bi-GRU baseline model without sentiment features also performs well in overall and even better than Bi-LSTM.

5.2.9 Using Unseen Data Set to Evaluate Model in Reality

Table 41: Evaluation on Unseen Dataset

Evaluation Matrix	Result
Accuracy	0.4853
Weighted F1 Score	0.4811
Micro F1 Score	0.4853
Macro F1 Score	0.4811
Binary Precision	0.4875
Binary Recall	0.5743
ROC-AUC Score	0.4782
Classification Report and Confusion Matrix	📌 Classification Report: precision recall f1-score support 0 0.48 0.40 0.43 18791 1 0.49 0.57 0.53 18791 accuracy 0.49 37582 macro avg 0.48 0.49 0.48 37582 weighted avg 0.48 0.49 0.48 37582 📌 Confusion Matrix: [[7446 11345] [8000 10791]]

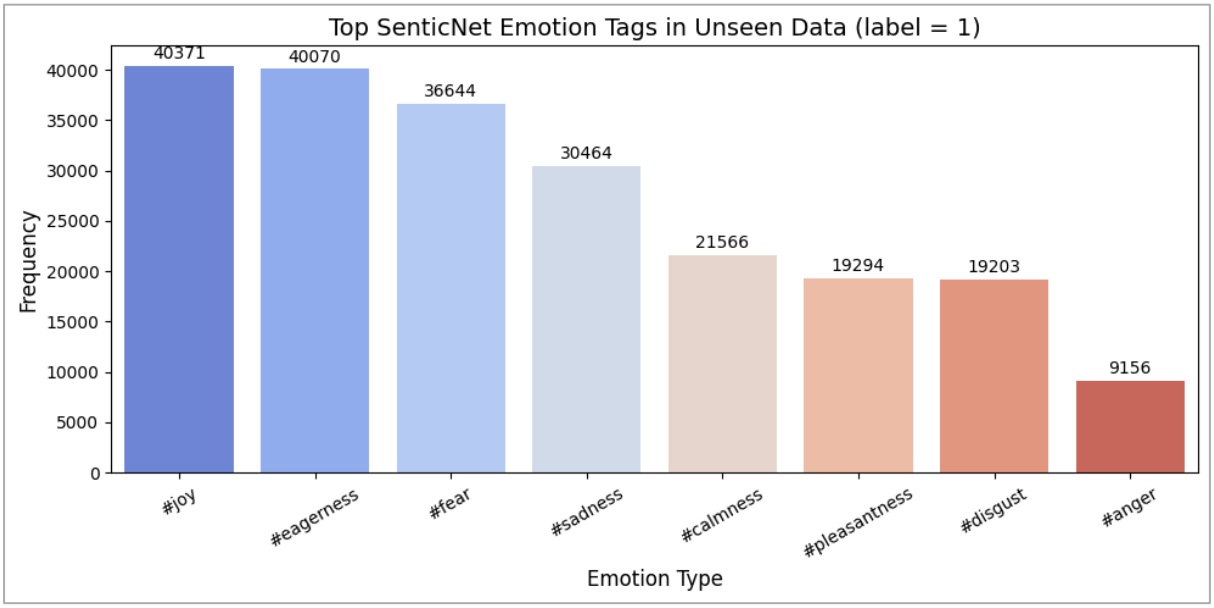


Figure 180: SenticNet Emotion Tags in Unseen Dataset

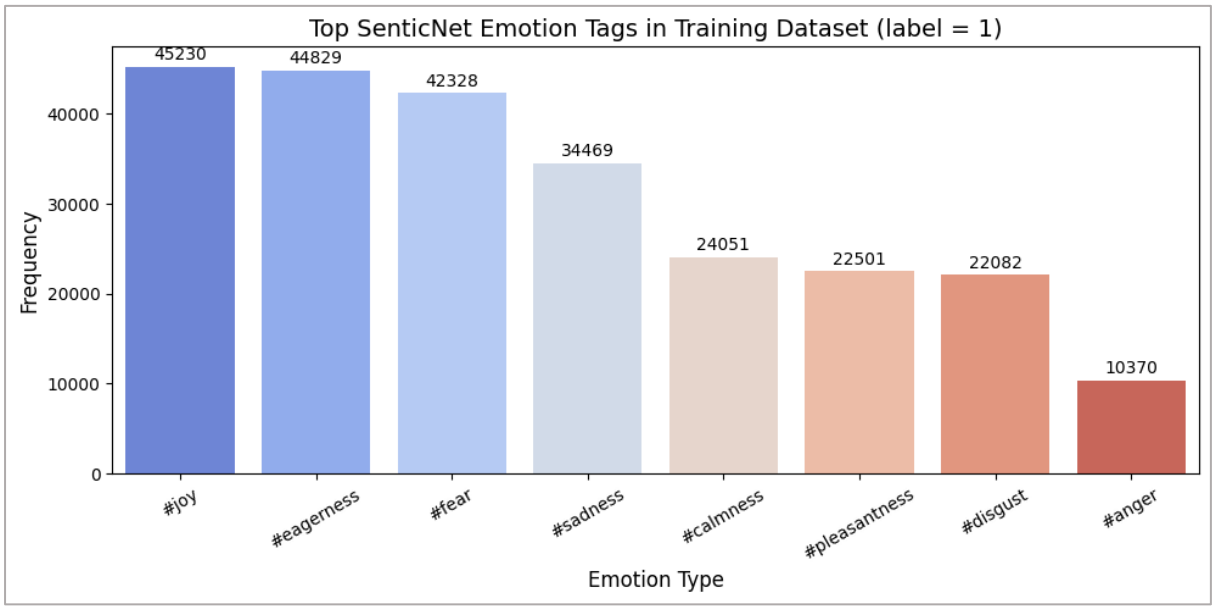


Figure 181: SenticNet Emotion Tags in Training and Validation Dataset

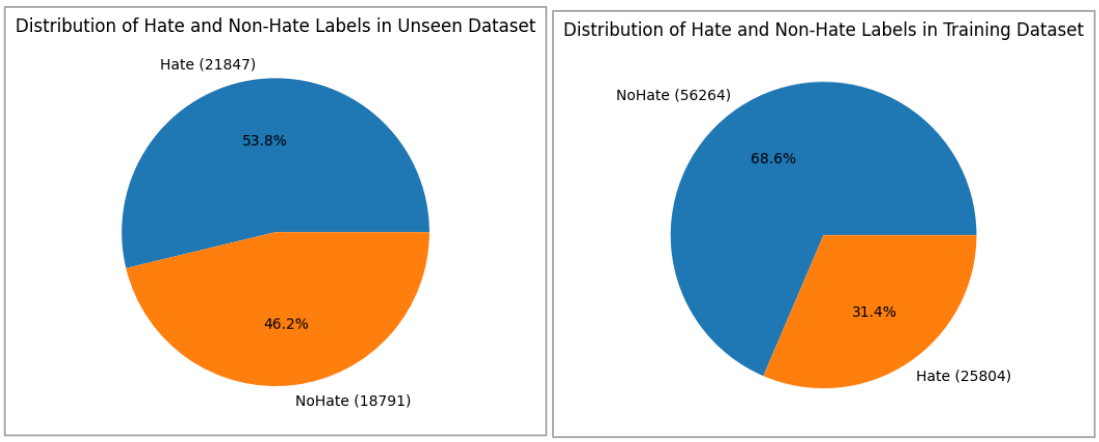


Figure 182: Two Datasets Hate Speech Distribution

The BiLSTM-SenticNet-FastText-Focal Loss model performs poorly on the unseen dataset, with an overall test accuracy of 48.53%, a weighted F1 score of 48.11%, and a ROC-AUC of 47.82%. This shows that the model has problems with generalization. From the two emotion distribution graphs above, we can see that the sentiment of most hate speech words is broken down into joy and desire, followed by fear and sadness. Emotions related to hate speech, such as anger and disgust, are ranked last. This is why the model performs poorly on the real-world unseen dataset, even though the word frequency, bigrams, trigrams, and sentiment distribution of words in the unseen and training datasets are similar. The difference between the two datasets lies in the distribution of hate speech. In the training dataset, non-hate samples account for a large proportion, so the model learns more positive emotions, such as joy and desire. In the hate-labelled samples, although positive emotions still dominate, the proportion of anger is

significantly higher than that of non-hate samples, with 7,664 more words labelled as anger, so the model can learn some hate recognition features from these words. However, in the unseen dataset, the proportion of hate speech is higher than that of non-hate. At the same time, many hate texts are expressed in sarcasm and joking tone and are labelled as positive emotions. Although hate speech accounts for the majority, the number of anger tags for hate and non-hate is almost the same in terms of emotional feature distribution, with a difference of only 145 words. This proves that the expressions of this unseen hate speech dataset are all very implicitly and do not carry angry emotions. Since the model learns the association between joy and non-hate during the training phase, when encountering texts with joy with hate label on unseen data, the model's prediction is prone to deviation.

13427	use fuckin alkie admit several occasion benzos essentially booze pill form alky never take benzos fuck doctor think	0
13428	use fuckin woman admit several occasion transes mess fuck doctor think	1
13429	perfect woman not exist must obey moldable others every single hot girl personality disorder attention soon grow pair boob continue drench attention late 2os earl	1
13430	perfect woman not exist attitude must obey moldable others people think every single hot girl personality disorder attention soon grow pair boob continue drench a	0
13431	perfect woman nonexistent must obey moldable others every single hot girl personality disorder attention soon grow pair boob continue drench attention late 2os e	1
13432	perfect child nonexistent must obey moldable others every single kid different personality trait attention soon grow pair teeth continue drench attention sibling bor	0
13433	perfect woman fantasy must obey moldable others every single hot girl personality disorder attention soon grow pair boob continue drench attention late 2os early	1
13434	perfect woman fantasy must look much look care others hot girl tend toxic attention soon hit puberty continue assault men no wonder many hatred towards u	0
13435	realize people come canada america china literally not spy u technological development country honestly come start respect tyou come legally step american not a	0
13436	anyone see call asian black hispanic not american xenophobic	1
13437	anyone call asian black hispanic not american xenophobic simple	1
13438	anyone call asian black hispanic not support enough people racist simple	0
13439	anyone call asian black hispanic not american xenophobic	1
13440	anyone call asian black hispanic american rest u fuck xenophobic people	0
13441	anyone call asian black hispanic not american simply xenophobic	1
13442	anyone call asian black hispanic also poc american	0
13443	attention need go 0 sleep woman set standard fuck position power fuck never talk slut	1
13444	attention need go 0 sleep man set standard fuck position power fuck never talk fuck loss	0
13445	attention need go 0 sleep woman set standard sex position power sleep never talk slut	1

Figure 183: Partly of Unseen Dataset

The unseen dataset contains many implicitly biased statements, such as “women” and “black people”, which appear more frequently in samples labelled as hate speech. Although these keywords do not inherently carry hateful connotations, their high occurrence in hate labelled data may cause the model to form misleading associations. Therefore, the model is more likely to misclassify sentences containing these words as hate speech, which causes a result of increase in the false positive rate.

5.3 Model Deployment

5.3.1 Hate Speech Predict User Interface

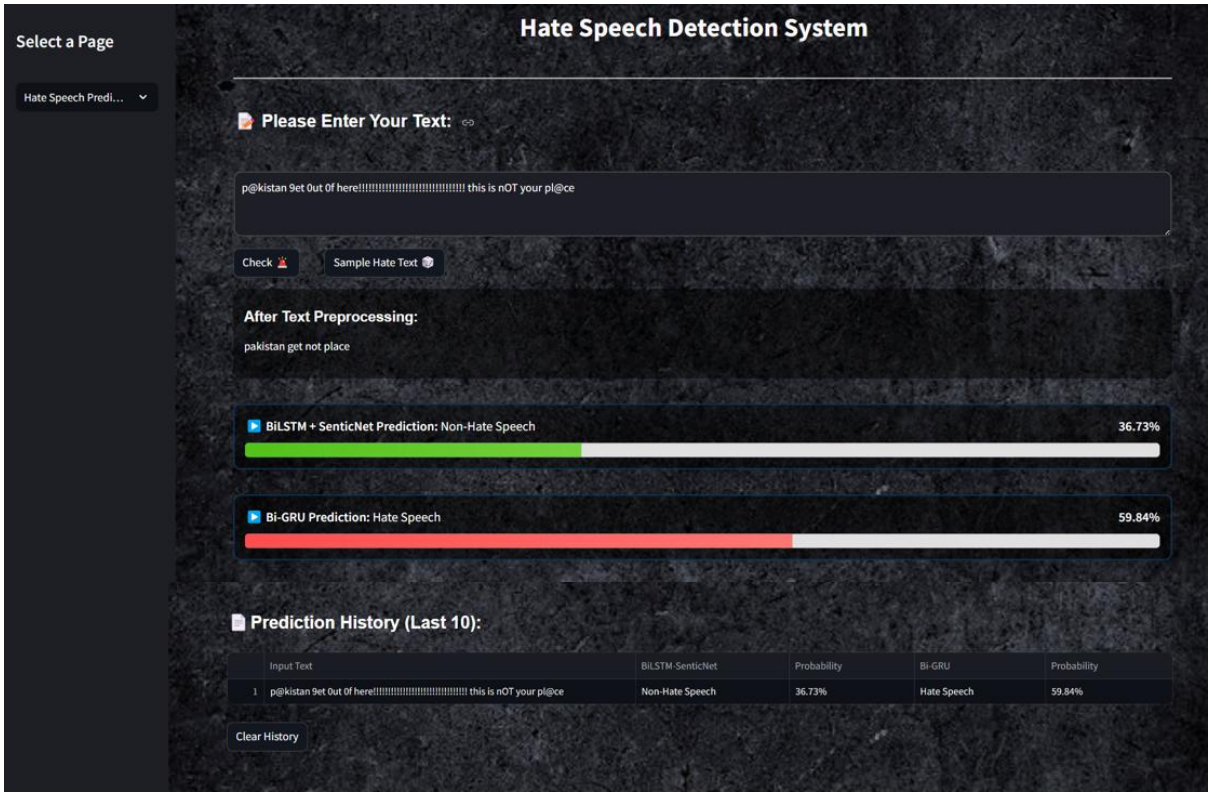


Figure 184: Hate Speech Predict Page

This is the hate speech prediction page, users can either input a sentence manually or click the "Sample Hate Text" button to randomly select a sentence from the unseen dataset. After submission, the system shows the sentence after text preprocessing, removing stop words, and lemmatization. The middle of the page has two progress bars that display the predictions from two models. The top one is integrated with emotion concept-level fusion, while the bottom one is the baseline Bi-GRU hate speech detection model. A green progress bar indicates the input is non-hate, while a red bar indicates the input sentence is hate speech. At the bottom of the page, there is a prediction history table that lists the latest 10 prediction results, allowing users to compare the output differences between the two models.

	Input Text	BiLSTM-SenticNet	Probability	Bi-GRU	Probability
1	you so fking smart.	Non-Hate Speech	26.86%	Non-Hate Speech	28.16%
2	That black dog looks crazy	Hate Speech	78.6%	Hate Speech	70.36%
3	Documentaries often show the devastating effects of disease outbreaks.	Non-Hate Speech	44.15%	Non-Hate Speech	40.54%
4	The mold on the wall looked absolutely disgusting.	Non-Hate Speech	46.89%	Non-Hate Speech	43.61%
5	He called himself trash as a joke after failing the test.	Hate Speech	55.72%	Non-Hate Speech	39.35%
6	I hate waking up early on Mondays	Hate Speech	60.11%	Non-Hate Speech	25.84%

Figure 185: Non-hate speech sentences but including hate speech keyword

These six sentences use keywords related to hate, but those sentences are not hate speech. From this table, can observe that the Bi-LSTM SenticNet model has a higher false positive rate in this case. While the Bi-LSTM SenticNet model benefits in emotion understanding, but it may be more sensitive to language with high emotions rather than specific to hate speech, so resulting in misclassification when using keywords related to hatred in non-hate contexts.

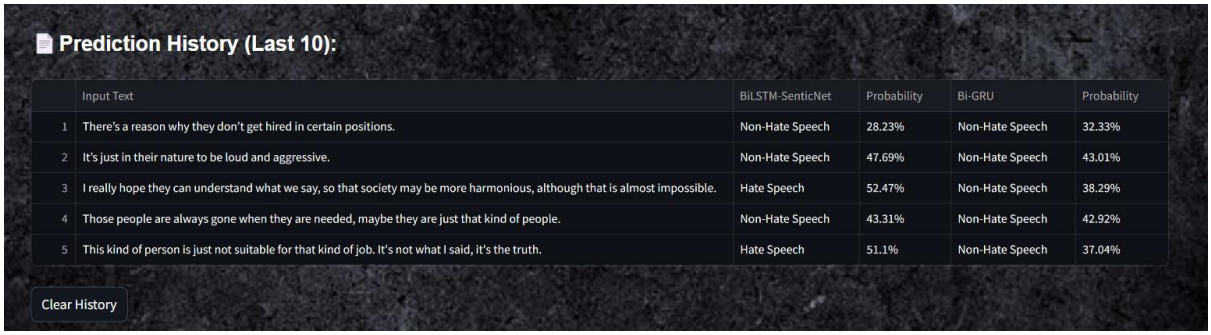


Figure 186: Implicit hate speech, without directly using offensive language

The table shows the test results for five subtle hate speech examples. The results show that the Bi-LSTM SenticNet model generally gives higher prediction scores to these subtle hate speeches than Bi-GRU hate base model. This shows that incorporating emotion understanding helps the model not rely on keyword-based analysis methods. Although the prediction scores have improved slightly, the percentage is still only around 45%, which is still not enough to reliably classify these texts as hate speech.

Table 42: Summary of the Model Deploy Test

Scenario	**Bi-GRU (Hate Speech Keyword-based)**	**Bi-LSTM SenticNet (Emotion + Keyword)**
Non-hate but contains negative/hate speech keywords	Better than Bi-LSTM SenticNet (but if contains keywords like "black", it is easy to misjudge)	High false positive rate (sensitive to angry but non-hate speech sentences)
Subtle hate speech	Weak detection capabilities (due to reliance on hate keywords)	Slightly better, but still limited (can sense emotions, but not accurately enough)

5.3.2 Hate Speech Dataset Dashboard

Figure 187: Hate Speech Dataset Dashboard Page

This dashboard shows that the hate speech dataset is used for training. The visualizations here show the distribution of hate and non-hate labels, word frequency, and the most frequent bigrams in both hate and non-hate texts. Additionally, the top 20 unknown words are presented, comparing the data before and after text preprocessing. This dashboard is clear that before preprocessing, the dataset contained a large number of abbreviations and informal expressions.

5.3.3 Top 3 Model Performances Dashboard

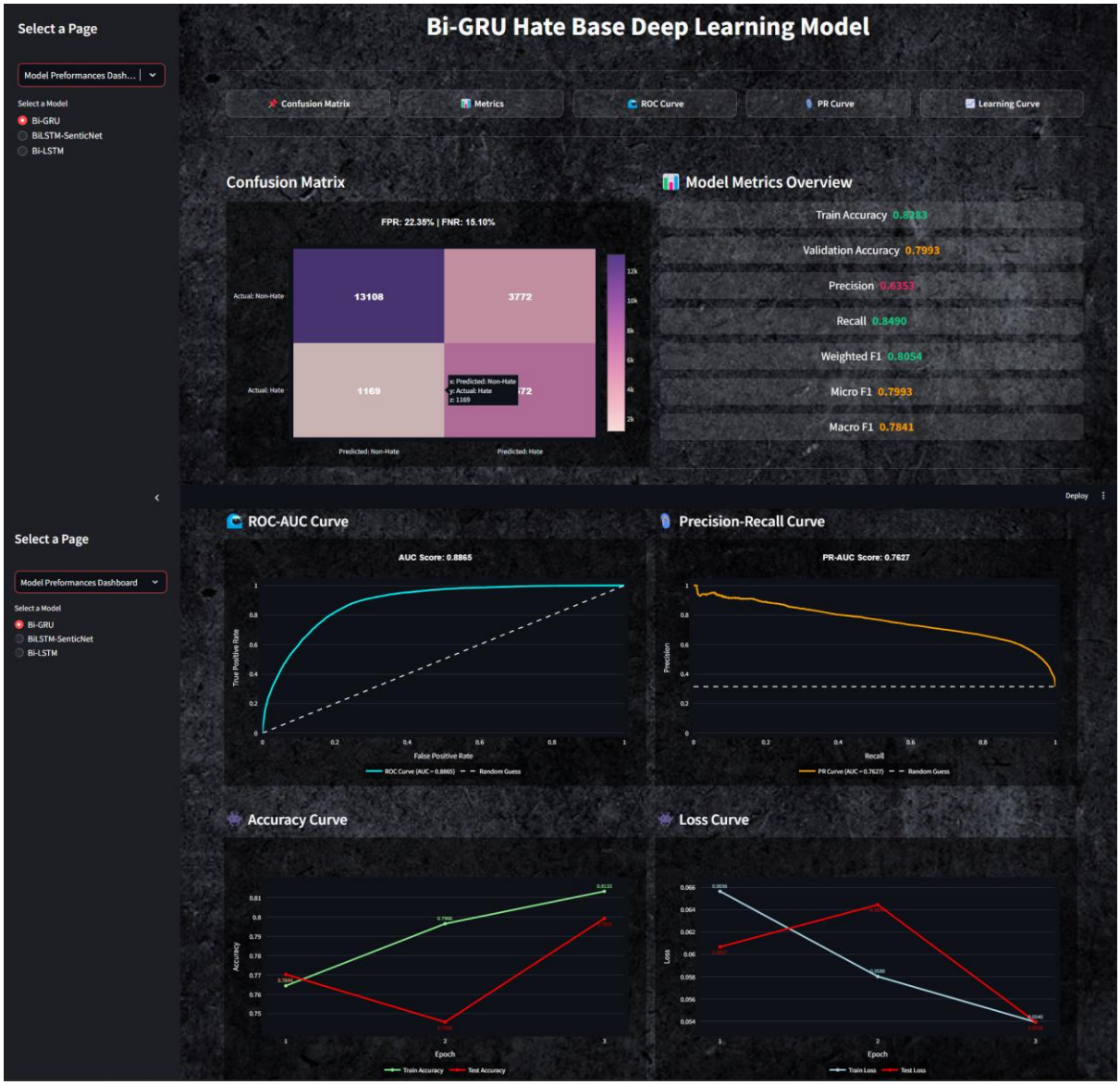


Figure 188: Model Performances Dashboard Page

This page also presents the performance of the top three models, which are the Bi-GRU hate base model, the BiLSTM-SenticNet concept-level fusion model, and the Bi-LSTM hate base model. These three models are showcased on separate pages with a consistent design layout. Each page provides seven evaluation metrics, including train accuracy, validation accuracy, binary precision and recall, weighted, macro, and micro F1 scores, a confusion matrix highlighting false positive and false negative rates, ROC curve, PR curve, and learning curve. All visualizations are available for download as images.

5.4 Summary

To summarize the results, through comparison that adding emotion dataset to the model is not a good choice. It is unhelpful for hate speech and affects the results of hate speech detection. A better choice is to integrate emotion concept-level instead of emotion dataset, which can reduce challenges and capture emotional features, but the cost is that the model will pay more attention to sentences with high emotions even if the sentences are not hate speech and resulting in a higher false positive rate. The performance of Bi-GRU hate base model is similar to that of Bi-LSTM SenticNet model, but the model without emotional features is weaker in detecting subtle hate speech. Since non-hate samples make up the majority of the training set and most of words are accompanied by positive emotions, the model learns the association between positive emotions and non-hate speech. When faced with hate speech expressed in a positive tone in real contexts, such as sarcasm and ridicule, the prediction accuracy drops significantly.

CHAPTER 6: CONCLUSION

6.1 Critical Evaluation

6.1.1 Achievement of the Project

The project lists the key challenges facing current hate speech detection models. First, replacing letters with numeric characters. Second, over-reliance on keyword-based models. Third, serious imbalance in the dataset. To address the problem of number replacement within words, this project implements a special text preprocessing step to replace numbers in alphabetical characters with corresponding letters. Most fine-tuning models use automatic learning rate reduction functions and focal loss functions to alleviate the problem of unstable model performance caused by data imbalance. Recognizing the limitations of keyword-based detection, this project focuses on integrating emotional features to improve model performance.

Next, this project completes the three objectives in the list. First, find a suitable hate speech and emotion dataset for text preprocessing and model training, and then integrate emotion feature into the model to help predict hate speech. In this project are using two methods to integrate emotion features, among which the result of concept-level fusion has 79% validation accuracy, and this model also successfully pays more attention to sentences with high emotions when detecting hate speech. This project also proves that integrating emotion datasets is a big challenge for hate speech detection. Finally, this model was successfully deployed to the local web.

6.1.2 Project's Contribution to the Community/Industry

The project is in line with Sustainable Development Goal 16: Peace, Justice and Strong Institutions. The establishment of hate speech detection further restricts people from making offensive remarks on the Internet at will, maintains fairness and peace online, and reduces unnecessary social conflicts, especially in this era when minors can easily access the internet, making hate speech detection more important.

Especially for social media systems, hate speech detection is critical to maintaining a healthy and inclusive user community. As the influence of these platforms continues to grow, models can be integrated into the review process to filter content, reduce the workload of human reviewers, and reduce attacks on victims or ethnic groups.

6.1.3 Strengths of the Project

The strength of this project is its focus on deep learning technology for hate speech detection. The model uses FastText word vectors to improve the understanding of unseen words. Users often modify spelling or replace sensitive words with numbers to evade detection in the online social media application. Therefore, the text preprocessing process has also been enhanced to detect and replace ambiguous words, such as those that deliberately replace letters in sensitive words with numbers.

In order to better identify subtle and indirect hate speech, the model also incorporates sentiment features, which can capture content with high emotions, such as sarcasm, without explicit hate-related keywords. In addition, this project also conducted in-depth research and provided the comparison of the baseline model that does not contain emotion features, which helps to verify the effectiveness of emotion integration. Lastly, this project also provides a user interface for user input sentences to test the model's performance.

6.2 Limitations of this Project

The limitation of this project is using the emotion concept-level fusion, which allows the model to pay more attention to sentences with high emotions, such as angry sentences. However, the cost is that the false positive rate will be higher because the model detects some sentences that are not necessarily hate speech but rather sentences expressing anger. In addition, this sentiment concept-level fusion model has a generalization issue. Since the proportion of positive emotions is too large during training and the proportion of non-hate labels in the training set is also relatively large, the model will classify sentences with positive emotions as non-hate speech. Another limitation is integrating an emotion dataset. In this project, the emotion dataset and hate speech datasets are not closely related to the hate speech domain is a limitation. In that case, it can lead to another limitation where the multi-task model encounters gradient conflicts during training, causing the model to behave unpredictably, almost like the model making random guesses. Additionally, most of the models used are deep learning-based, which leads to long training times, and those models are only suitable for hate speech detection in English and cannot be extended to multiple languages.

6.3 Recommendation for Future Work

In this project experimented with loss weighting strategies and attention-based mechanisms to balance the hate speech and emotion-related tasks, these methods were insufficient to fully mitigate the interference between tasks. In future work, to solve multi-task model gradient conflicts problem can use Projecting Conflicting Gradients (PCGrad). However, implementing PCGrad within Keras introduces certain compatibility challenges, which may require custom training frameworks. Future model development can be specific to region or language adaptations, allowing the model to handle linguistic and cultural nuances more effectively. To improve the model's ability and to detect implicit or non-keyword-based hate speech, necessary to manually collect hate speech sentences without hate speech keywords, filter the data and labelled the emotion feature on the hate speech dataset, but this is undoubtedly a difficult and time-consuming task.

REFERENCES

Ahmed , Montaser, Mahmoud, & Korany. (2021, October). *Deep Learning Based Antenna Design and Beam-Steering Capabilities for Millimeter-Wave Applications*. https://www.researchgate.net/publication/355672799_Deep_Learning_Based_Antenna_Design_and_Beam-Steering_Capabilities_for_Millimeter-Wave_Applications

aldi-dimara. (2018). *keras-focal-loss/focal_loss.py at master · aldi-dimara/keras-focal-loss*. GitHub. https://github.com/aldi-dimara/keras-focal-loss/blob/master/focal_loss.py

Allen, C., & Hospedales, T. (2019). *Analogies Explained: Towards Understanding Word Embeddings*. https://proceedings.mlr.press/v97/allen19a/allen19a.pdf

Analytics Vidhya. (2025, March 5). *Complete Guide to Parameter Tuning in XGBoost (with codes in Python)*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

Andre Luiz. (2023, January 3). *A methodology is a general strategy that guides activities within a process. A methodology doesn't depend on technologies or tools, and it's not a set of techniques or recipes.* Linkedin.com. https://www.linkedin.com/pulse/methodology-data-science-andre-luiz-coelho-da-silva/

Andrew Tucker. (2014). *Contractions*. https://www.sjsu.edu/writingcenter/docs/handouts/Contractions.pdf

Antoniadis, P. (2022, May 21). *Cross-Validation: K-Fold vs. Leave-One-Out | Baeldung on Computer Science*. Www.baeldung.com. https://www.baeldung.com/cs/cross-validation-k-fold-loo

Arslan, E. (2024, June 25). *Natural language Processing: Deep Dive into Text Preprocessing and Tokenization— Step 2*. Medium; Medium.

https://medium.com/@erhan_arslan/natural-language-processing-deep-dive-into-text-preprocessing-and-tokenization-step-2-b5dcf9520266

Aszemi, N. M., & Dominic, P. D. D. (2019). Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms. *International Journal of Advanced Computer Science and Applications*, *10*(6). https://doi.org/10.14569/ijacsa.2019.0100638

AWS. (n.d.). *What is RNN? - Recurrent Neural Networks Explained - AWS*. Amazon Web Services, Inc. https://aws.amazon.com/what-is/recurrent-neural-network/

Badri, N., Kboubi, F., & Chaibi, A. H. (2022). Combining FastText and Glove Word Embedding for Offensive and Hate speech Text Detection. *Procedia Computer Science*, *207*, 769–778. https://doi.org/10.1016/j.procs.2022.09.132

Baheti, P. (2022, March 8). *12 Types of Neural Networks Activation Functions: How to Choose?* Www.v7labs.com. https://www.v7labs.com/blog/neural-networks-activation-functions

Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization Yoshua Bengio. *Journal of Machine Learning Research*, *13*, 281–305. https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Bhavik Jikadara. (2023). *Emotions Dataset*. Kaggle.com. https://www.kaggle.com/datasets/bhavikjikadara/emotions-dataset/data

Boishakhi, F. T., Shill, P. C., & Alam, Md. G. R. (2021, December 1). *Multi-modal Hate Speech Detection using Machine Learning*. IEEE Xplore. https://doi.org/10.1109/BigData52589.2021.9671955

Bradter, U., Altringham, J. D., Kunin, W. E., Thom, T. J., O'Connell, J., & Benton, T. G. (2022). Variable ranking and selection with random forest for unbalanced data. *Environmental Data Science*, *1*, e30. https://doi.org/10.1017/eds.2022.34

Butwall, M., Ranka, P., & Shah, S. (2019). Python in Field of Data Science: A Review. *International Journal of Computer Applications*, *178*(49), 20–24. https://doi.org/10.5120/ijca2019919404

Changrong Min, Hongfei Lin, Ximing Li, He Zhao, Junyu Lu, Liang Yang, & Bo Xu. (2023). *Finding hate speech with auxiliary emotion detection from self-training multi-label learning perspective*. https://www.sciencedirect.com/science/article/abs/pii/S1566253523001045

Chen, W., Yang, K., Yu, Z., Shi, Y., & Chen, P. (2024). A survey on imbalanced learning: latest research, applications and future directions. *Artificial Intelligence Review*, *57*(6). https://doi.org/10.1007/s10462-024-10759-6

Chetana Kamlaskar, & Abhyankar, A. (2020). *Multimodal System Framework for Feature Level Fusion based on CCA with SVM Classifier*. 1–8. https://doi.org/10.1109/hydcon48903.2020.9242785

Chiril, P., Pamungkas, E. W., Benamara, F., Moriceau, V., & Patti, V. (2021). Emotionally Informed Hate Speech Detection: A Multi-target Perspective. *Cognitive Computation*. https://doi.org/10.1007/s12559-021-09862-5

Cho. (2014). *tf.keras.layers.GRU | TensorFlow v2.13.0*. TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/layers/GRU

Chumbar, S. (2023, September 24). *The CRISP-DM Process: A Comprehensive Guide*. Medium. https://medium.com/@shawn.chumbar/the-crisp-dm-process-a-comprehensive-guide-4d893aecb151

Ciprian Stratulat. (2023). *How to Write Markdown in Jupyter Notebooks | Edlitera*. Edlitera. https://www.edlitera.com/blog/posts/jupyter-markdown-tutorial

Data Mining and the Case for Sampling Solving Business Problems Using SAS ® Enterprise Miner ™ Software. (n.d.). https://sceweb.uhcl.edu/boetticher/ML_DataMining/SAS-

SEMMA.pdf

Data, D. S. meets C. S. (2023, August 22). *DEEP NEURAL NETWORKS: (DNNS) - Data Science meets Cyber Security - Medium*. Medium. https://medium.com/@datasciencemeetscybersecurity/deep-neural-networks-dnns-24e76aa4cb67

De la Peña Sarracén, G. L., & Rosso, P. (2023). Systematic keyword and bias analyses in hate speech detection. *Information Processing & Management*, *60*(5), 103433. https://doi.org/10.1016/j.ipm.2023.103433

Devlin, J., Chang, M.-W., Lee, K., Google, K., & Language, A. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. https://arxiv.org/pdf/1810.04805

Dube, A. (2023, May 10). *Making the NLP Pipeline - Aditya Dube - Medium*. Medium. https://medium.com/@dube.aditya8/making-the-nlp-pipeline-4d928fe92ea2

Dwivedi, R. (2021). *Role of Stakeholders in Project Success: Theoretical Background and Approach*. ResearchGate; www.researchgate.net. https://www.researchgate.net/publication/351638322_Role_of_Stakeholders_in_Project_Success_Theoretical_Background_and_Approach

Dwivedy, V., & Roy, P. K. (2023). Deep feature fusion for hate speech detection: a transfer learning approach. *Multimedia Tools and Applications*. https://doi.org/10.1007/s11042-023-14850-y

Erin Moore, SSGB, LSSBB, FMVA, M.S., & AWS-CCP. (2023, May 18). *10 Reasons Data Scientists Love Jupyter Notebooks*. Www.linkedin.com. https://www.linkedin.com/pulse/10-reasons-data-scientists-love-jupyter-notebooks-erin/

Feng, W., Gou, J., Fan, Z., & Chen, X. (2023). An ensemble machine learning approach for

classification tasks using feature generation. *Connection Science*, *35*(1). https://doi.org/10.1080/09540091.2023.2231168

GeeksforGeeks. (2021, September 18). *XGBoost*. GeeksforGeeks. https://www.geeksforgeeks.org/xgboost/

GeeksforGeeks. (2024, July 19). *Multimodal Fusion in Computer Vision Systems*. GeeksforGeeks. https://www.geeksforgeeks.org/multi-modal-fusion-in-computer-vision-systems/

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. Deeplearningbook.org; MIT Press. https://www.deeplearningbook.org/

google-research/goemotions at master · google-research/google-research. (2024, January). GitHub. https://github.com/google-research/google-research/tree/master/goemotions

Haque, A., & Md Naseef-Ur-Rahman Chowdhury. (2023). Hate Speech Detection in Social Media Using the Ensemble Learning Technique. *INDIGO (University of Illinois at Chicago)*. https://doi.org/10.36227/techrxiv.22583857.v1

Hietanen, M., & Eddebo, J. (2022). Towards a Definition of Hate Speech—With a Focus on Online Contexts. *Journal of Communication Inquiry*, *47*(4), 019685992211243. https://doi.org/10.1177/01968599221124309

Hotz, N. (2024, April 28). *What Is CRISP DM?* Data Science Project Management. https://www.datascience-pm.com/crisp-dm-2/

IBM. (2021, October 20). *Random Forest*. Ibm.com. https://www.ibm.com/think/topics/random-forest

Jiang, X., Yao, Y., Liu, S., Shen, F., Nie, L., & Hua, X. (2024, April 30). *Dual Dynamic Threshold Adjustment Strategy for Deep Metric Learning*. ArXiv.org. https://arxiv.org/abs/2404.19282

Jitendarpal Singn. (2018, June 25). *What should be the value for learning rate and*

momentum? Stack Overflow. https://stackoverflow.com/questions/51017181/what-should-be-the-value-for-learning-rate-and-momentum

Jurek, A., Bi, Y., Wu, S., & Nugent, C. (2013). A survey of commonly used ensemble-based classification techniques. *The Knowledge Engineering Review*, *29*(5), 551–581. https://doi.org/10.1017/s0269888913000155

Kapil, P., & Asif Ekbal. (2022). Transformer based Ensemble Learning to Hate Speech Detection Leveraging Sentiment and Emotion Knowledge Sharing. *TRANSFORMER BASED ENSEMBLE LEARNING to HATE SPEECH DETECTION LEVERAGING SENTIMENT and EMOTION KNOWLEDGE SHARING*. https://doi.org/10.5121/csit.2022.121014

Kapil, P., & Ekbal, A. (2024, November 24). *A Unified Multi-Task Learning Architecture for Hate Detection Leveraging User-Based Information*. ArXiv.org. https://arxiv.org/abs/2411.06855

Karabiber, F. (n.d.). *TF-IDF — Term Frequency-Inverse Document Frequency – LearnDataSci*. Www.learndatasci.com. https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/

Keras. (n.d.). *Keras documentation: Adam*. Keras.io. Retrieved March 5, 2025, from https://keras.io/2.18/api/optimizers/adam/

Khan, M. A., Yadav, N., Jain, M., & Goyal, S. (2023, June 26). *The Art of Embedding Fusion: Optimizing Hate Speech Detection*. ArXiv.org. https://arxiv.org/abs/2306.14939

Khasanah, I. N. (2021). Sentiment Classification Using fastText Embedding and Deep Learning Model. *Procedia Computer Science*, *189*, 343–350. https://doi.org/10.1016/j.procs.2021.05.103

Khouloud Mnassri, Praboda Rajapaksha, Reza Farahbakhsh, & Crespi, N. (2023). Hate

Speech and Offensive Language Detection Using an Emotion-Aware Shared Encoder. *Hate Speech and Offensive Language Detection Using an Emotion-Aware Shared Encoder*. https://doi.org/10.1109/icc45041.2023.10279690

KHULLAR, A., NKEMELU, D., V. NGUYEN, C., & L. BEST, M. (2023). Hate Speech Detection in Limited Data Contexts using Synthetic Data Generation. *Hate Speech Detection in Limited Data Contexts Using Synthetic Data Generation*, *18*, 1–18. https://arxiv.org/pdf/2310.02876

Knowledge Discovery in Databases. (2024, September 25). Dremio. https://www.dremio.com/wiki/knowledge-discovery-in-databases/

Kovács, G., Alonso, P., & Saini, R. (2021). Challenges of Hate Speech Detection in Social Media. *SN Computer Science*, *2*(2). https://doi.org/10.1007/s42979-021-00457-3

Kristína Machová, Martina Szabóova, Ján Paralič, & Ján Mičko. (2023, September 20). *Detection of emotion by text analysis using machine learning*. https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2023.1190326/full

Kumar, S. (2024, March 21). *BERT: Bidirectional Encoder Representations from Transformers*. Sushant Kumar. https://sushant-kumar.com/blog/bert

Laaksonen, S.-M., Haapoja, J., Kinnunen, T., Pöyhtäri, R., & Nelimarkka, M. (2020, February 5). *The Datafication of Hate: Expectations and Challenges in Automated Hate Speech Monitoring*. https://www.frontiersin.org/journals/big-data/articles/10.3389/fdata.2020.00003/full

Li, Q., Peng, H., Li, J., Xia, C., Yang, R., Sun, L., Yu, P. S., & He, L. (2022). A Survey on Text Classification: From Traditional to Deep Learning. *ACM Transactions on Intelligent Systems and Technology*, *13*(2), 1–41. https://doi.org/10.1145/3495162

Lim, Y. (2022, March 25). *Stacked ensembles - improving model performance on a higher*

level | Towards Data Science. Towards Data Science. https://towardsdatascience.com/stacked-ensembles-improving-model-performance-on-a-higher-level-99ffc4ea5523/

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollar, P. (2018). Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *10*(5), 1–1. https://doi.org/10.1109/tpami.2018.2858826

Luna, Z. (2021, August 5). *CRISP-DM Phase 1: Business Understanding*. Analytics Vidhya. https://medium.com/analytics-vidhya/crisp-dm-phase-1-business-understanding-255b47adf90a

MacAvaney, S., Yao, H.-R., Yang, E., Russell, K., Goharian, N., & Frieder, O. (2019). Hate speech detection: Challenges and solutions. *PLOS ONE*, *14*(8), e0221152. https://doi.org/10.1371/journal.pone.0221152

Marzieh Babaeianjelodar, Prudhvi, Gurram Poorna, Lorenz, S. R., Chen, K., Mondal, S., Dey, S., & Kumar, N. (2022). Interpretable and High-Performance Hate and Offensive Speech Detection. *Interpretable and High-Performance Hate and Offensive Speech Detection*, 233–244. https://doi.org/10.1007/978-3-031-21707-4_18

Maslej-Krešňáková, V., Sarnovský, M., Butka, P., & Machová, K. (2020). Comparison of Deep Learning Models and Various Text Pre-Processing Techniques for the Toxic Comments Classification. *Applied Sciences*, *10*(23), 8631. https://doi.org/10.3390/app10238631

Mazari, A. C., Boudoukhani, N., & Djeffal, A. (2023). BERT-based ensemble learning for multi-aspect hate speech detection. *Cluster Computing*. https://doi.org/10.1007/s10586-022-03956-x

Microsoft. (n.d.). *Support Overview | Visual Studio*. Visual Studio. https://visualstudio.microsoft.com/support/

Microsoft. (2021, August). *Windows 11 System Requirements*. Support.microsoft.com. https://support.microsoft.com/en-us/windows/windows-11-system-requirements-86c11283-ea52-4782-9efd-7674389a7ba3

Moez Ali. (2024, January 12). *Top 26 Python Libraries for Data Science in 2024*. https://www.datacamp.com/blog/top-python-libraries-for-data-science?utm_source=google&utm_medium=paid_search&utm_campaignid=19589720824&utm_adgroupid=152984014214&utm_device=c&utm_keyword=&utm_matchtype=&utm_network=g&utm_adpostion=&utm_creative=716160943615&utm_targetid=dsa-2222697811398&utm_loc_interest_ms=&utm_loc_physical_ms=9066777&utm_content=DSA~blog~Python&utm_campaign=230119_1-sea~dsa~tofu_2-b2c_3-row-p2_4-prc_5-na_6-na_7-le_8-pdsh-go_9-nb-e_10-na_11-na-oct24&gad_source=1&gclid=Cj0KCQjw1Yy5BhD-ARIsAI0RbXbGC-8PNr4gHUZCnVUNKVv5AQX_NfvjUrR5qRGRexKOx_Y-EH9hc2IaAn5dEALw_wcB

Ng, C. (2024, June 13). *The difference between SEMMA and CRISP-DM*. Starburst. https://www.starburst.io/blog/semma-vs-crisp-dm/

None Sunanda, Archana Balmik, & Nandy, A. (2024). A novel feature fusion technique for robust hand gesture recognition. *Multimedia Tools and Applications*, *83*(25), 65815–65831. https://doi.org/10.1007/s11042-024-18173-4

Occhipinti, A., Rogers, L., & Angione, C. (2022). A pipeline and comparative study of 12 machine learning models for text classification. *Expert Systems with Applications*, *201*, 117193. https://doi.org/10.1016/j.eswa.2022.117193

Office, U. S. G. A. (2024). *Online Extremism is a Growing Problem, But What's Being Done About It? | U.S. GAO*. Www.gao.gov. https://www.gao.gov/blog/online-extremism-

growing-problem-whats-being-done-about-it#:~:text=Research%20indicated%20up%20to%20a

Palacios, H. J. G., Toledo, R. A. J., Pantoja, G. A. H., & Navarro, Á. A. M. (2017). A comparative between CRISP-DM and SEMMA through the construction of a MODIS repository for studies of land use and cover change. *Advances in Science, Technology and Engineering Systems Journal*, *2*(3), 598–604. https://doi.org/10.25046/aj020376

Pallavi Padav. (2024, May 11). *From Raw Text to Insightful Analysis: NLP Text Preprocessing Explained*. Medium; Women in Technology. https://medium.com/womenintechnology/from-raw-text-to-insightful-analysis-nlp-text-preprocessing-explained-03dad2d1a3c6

Parate, N. (2024, August 12). *Joint Representation Learning Vs Coordinated Learning in Multimodal Context*. Medium. https://medium.com/@nikitaparate9/joint-representation-learning-vs-coordinated-learning-in-multimodal-context-01890e0a4730

Penn Treebank P.O.S. Tags. (n.d.). Www.ling.upenn.edu. https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Plaza-Del-Arco, F. M., Molina-González, M. D., Ureña-López, L. A., & Martín-Valdivia, M. T. (2021). A Multi-Task Learning Approach to Hate Speech Detection Leveraging Sentiment Analysis. *IEEE Access*, *9*, 112478–112489. https://doi.org/10.1109/ACCESS.2021.3103697

Politi, M. (2022, July 6). *Feature Extraction with BERT for Text Classification*. Medium. https://towardsdatascience.com/feature-extraction-with-bert-for-text-classification-533dde44dc2f

Poulinakis, K. (2022, December 15). *Multimodal Deep Learning: Definition, Examples, Applications*. Www.v7labs.com. https://www.v7labs.com/blog/multimodal-deep-

learning-guide

Pritom Mojumder, Hasan, M., Hossain, M. F., & Hasan, A. (2020). A Study of fastText Word Embedding Effects in Document Classification in Bangla Language. *Springer EBooks*, 441–453. https://doi.org/10.1007/978-3-030-52856-0_35

Proscura, P., & Zaytsev, A. (2022, June 27). *Effective training-time stacking for ensembling of deep neural networks*. ArXiv.org. https://arxiv.org/abs/2206.13491

Python, R. (n.d.). *How to Use Python Lambda Functions – Real Python*. Realpython.com. https://realpython.com/python-lambda/

RAJESWARI, R. (2025). *Optimization in Deep Learning* [Personal communication].

Rana, A., & Jha, S. (2022). *Emotion Based Hate Speech Detection using Multimodal Learning*. https://arxiv.org/pdf/2202.06218

Richard O'Hara, CMA, CFA, Lisa S. Haylon, & Douglas M. Boyle. (2023, February 1). *A Data Analytics Mindset with CRISP-DM | IMA*. IMA. https://www.sfmagazine.com/articles/2023/february/a-data-analytics-mindset-with-crisp-dm

Sáez, J. A., & Romero-Béjar, J. L. (2022). On the Suitability of Bagging-Based Ensembles with Borderline Label Noise. *Mathematics*, *10*(11), 1892. https://doi.org/10.3390/math10111892

Saha, K., Chandrasekharan, E., & De Choudhury, M. (2019). Prevalence and Psychological Effects of Hateful Speech in Online College Communities. *Proceedings of the 10th ACM Conference on Web Science - WebSci '19*, 255–264. https://doi.org/10.1145/3292522.3326032

Sakib Ashraf Zargar. (2021, April 17). *Introduction to Sequence Learning Models: RNN, LSTM, GRU*. https://doi.org/10.13140/RG.2.2.36370.99522

Saleh, H., Alhothali, A., & Moria, K. (2023). Detection of Hate Speech using BERT and Hate

Speech Word Embedding with Deep Model. *Applied Artificial Intelligence*, *37*(1). https://doi.org/10.1080/08839514.2023.2166719

Schlötterer, J., Seifert, C., & Granitzer, M. (2017). On Joint Representation Learning of Network Structure and Document Content. *Lecture Notes in Computer Science*, 237–251. https://doi.org/10.1007/978-3-319-66808-6_16

SenticNet. (n.d.). Sentic.net. https://sentic.net/

Sharkawy, A.-N. (2020). Principle of Neural Network and Its Main Types: Review. *Journal of Advances in Applied & Computational Mathematics*, *7*(1), 8–19. https://doi.org/10.15377/2409-5761.2020.07.2

Shiksha. (2023, August 25). *All About Train Test Split*. Shiksha.com. https://www.shiksha.com/online-courses/articles/train-test-split/

Siino, M., Tinnirello, I., & Marco La Cascia. (2024). Is text preprocessing still worth the time? A comparative survey on the influence of popular preprocessing methods on Transformers and traditional classifiers. *Information Systems*, *121*, 102342–102342. https://doi.org/10.1016/j.is.2023.102342

Smart Vision Europe. (2017). *What is the CRISP-DM methodology?* Smart Vision - Europe. https://www.sv-europe.com/crisp-dm-methodology/

SPRH LABS. (2019, March 21). *Understanding Deep Learning: DNN, RNN, LSTM, CNN and R-CNN*. Medium; Medium. https://medium.com/@sprhlabs/understanding-deep-learning-dnn-rnn-lstm-cnn-and-r-cnn-6602ed94dbff

Subramanian, M., Easwaramoorthy Sathiskumar, V., Deepalakshmi, G., Cho, J., & Manikandan, G. (2023). A survey on hate speech detection and sentiment analysis using machine learning and deep learning models. *Alexandria Engineering Journal*, *80*, 110–121. https://doi.org/10.1016/j.aej.2023.08.038

Syed Muhammad Mubashir Rizvi. (2024, August 16). *Gini Index and Entropy | 2 Ways to*

Measure Impurity in Data. Data Science Dojo. https://datasciencedojo.com/blog/gini-index-and-entropy/

Um, A. (2021, April 12). *L1, L2 Regularization in XGBoost Regression*. Medium. https://albertum.medium.com/l1-l2-regularization-in-xgboost-regression-7b2db08a59e0

Van Houdt, G., Mosquera, C., & Nápoles, G. (2020). A review on the long short-term memory model. *Artificial Intelligence Review*, *53*(8). https://doi.org/10.1007/s10462-020-09838-1

Velankar, A., Patil, H., & Joshi, R. (2022). *A Review of Challenges in Machine Learning based Automated Hate Speech Detection*. https://arxiv.org/pdf/2209.05294

Wang, S., Zhou, W., & Jiang, C. (2019). A survey of word embeddings based on deep learning. *Computing*, *102*(3), 717–740. https://doi.org/10.1007/s00607-019-00768-7

What are Word & Sentence Embedding? 5 Applications. (2024, August 28). Airbyte.com; Airbyte. https://airbyte.com/data-engineering-resources/sentence-word-embeddings

What is Jupyter Notebook? | Domino Data Lab. (n.d.). Domino.ai. https://domino.ai/data-science-dictionary/jupyter-notebook

XGBoost Parameters — xgboost 1.5.2 documentation. (2022). Xgboost.readthedocs.io. https://xgboost.readthedocs.io/en/stable/parameter.html

Yadav, R. K. (2023, January 7). *5 Easy Ways to Use ast.literal_eval() and its Functions*. Python Pool. https://www.pythonpool.com/ast-literal_eval/

Yesi Novaria Kunang, Siti Nurmaini, Deris Stiawan, Yudho Bhakti, & Suprapto. (2021). Deep learning with focal loss approach for attacks classification. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, *19*(4), 1407–1418. https://doi.org/10.12928/TELKOMNIKA.v19i4.18772

Yu, L., & Zhou, N. (2021). *Survey of Imbalanced Data Methodologies*.

https://arxiv.org/pdf/2104.02240

Yuan, L., & Marian-Andrei Rizoiu. (2024). Generalizing Hate Speech Detection Using Multi-Task Learning: A Case Study of Political Public Figures. *Computer Speech & Language*, *89*, 101690–101690. https://doi.org/10.1016/j.csl.2024.101690

YugmPatel. (2024). *GitHub - YugmPatel/CMPE-255-CRISP-DM-SEMMA-KDD*. GitHub. https://github.com/YugmPatel/CMPE-255-CRISP-DM-SEMMA-KDD/tree/main

Zhang, R., Yu, Y., Shen, J., Cui, X., & Zhang, C. (2023). Local Boosting for Weakly-Supervised Learning. *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 3364–3375. https://doi.org/10.1145/3580305.3599417

Zilber, A. (2024, May 8). *Facebook oversight board set to rule on whether the phrase "From the river to the sea" is "hate speech."* New York Post. https://nypost.com/2024/05/08/business/facebook-oversight-board-set-to-rule-on-from-the-river-to-the-sea-ban/?utm_source=chatgpt.com

APPENDICES

Appendix A: PPF

Office Record	Receipt
Date Received: Received by whom:	Student name: Student number: Received by: Date:

DRAFT PROJECT PROPOSAL FORM

Proposal ID :

Supervisor : **1)** Assoc. Prof . Dr. Raja Rajeswari
2) Dr. Minnu Helen Joseph
3) Assoc. Prof. Dr. Nirase Fathima Abubacker
4) Mr. Raheem Mafas
5) Dr. Vazeerudeen Hameed

Student Name : **Tang Yu Tong**

Student No : **TP069018**

Email Address : tp069018@mail.apu.edu.my & yutongtang10@gmail.com

Programme Name : Bachelor of Science (Honours) in **Computer Science** specialization in **Data Analytics**

Title of project : **Emotion-Aware Hate Speech Detection in User-Generated Content Using Natural Language Processing**

Please record which module(s) your topic is related to:

CT107-3-3- Text Analytics and Sentiment Analysis
CT119-3-2- Data Mining and Predictive Modelling
CT075-3-2- Data Management
CT108-3-1- Programming with Python

1

1. Introduction

Assume the reader has very little knowledge of the subject. Introduce the topic, the sector of business/industry concerned and how the project relates to it. Define the context of the problem and identify the research required to solve it.

New social media platforms like Reddit and Twitter allow people to freely, quickly, and broadly connect with others in our daily lives. But as everything has two sides, social media can also be used to disseminate false information. This misinformation can be aggressive, radical, or even phony. This is because social media is somewhat anonymous. The so-called "hate speech," which communicates hate feeling on social media, is one of the most typical patterns among the different negative information patterns. Hate speech is a complicated phenomenon that is mostly exclusive to connections between groups, in contrast to generic emotions.

Hate speech has been shown to have profound psychological and emotional impacts, often leading to trauma-like symptoms in its victims. Studies using the crisis reaction model reveal that hate speech triggers a cycle of emotional, cognitive, and behavioral responses similar to those caused by other traumatic events. Victims often experience symptoms akin to post-traumatic stress disorder (PTSD), such as pain, fear, anxiety, nightmares, and persistent intrusive thoughts of intimidation and dehumanization. Even individuals who are not direct targets of hate speech, particularly in environments like college campuses, may experience stress, lowered self-esteem, and social isolation. Prejudice, discrimination, and intolerance create a hostile atmosphere, affecting the mental well-being and social integration of minority groups. Hate speech, therefore, poses not only a threat to personal safety but also contributes to long-term psychological harm and societal division. (Saha et al., 2019)

Numerous online forums, including Facebook, YouTube, and Twitter, have standards in place to delete anything that is considered hate speech. The research of automatic hate speech identification is highly motivated by societal concerns and the growing amount of hate speech on the internet. Hateful content may be stopped from spreading by automating its identification, which contributes to the development of a safer online community. However, identifying hate speech is a difficult process. There are differences of opinion over the

definition of hate speech. This implies that depending on how each term is used, certain content may be regarded as hate speech by some people but not by others. (MacAvaney et al., 2019)

Due to the significant social issues caused by the spread of hate speech, this research paper proposes a deep learning approach incorporating emotional features to improve detection accuracy. Hate speech often stems from strong negative emotions like anger, disgust, fear, and sadness. By leveraging emotion classification, which automatically categorizes text into various emotional categories, this study aims to enhance the detection of hate speech and offensive language. Integrating emotional awareness into the model offers a more nuanced understanding of the content, ultimately contributing to more effective hate speech moderation.

2. Problem Statement

Identify past and current work in the subject area. Outline the key references to other people's work, indicate for the most pertinent of these how your proposal relates to the ideas they contain.

The rise of online hate speech has emerged as a pressing issue, significantly exacerbated by the proliferation of social media and other internet platforms. High-profile extremist attacks, such as the 2015 shooting at Emanuel African Methodist Episcopal Church, the 2019 Walmart attack in El Paso, and the 2022 shooting at a nightclub in Colorado Springs, starkly illustrate the severe and often tragic consequences of unchecked online hate speech. This pervasive problem manifests in numerous forms, including derogatory comments targeting individuals based on race, nationality, ethnicity, gender, gender identity, religion, disability, and sexual orientation. Research indicates that up to one-third of internet users have encountered such harmful content, with the prevalence even higher in online gaming communities, where approximately 50% of users report experiencing hate speech (Office, 2024).

Data Analytics Challenges in Imbalanced Distribution of Hate Speech

The detection of hateful content faces a significant challenge due to the inherent imbalance in the distribution of such content within the overall data. While harmful speech is a serious concern, it constitutes only a small fraction of social media interactions. This data imbalance is also

reflected in hate speech corpora, such as the HASOC 2019 training set, where less than 40% of the tweets were categorized as hateful or offensive. As a result, the rarity of hate speech instances poses difficulties for machine learning models, which can struggle to detect minority classes accurately. Despite the existence of standard resampling methods to address this issue, effectively mitigating its impact remains a key challenge in ensuring accurate and fair hate speech detection. (Kovács et al., 2021).

Challenges of Keyword-Based Approaches in Identifying Hate Speech

Detecting and moderating hateful and offensive speech on social media presents significant challenges due to the limitations of current keyword-based approaches. Users often employ obfuscation techniques, such as substituting letters with similar-looking numbers or using creative spellings, to bypass automated moderation systems, “E” with 3, or “l” with 1, and so on. Additionally, offensive content can be embedded within seemingly innocuous statements, making it difficult to identify. For example, a comment like "You have a lot of money, unlike me, who has very little money" may mask underlying hostility that is not immediately apparent (Kovács et al., 2021). These factors diminish the effectiveness of keyword-based filters and highlight the need for more sophisticated detection methods. (Laaksonen et al., 2020)

Limitations of Basic Text Analysis Methods

The detection of hate speech is significantly complicated by the subtleties and nuances that basic text analysis methods often fail to capture. Non-profanic hate speech, which targets specific communities with harmful intent without employing explicit offensive language, presents a formidable challenge for traditional text analysis tools. On the other hand, profanity is not always a sign of malice; it can be employed in a number of situations, such a joke or a show of enthusiasm, which could result in false positives. For example, comments like "He is a fking genius" and "He is a fking idiot" may be incorrectly flagged as hate speech, particularly in situations where such language is used more casually (Kovács et al., 2021).

The current methodologies for detecting abusive content on social media encounter several key challenges, including data bias, insufficient training datasets, and difficulties in capturing contextual elements, sarcasm, and cultural nuances. The complexity of hate speech, which

involves psychological, cognitive, and cultural dimensions, is often not adequately addressed by traditional text classification models (Changrong Min et al., 2023).

3. Project Aim and Objectives

Identify the AIM(s) of the project, i.e. what the overall achievement is intended to be, in terms of both academic and commercial/industrial advances. Identify the particular intellectual difficulties posed by the proposal, the problems to be addressed, and explain how these might be solved. Clearly list individual measurable OBJECTIVES which can be related to the workplan and deliverables.
Aims and objectives are subject to approval from supervisor and students are expected to revise them if deemed inappropriate for a Level 3 project.

Aim:

The primary aim of this research is to enhance the detection of hate speech in user-generated content by incorporating emotional analysis using NLP techniques. This approach aims to improve the accuracy and contextual understanding of hate speech detection, addressing limitations in existing basic text analysis models.

The objectives of this project are:

1. To investigate and obtain comprehensive datasets for emotion analysis and hate speech detection to ensure effective model training.
2. To develop, train, and deploy an advanced NLP model capable of detecting hate speech by incorporating emotion analysis, overcoming the limitations of traditional keyword-based and basic text analysis methods, and implementing it in a web-based application for practical use.
3. To evaluate the performance of the developed model with a focus on improving its accuracy.

4. Literature Review

The research was done by (Velankar et al., 2022), which explored hate speech detection presents numerous challenges that arise at different levels, including data, models, and human factors. At the data level, one of the primary obstacles is the limited availability of large, diverse datasets that accurately represent the wide spectrum of hateful content. Collecting a sufficient amount of labeled data for various hate speech categories is a difficult task, particularly given the imbalance between the volume of hateful versus non-hateful content. This problem is further exacerbated by the absence of a universal definition of hate speech, making it difficult to standardize labeling criteria. Additionally, the complexity increases when dealing with regional languages, which are often underrepresented in existing datasets, and with the prevalence of code-mixed text, where users mix languages within the same text. Disguised hate speech, where offensive words are intentionally misspelled or altered to avoid detection, also presents significant challenges, particularly as this requires advanced techniques to detect hate speech beyond traditional word-based filtering.

At the model level, the challenges are equally significant. Bias in models is a recurrent issue, as models trained on biased datasets may disproportionately misclassify certain demographic groups or content types. For example, hate speech against underrepresented groups in training data may not be detected as accurately as against more frequently represented groups. Additionally, the presence of non-profanic hate speech—where offensive content is conveyed without using explicitly offensive words—makes it difficult for models that rely heavily on keyword detection. Similarly, false positives can occur when profanity is used in non-hateful contexts, which models may still flag as hate speech due to the use of specific terms. Overlapping labels also complicate hate speech detection, as content often exhibits characteristics of multiple categories, making it hard for models to assign the appropriate label. (Velankar et al., 2022)

Human-level challenges play a crucial role in complicating hate speech detection. The subjective nature of hate speech means that different annotators may interpret the same content differently, resulting in inconsistencies in the labeling process. Annotators' cultural, linguistic, and personal backgrounds influence their interpretation of what constitutes hate speech, leading to a lack of consensus and ultimately affecting the quality of the training data. Additionally, understanding

the user's intent behind a statement is essential, as hate speech can sometimes be disguised as sarcasm or presented in a historical or political context, making it challenging to label accurately. For instance, without understanding the historical connotations of certain statements, annotators may overlook content that qualifies as hate speech. (Velankar et al., 2022).

The research done by (Laaksonen et al., 2020), delves into the complexities of hate speech detection, highlighting both the technical and social challenges encountered during the project. The research list out how hate speech is a difficult concept to quantify due to its evolving nature and diverse cultural, linguistic, and juridical interpretations.

From a technical standpoint, hate speech detection often simplifies intricate emotions into numerical representations, which flattens the variety of expressions and overlooks the social and contextual nuances of language. Support vector machines and Bag-of-Words are two popular machine learning techniques. Bag-of-Words emphasizes specific words as features, which is insufficient for accurately capturing hate speech in complex social media environments. The evolving nature of language on platforms like social media, including the use of visual forms of communication, makes it harder to detect hate speech that uses new pejoratives, misspellings, or covert expressions.(Laaksonen et al., 2020)

The study underscores the need for more iterative, context-aware approaches in developing hate speech detection systems. It highlights the importance of trained annotators who understand the cultural and linguistic subtleties of hate speech and suggests that future models should account for contextual factors, such as the broader discussion context and the status of the message's author. However, including these features may not necessarily yield more usable models, reinforcing the need for careful validation and human-computing hybrid approaches. (Laaksonen et al., 2020)

From this research (Haque & Md Naseef-Ur-Rahman Chowdhury, 2023), notice that ensemble learning techniques in hate speech detection offer notable advantages, such as reducing false positives and negatives, but also face some challenges. These methods require large datasets for training and depend heavily on accurate preprocessing to achieve optimal performance.

Additionally, the continuous evolution of social media leads to new forms of hate speech, necessitating frequent updates to ensemble models to maintain their effectiveness.

Psychological research by (Chiril et al., 2021) emphasizes that abusive language is closely related to the speaker's emotional state. Capturing such emotion in the text is challenging, especially in identifying complex linguistic elements like sarcasm, irony, idioms, metaphors, and similes, all of which deviate from literal meaning. Unimodal text models struggle to interpret these nuances. Additionally, detecting homophones, which are highly context- or culture-dependent, further complicates sentiment analysis in the text. (Kristína Machová et al., 2023).

5. Deliverables

Provide a clear list of the outputs from the project.

Target User:

The primary users of this project are clients and developers responsible for implementing hate speech detection systems. These users may include researchers, software engineers, and data scientists focused on enhancing online safety and promoting responsible digital communication through effective hate speech detection solutions.

Deliverables:

1. **Data Understanding**

   A comprehensive report detailing the data collection and exploration processes for both hate speech and sentiment datasets, highlighting the sources, criteria for selection, and initial insights gained from the data analysis.

2. **Data Preprocessing Pipeline**

   A documented preprocessing pipeline outlining the steps taken, including data cleaning (removal of symbols and stop words), normalization (lowercasing), stemming, and tokenization to prepare the datasets for analysis.

3. **Text Analysis and Natural Language Processing Techniques**

A summary of the natural language processing (NLP) techniques and text analysis methods employed, including sentiment analysis and emotion classification frameworks, tailored for detecting hate speech.

4. **Trained Emotion-Aware Deep Learning Model**

   A fully trained and fine-tuned deep learning model capable of emotion-aware hate speech detection, which incorporates emotion classification to enhance detection performance. The model will undergo rigorous testing and validation.

5. **Performance Evaluation Report**

   A detailed report evaluating the performance of the emotion-aware hate speech detection model, employing metrics such as accuracy, precision, recall, F1 score, and confusion matrix analysis to provide a thorough assessment of the model's capabilities.

6. **Model Deployment and Visualization Tools**

   A deployed version of the trained model along with interactive data visualizations, including charts and graphs that illustrate key findings, such as the correlation between emotional states and hate speech, trends in emotional language across user-generated content, and model performance metrics.

6. References

- Changrong Min, Hongfei Lin, Ximing Li, He Zhao, Junyu Lu, Liang Yang, & Bo Xu. (2023). *Finding hate speech with auxiliary emotion detection from self-training multi-label learning perspective.* https://www.sciencedirect.com/science/article/abs/pii/S1566253523001045

- Chiril, P., Pamungkas, E. W., Benamara, F., Moriceau, V., & Patti, V. (2021). Emotionally Informed Hate Speech Detection: A Multi-target Perspective. *Cognitive Computation.* https://doi.org/10.1007/s12559-021-09862-5

- Haque, A., & Md Naseef-Ur-Rahman Chowdhury. (2023). Hate Speech Detection in Social Media Using the Ensemble Learning Technique. *INDIGO (University of Illinois at Chicago)*. https://doi.org/10.36227/techrxiv.22583857.v1

- Khouloud Mnassri, Praboda Rajapaksha, Reza Farahbakhsh, & Crespi, N. (2023). Hate Speech and Offensive Language Detection Using an Emotion-Aware Shared Encoder. *Hate Speech and Offensive Language Detection Using an Emotion-Aware Shared Encoder*. https://doi.org/10.1109/icc45041.2023.10279690

- Kovács, G., Alonso, P., & Saini, R. (2021). Challenges of Hate Speech Detection in Social Media. *SN Computer Science*, *2*(2). https://doi.org/10.1007/s42979-021-00457-3

- Kristína Machová, Martina Szabóova, Ján Paralič, & Ján Mičko. (2023, September 20). *Detection of emotion by text analysis using machine learning.* https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2023.1190326/full

- Laaksonen, S.-M., Haapoja, J., Kinnunen, T., Pöyhtäri, R., & Nelimarkka, M. (2020, February 5). *The Datafication of Hate: Expectations and Challenges in Automated Hate Speech Monitoring*. https://www.frontiersin.org/journals/big-data/articles/10.3389/fdata.2020.00003/full

- MacAvaney, S., Yao, H.-R., Yang, E., Russell, K., Goharian, N., & Frieder, O. (2019). Hate speech detection: Challenges and solutions. *PLOS ONE*, *14*(8), e0221152. https://doi.org/10.1371/journal.pone.0221152

- Office, U. S. G. A. (2024). *Online Extremism is a Growing Problem, But What's Being Done About It? | U.S. GAO*. Www.gao.gov. https://www.gao.gov/blog/online-extremism-growing-problem-whats-being-done-about-it#:~:text=Research%20indicated%20up%20to%20a

- Plaza-Del-Arco, F., Halat, S., Padó, S., & Klinger, R. (2022). *Multi-Task Learning with Sentiment, Emotion, and Target Detection to Recognize Hate Speech and Offensive Language 4.0 International (CC BY 4.0)*. https://arxiv.org/pdf/2109.10255

- Saha, K., Chandrasekharan, E., & De Choudhury, M. (2019). Prevalence and Psychological Effects of Hateful Speech in Online College Communities. *Proceedings of the 10th ACM Conference on Web Science - WebSci '19*, 255–264. https://doi.org/10.1145/3292522.3326032

- Velankar, A., Patil, H., & Joshi, R. (2022). *A Review of Challenges in Machine Learning based Automated Hate Speech Detection*. https://arxiv.org/pdf/2209.05294

Appendix B: Ethics Forms (Fast Track)

Office Record	Receipt – Fast-Track Ethical Approval
Date Received:	Student name: Student number:
Received by whom:	Received by: Date:

APU / APIIT FAST-TRACK ETHICAL APPROVAL FORM (STUDENTS)

Tick one box (level of study):
- ☐ POSTGRADUATE (PhD / MPhil / Masters)
- ☐ UNDERGRADUATE (Bachelors degree)
- ☐ FOUNDATION / DIPLOMA / Other categories

Tick one box (purpose of approval):
- ☒ Thesis / Dissertation / FYP project
- ☐ Module assignment
- ☐ Other: _____

Title of Programme on which enrolled:
Bachelor Of Science (Honours) In Computer Science Specialism In Data Analytics

Tick one box: ☒ Full-Time Study or ☐ Part-Time Study

Title of project / assignment:
Emotion-Aware Hate Speech Detection in User-Generated Content Using Natural Language Processing

Name of student researcher:
Tang Yu Tong

Name of supervisor / lecturer:
Ts. Nicholas Teh Sek Kit

Student Researchers- please note that certain professional organisations have ethical guidelines that you may need to consult when completing this form.

Supervisors/Module Lecturers - please seek guidance from the Chair of the APU Research Ethics Committee if you are uncertain about any ethical issue arising from this application.

		YES	NO	N/A
1	Will you describe the main procedures to participants in advance, so that they are informed about what to expect?			✓
2	Will you tell participants that their participation is voluntary?			✓
3	Will you obtain written consent for participation?			✓
4	If the research is observational, will you ask participants for their consent to being observed?			✓
5	Will you tell participants that they may withdraw from the research at any time and for any reason?			✓
6	With questionnaires and interviews will you give participants the option of omitting questions they do not want to answer?			✓
7	Will you tell participants that their data will be treated with full confidentiality and that, if published, it will not be identifiable as theirs?			✓
8	Will you give participants the opportunity to be debriefed i.e. to find out more about the study and its results?			✓

If you have ticked **No** to any of Q1-8 you should complete the full Ethics Approval Form.

		YES	NO	N/A
9	Will your project/assignment deliberately mislead participants in any way?		✓	

10	Is there any realistic risk of any participants experiencing either physical or psychological distress or discomfort?		✓	
11	Is the nature of the research such that contentious or sensitive issues might be involved?		✓	

If you have ticked **Yes** to 9, 10 or 11 you should complete the full Ethics Approval Form. In relation to question 10 this should include details of what you will tell participants to do if they should experience any problems (e.g. who they can contact for help). You may also need to consider risk assessment issues.

			YES	NO	N/A
12	Does your project/assignment involve work with animals?			✓	
13	Do participants fall into any of the following special groups? **Note that you may also need to obtain satisfactory clearance from the relevant authorities**	Children (under 18 years of age)		✓	
		People with communication or learning difficulties			
		Patients			
		People in custody			
		People who could be regarded as vulnerable			
		People engaged in illegal activities (eg drug taking)			
14	Does the project/assignment involve external funding or external collaboration where the funding body or external collaborative partner requires the University to provide evidence that the project/assignment had been subject to ethical scrutiny?			✓	

If you have ticked **Yes** to 12, 13 or 14 you should complete the full Ethics Approval Form. There is an obligation on student and supervisor to bring to the attention of the APU Research Ethics Committee any issues with ethical implications not clearly covered by the above checklist.

STUDENT RESEARCHER

Provide in the boxes below (plus any other appended details) information required in support of your application, THEN SIGN THE FORM.

Please Tick Boxes

I consider that this project/assignment has no significant ethical implications requiring a full ethics submission to the APU Research Ethics Committee.	✓
Give a brief description of participants and procedure (methods, tests used etc) in up to 150 words. The dataset for this project was collected from an open-source repository on GitHub. It consists of two datasets: a hate speech dataset containing 85,048 rows with two variables (text and hate speech label) and an emotion dataset containing 70,001 rows with 28 variables and one text column. The procedure involves data understanding and preprocessing steps, including cleaning, handling missing values, and feature extraction. Following this, two individual models will be built. Lastly, an ensemble model will be implemented to improve accuracy and performance by combining the strengths of the individual models. The ensemble approach is expected to enhance the detection of hate speech by leveraging multiple algorithms.	
I also confirm that: i) All key documents e.g. consent form, information sheet, questionnaire/interview are appended to this application.	✓

Or ii) Any key documents e.g. consent form, information sheet, questionnaire/interview schedules which need to be finalised following initial investigations will be submitted for approval by the project/assignment supervisor/module lecturer before they are used in primary data collection.	

E-signature		Print Name	Tang Yu Tong	Date	22 October 2024

(Student Researcher)

Please note that any variation to that contained within this document that in any way affects ethical issues of the stated research requires the appending of new ethical details. New ethical consent may need to be sought.
The completed form (and any attachments) should be submitted for consideration by your Supervisor/Module Lecturer

SUPERVISOR/MODULE LECTURER
PLEASE CONFIRM THE FOLLOWING:

	Please Tick Box
I consider that this project/assignment has no significant ethical implications requiring a full ethics submission to the APU Research Ethics Committee	√
i) I have checked and approved the key documents required for this proposal (e.g. consent form, information sheet, questionnaire, interview schedule) Or	√
ii) I have checked and approved draft documents required for this proposal which provide a basis for the preliminary investigations which will inform the main research study. I have informed the student researcher that finalised and additional documents (e.g. consent form, information sheet, questionnaire, interview schedule) must be submitted for approval by me before they are used for primary data collection.	√

SUPERVISOR AND SECOND ACADEMIC SIGNATORY

STATEMENT OF ETHICAL APPROVAL (please delete as appropriate)

1) **THIS PROJECT/ASSIGNMENT HAS BEEN CONSIDERED USING AGREED APU/SU PROCEDURES AND IS NOW APPROVED**

2) **THIS PROJECT/ASSIGNMENT HAS BEEN APPROVED IN PRINCIPLE AS INVOLVING NO SIGNIFICANT ETHICAL IMPLICATIONS, BUT FINAL APPROVAL FOR DATA COLLECTION IS SUBJECT TO THE SUBMISSION OF KEY DOCUMENTS FOR APPROVAL BY SUPERVISOR (see Appendix A)**

E-signature…  … … … … … … … … Print Name… **NICHOLAS TEH SEK KIT** Date… **23/10/2024**
(Supervisor/Lecturer)

E-signature… … … … … … … … … … … Print Name… … … … … … … … … … … Date… … … …
(Second Academic Signatory)

Office Record	Receipt – Appendix A (Fast-Track Ethics Form)
Date Received: Received by whom:	Student name: Student number: Received by: Date:

APPENDIX A
AUTHORISATION FOR USE OF KEY DOCUMENTS

Completion of Appendix A is required when for good reasons key documents are not available when a fast track application is approved by the supervisor/module lecturer and second academic signatory.

I have now checked and approved all the key documents associated with this proposal e.g. consent form, information sheet, questionnaire, interview schedule

Title of project/assignment… …

… ....

Name of student researcher ….

Student ID: ……………………………… Intake: … … … … … … … … … … … … … … … …

E-signature… … … … … … … … … … … … Print Name… … … … … … … … … … … … Date… … …….. …
(Supervisor/Lecturer)

Appendix C: Log Sheets (Six Meeting Log Sheets)

First Meeting Log Sheets

(APU: Serial Number)

PLS V1.0

Project Log Sheet – Supervisory Session

Notes on use of the project log sheet:

1. This log sheet is designed for meetings of more than 15 minutes duration, of which there must be at minimum SIX (6) during the course of the project (SIX mandatory supervisory sessions).

2. The student should prepare for the supervisory sessions by deciding which question(s) he or she needs to ask the supervisor and what progress has been made (if any) since the last session, and noting these in the relevant sections of the form, effectively forming an agenda for the session.

3. A log sheet is to be brought by the STUDENT to each supervisory session.

4. The actions by the student (and, perhaps the supervisor), which should be carried out before the next session should be noted briefly in the relevant section of the form.

5. The student should leave a copy (after the session) of the Project Log Sheet with the supervisor and to the administrator at the academic counter. A copy is retained by the student to be filed in the project file.

6. It is recommended that students bring along log sheets of previous meetings together with the project file during each supervisory session.

7. The log sheet is an important deliverable for the project and an important record of a student's organisation and learning experience. The student **must** hand in the log sheets as an appendix of the final year documentation, with sheets dated and numbered consecutively.

Student's name: Tang Yu Tong **Date:** 14th October 2024 **Meeting No:** 1

Project title:
Emotion-Aware Hate Speech Detection in User-Generated Content Using Natural Language Processing

Intake: APU3F2408CS(DA)

Supervisor's name: Ts. Nicholas Teh Sek Kit **Supervisor's signature:**

Items for discussion (noted by student before mandatory supervisory meeting):

1. Checking my final year project title, objective, and problem statement.
2. Discuss the idea of my final year project.
3. Showing the datasets I will use for the final year project.

Record of discussion (noted by student during mandatory supervisory meeting):

1. **Objective Structure and Problem Statement:** The objective structure needs to be updated, and the problem statement should include at least three key points.
2. **Literature Review (LR):** The literature review cannot be written in a textbook style. It should be revised to focus on the research gap or challenge based on the problem statement.
3. **Deliverables:** The deliverables need to be improved and should be aligned with the methodology process, outlining the outcomes of each step.

Project Log Sheet

Action List (to be attempted or completed by student by the next mandatory supervisory meeting):
1. Complete FYP chapter 1 and chapter 2.

Note: A student should make an appointment to meet his or her supervisor (via the consultation system) at least ONE (1) week prior to a mandatory supervisor session – please see document on project timelines. In the event a supervisor could not be booked for consultation, the project manager should be informed ONE (1) week prior to the session so that a meeting can be subsequently arranged.

Second Meeting Log Sheets

(APU: Serial Number)

PLS V1.0

Project Log Sheet – Supervisory Session

Notes on use of the project log sheet:

1. This log sheet is designed for meetings of more than 15 minutes duration, of which there must be at minimum SIX (6) during the course of the project (SIX mandatory supervisory sessions).
2. The student should prepare for the supervisory sessions by deciding which question(s) he or she needs to ask the supervisor and what progress has been made (if any) since the last session, and noting these in the relevant sections of the form, effectively forming an agenda for the session.
3. A log sheet is to be brought by the STUDENT to each supervisory session.
4. The actions by the student (and, perhaps the supervisor), which should be carried out before the next session should be noted briefly in the relevant section of the form.
5. The student should leave a copy (after the session) of the Project Log Sheet with the supervisor and to the administrator at the academic counter. A copy is retained by the student to be filed in the project file.
6. It is recommended that students bring along log sheets of previous meetings together with the project file during each supervisory session.
7. The log sheet is an important deliverable for the project and an important record of a student's organisation and learning experience. The student **must** hand in the log sheets as an appendix of the final year documentation, with sheets dated and numbered consecutively.

Student's name: Tang Yu Tong **Date:** 19th November 2024 **Meeting No:** 2

Project title:
Emotion-Aware Hate Speech Detection in User-Generated Content Using Natural Language Processing

Intake: APU3F2408CS(DA)

Supervisor's name: Nicholas Teh Sek Kit **Supervisor's signature:**

Items for discussion (noted by student before mandatory supervisory meeting):

1. Chapter 3 IR structure.
2. Asking if need to add additional datasets for the future is possible.
3. Text Preprocessing and EDA some code references from website is it possible.

Record of discussion (noted by student during mandatory supervisory meeting):

1. Chapter 3 EDA, Data Preprocessing, and Data Understanding part need to be restructured and combined.
2. The formula in Chapter 2 uses typing not screenshots.
3. Code possible references from the website but need to put citations.

Action List (to be attempted or completed by student by the next mandatory supervisory meeting):

1. Improve chapter 2 and chapter 3.
2. Done the whole IR and let the supervisor check.

Project Log Sheet

Third Meeting Log Sheets

(APU: Serial Number)

PLS V1.0

Project Log Sheet – Supervisory Session

Notes on use of the project log sheet:

1. This log sheet is designed for meetings of more than 15 minutes duration, of which there must be at minimum SIX (6) during the course of the project (SIX mandatory supervisory sessions).
2. The student should prepare for the supervisory sessions by deciding which question(s) he or she needs to ask the supervisor and what progress has been made (if any) since the last session, and noting these in the relevant sections of the form, effectively forming an agenda for the session.
3. A log sheet is to be brought by the STUDENT to each supervisory session.
4. The actions by the student (and, perhaps the supervisor), which should be carried out before the next session should be noted briefly in the relevant section of the form.
5. The student should leave a copy (after the session) of the Project Log Sheet with the supervisor and to the administrator at the academic counter. A copy is retained by the student to be filed in the project file.
6. It is recommended that students bring along log sheets of previous meetings together with the project file during each supervisory session.
7. The log sheet is an important deliverable for the project and an important record of a student's organisation and learning experience. The student **must** hand in the log sheets as an appendix of the final year documentation, with sheets dated and numbered consecutively.

Student's name: Tang Yu Tong **Date:** 25th November 2024 **Meeting No: 3**

Project title:
Emotion-Aware Hate Speech Detection in User-Generated Content Using Natural Language Processing

Intake: APU3F2408CS(DA)

Supervisor's name: Nicholas Teh Sek Kit **Supervisor's signature:**

Items for discussion (noted by student before mandatory supervisory meeting):

1. Checking overall IR.

Record of discussion (noted by student during mandatory supervisory meeting):

Include a flowchart or diagram summarizing the entire process, from data collection to model deployment, for clarity

Expand on real-world implications or case studies to demonstrate the potential impact of the system.

Discuss potential biases in hate speech detection due to dataset limitations or inherent model biases.

Action List (to be attempted or completed by student by the next mandatory supervisory meeting):

Update and improve IR before submitting it to Moodle.

Note: A student should make an appointment to meet his or her supervisor (via the consultation system) at least ONE (1) week prior to a mandatory supervisor session – please see document on project timelines. In the event a supervisor could not be booked for consultation, the project manager should be informed ONE (1) week prior to the session so that a meeting can be subsequently arr

Project Log Sheet

Fourth Meeting Log Sheets

(APU: Serial Number)

PLS V1.0

Project Log Sheet – Supervisory Session

Notes on use of the project log sheet:

1. This log sheet is designed for meetings of more than 15 minutes duration, of which there must be at minimum SIX (6) during the course of the project (SIX mandatory supervisory sessions).
2. The student should prepare for the supervisory sessions by deciding which question(s) he or she needs to ask the supervisor and what progress has been made (if any) since the last session, and noting these in the relevant sections of the form, effectively forming an agenda for the session.
3. A log sheet is to be brought by the STUDENT to each supervisory session.
4. The actions by the student (and, perhaps the supervisor), which should be carried out before the next session should be noted briefly in the relevant section of the form.
5. The student should leave a copy (after the session) of the Project Log Sheet with the supervisor and to the administrator at the academic counter. A copy is retained by the student to be filed in the project file.
6. It is recommended that students bring along log sheets of previous meetings together with the project file during each supervisory session.
7. The log sheet is an important deliverable for the project and an important record of a student's organisation and learning experience. The student **must** hand in the log sheets as an appendix of the final year documentation, with sheets dated and numbered consecutively.

Student's name: Tang Yu Tong **Date:** 27th February 2025 **Meeting No:** 4

Project title:
Emotion-Aware Hate Speech Detection in User-Generated Content Using Natural Language Processing

Intake: APU3F2408CS(DA)

Supervisor's name: Nicholas Teh Sek Kit **Supervisor's signature:**

Items for discussion (noted by student before mandatory supervisory meeting):

1. Model selected for FYP.
2. For the emotion dataset can I build only one model?
3. Can I change the emotion dataset due to the worst predicted result?
4. What is the minimum number of models I need to build?
5. Can I use the pre-train model?

Record of discussion (noted by student during mandatory supervisory meeting):

1. For the emotion dataset can I build only one model? Can.
2. Can I change the emotion dataset due to the worst predicted result? Yes.
3. What is the minimum number of models I need to build? Min Five.

Project Log Sheet

4. Can I use pre-train model? No.

Action List (to be attempted or completed by student by the next mandatory supervisory meeting):

1. Improve chapter 2 and chapter 3.
2. Done the whole IR and let the supervisor check.

Note: A student should make an appointment to meet his or her supervisor (via the consultation system) at least ONE (1) week prior to a mandatory supervisor session – please see document on project timelines. In the event a supervisor could not be booked for consultation, the project manager should be informed ONE (1) week prior to the session so that a meeting can be subsequently arranged.

Fifth Meeting Log Sheets

(APU: Serial Number)

PLS V1.0

Project Log Sheet – Supervisory Session

Notes on use of the project log sheet:

1. This log sheet is designed for meetings of more than 15 minutes duration, of which there must be at minimum SIX (6) during the course of the project (SIX mandatory supervisory sessions).
2. The student should prepare for the supervisory sessions by deciding which question(s) he or she needs to ask the supervisor and what progress has been made (if any) since the last session, and noting these in the relevant sections of the form, effectively forming an agenda for the session.
3. A log sheet is to be brought by the STUDENT to each supervisory session.
4. The actions by the student (and, perhaps the supervisor), which should be carried out before the next session should be noted briefly in the relevant section of the form.
5. The student should leave a copy (after the session) of the Project Log Sheet with the supervisor and to the administrator at the academic counter. A copy is retained by the student to be filed in the project file.
6. It is recommended that students bring along log sheets of previous meetings together with the project file during each supervisory session.
7. The log sheet is an important deliverable for the project and an important record of a student's organisation and learning experience. The student **must** hand in the log sheets as an appendix of the final year documentation, with sheets dated and numbered consecutively.

Student's name: Tang Yu Tong **Date:** 27th March 2025 **Meeting No: 5**

Project title:
Emotion-Aware Hate Speech Detection in User-Generated Content Using Natural Language Processing

Intake: APU3F2408CS(DA)

Supervisor's name: Nicholas Teh Sek Kit **Supervisor's signature:**

Items for discussion (noted by student before mandatory supervisory meeting):

1. Check FYP documentation Chapter 4 and Chapter 5.
2. CNN-BiLSTM model I don't do base model, only do hyperparameter tuning and fine tuning.
3. Discuss why Bi-GRU learning curve problem and Bi-LSTM Multi-task model bad result.
4. In chapter 5 model comparison, did I need compare with others research paper models?
5. What model deploy and what matrix need to show in dashboard?

Record of discussion (noted by student during mandatory supervisory meeting):

1. CNN-BiLSTM model I don't do base model, only do hyperparameter tuning and fine tuning. Ans: Okay No, problem.
2. Bi-GRU learning curve is okay like this because others PR curve, evaluation matrix are normal.
3. For multi-task bad result need clearly explain the limitation and challenges in FYP documentation.

Project Log Sheet

4. No need compare with other people models just compare what model you are done.
5. Dashboard can include evaluation matrix and graph, for system deploy can integrate some real feedback.

Action List (to be attempted or completed by student by the next mandatory supervisory meeting):

1. Settle chapter 4 and chapter 5, clearly explain the result.
2. Done model deployment, dataset dashboard and model performance dashboard.

Note: A student should make an appointment to meet his or her supervisor (via the consultation system) at least ONE (1) week prior to a mandatory supervisor session – please see document on project timelines. In the event a supervisor could not be booked for consultation, the project manager should be informed ONE (1) week prior to the session so that a meeting can be subsequently arranged.

Sixth Meeting Log Sheets

(APU: Serial Number)

PLS V1.0

Project Log Sheet – Supervisory Session

Notes on use of the project log sheet:

1. This log sheet is designed for meetings of more than 15 minutes duration, of which there must be at minimum SIX (6) during the course of the project (SIX mandatory supervisory sessions).
2. The student should prepare for the supervisory sessions by deciding which question(s) he or she needs to ask the supervisor and what progress has been made (if any) since the last session, and noting these in the relevant sections of the form, effectively forming an agenda for the session.
3. A log sheet is to be brought by the STUDENT to each supervisory session.
4. The actions by the student (and, perhaps the supervisor), which should be carried out before the next session should be noted briefly in the relevant section of the form.
5. The student should leave a copy (after the session) of the Project Log Sheet with the supervisor and to the administrator at the academic counter. A copy is retained by the student to be filed in the project file.
6. It is recommended that students bring along log sheets of previous meetings together with the project file during each supervisory session.
7. The log sheet is an important deliverable for the project and an important record of a student's organisation and learning experience. The student **must** hand in the log sheets as an appendix of the final year documentation, with sheets dated and numbered consecutively.

Student's name: Tang Yu Tong **Date:** 17th April 2025 **Meeting No:** 6

Project title:
Emotion-Aware Hate Speech Detection in User-Generated Content Using Natural Language Processing

Intake: APU3F2408CS(DA)

Supervisor's name: Nicholas Teh Sek Kit **Supervisor's signature:** 

Items for discussion (noted by student before mandatory supervisory meeting):

1. Check system interface.
2. FYP final presentation time slot.

Record of discussion (noted by student during mandatory supervisory meeting):

1. User interface already okay no need to change anything, focus on prepare FYP presentation.

Action List (to be attempted or completed by student by the next mandatory supervisory meeting):

1. Done all FYP documentation.
2. Done presentation poster, presentation slide, presentation video.

Note: A student should make an appointment to meet his or her supervisor (via the consultation system) at least ONE (1) week prior to a mandatory supervisor session – please see document on project timelines. In the event a supervisor could not be booked for consultation, the project manager should be informed ONE (1) week prior to the session so that a meeting can be subsequently arranged.

Appendix D: Poster

Emotion-Aware Hate Speech Detection in User-Generated Content Using Natural Language Processing

TANG YU TONG (TP069018) - APU3F2408CS(DA)
BACHELOR OF SCIENCE (HONOURS) IN COMPUTER SCIENCE WITH A SPECIALISM IN DATA ANALYTICS
SUPERVISED BY: MR. TS. NICHOLAS TEH SEK KIT
SECOND MARKER: MR. RAHEEM MAFAS

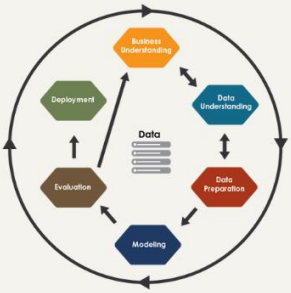
INTRODUCTION

With the rise of online platforms, hate speech has become a growing concern. This project explores whether **incorporating emotional features into NLP models** can enhance the detection of hate speech. By comparing traditional machine learning, deep learning, and emotion-integrated models, aligning with **SDG 16: Peace, Justice, and Strong Institutions.**

PROBLEM STATEMENTS

- Imbalanced Distribution of Hate Speech
- Challenges of Keyword-Based Approaches in Identifying Hate Speech
- Complexity of Hate Speech in Text Analysis (Replacing letters with numeric characters. Exp. h@te)

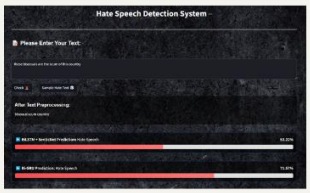
METHODOLOGY : CRISP-DM

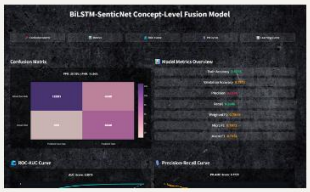


OBJECTIVES

- To **investigate** and obtain comprehensive **datasets** for emotion feature extraction and hate speech detection to ensure effective model training.
- To **develop, train, and deploy** an advanced NLP model capable of detecting hate speech by **incorporating emotion analysis,** overcoming the limitations of traditional keyword-based and basic text analysis methods, and **implementing in a local web page** for practical use.
- To **evaluate the performance** of the developed model

MODEL DEPLOYMENT





CONCLUSION

In order to improve the detection accuracy, the project explored two emotion fusion methods (concept-level fusion & intergrate emotion dataset) by integrating emotion concept-level features into the deep learning model, achieved a validation accuracy of 79%, but it would increase the false positive rate. Multi-task model that integrates emotion datasets faces the MTL problem caused by gradient conflicts, which affects the model's stability and generalization ability.

SELECTED MODEL

- XGBoost
- **Bi-LSTM**
- **Bi-GRU**
- CNN-LSTM
- MTL-BiLSTM
- **Bi-LSTM-SenticNet**

LIMITATION

- High false positives from emotion-based detection
- Poor generalization
- Gradient conflicts in multi-task learning
- Lacks multilingual support

Figure 189: FYP Poster

Appendix E: Gantt Chart

Gantt Chart of Final Year Project Investigation Report

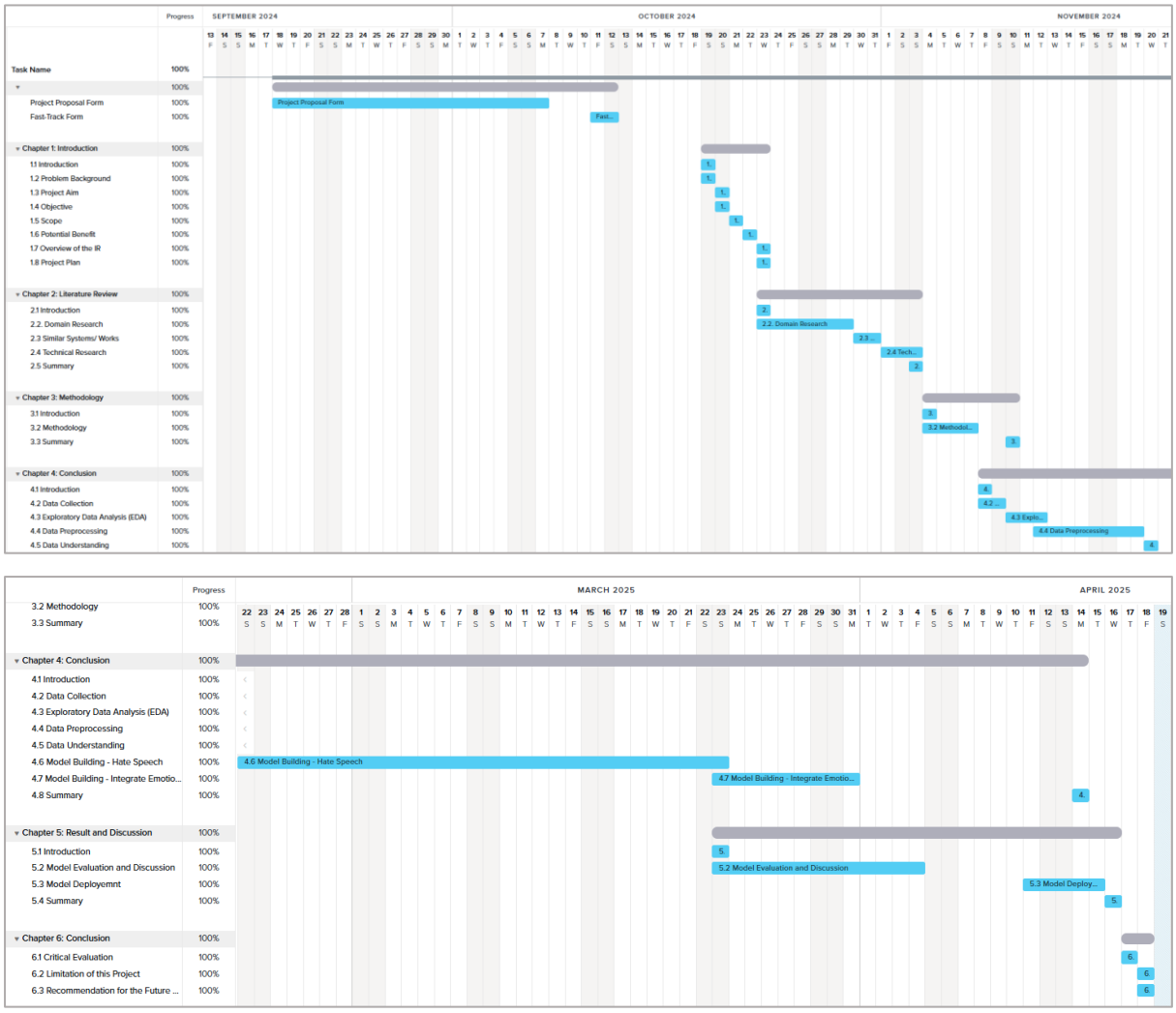


Figure 190: Project IR Gantt Chart

Appendix F: Sample Code Implementation

Main Web Page

```
import streamlit as st
import base64
import random
import pandas as pd
import streamlit as st
import numpy as np
from datetime import datetime
from datasetDashboard import run_dataset_dashboard
from modelDashboard import run_model_dashboard
from predictHate import text_preprocessing,predict_hate

st.set_page_config(layout="wide")

# ------------------------------------------------------------------------------------------------------------------------

# Backgroud Image
def image_to_base64(image_path):
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read()).decode()
image_path = r"./image/blue-background-7470781_1280.jpg"
image_base64 = image_to_base64(image_path)

# CSS Design
st.markdown(···

st.markdown("""···
```

```
# Web Page Content----------------------------------------------------------------------------------------------------
@st.cache_data
def load_sample_data():
    df = pd.read_csv("Model Deploy Sample Data.csv")
    data = df[df['label'] == 1]['text'].tolist()
    return data

sample_texts = load_sample_data()

st.sidebar.title("Select a Page")
main_page = st.sidebar.selectbox("", [
    "Hate Speech Predict Page",
    "Hate Speech Dataset Dashboard",
    "Model Preformances Dashboard"
])

if main_page == "Hate Speech Predict Page":
    st.markdown(""" <h1 style="text-align: center; font-size: 36px;">Hate Speech Detection System</h1>
                <hr style="height:2px; border:none; background-color:#ccc;" /> """, unsafe_allow_html=True)
    # Initialize session states
    if 'user_input' not in st.session_state:
        st.session_state.user_input = ""
    if 'history' not in st.session_state:
        st.session_state.history = []

    st.markdown("""<h3 style=" font-family: 'Arial', sans-serif; font-size: 24px; color: white; font-weight: 600; margin-bottom: 10px;
    "> Please Enter Your Text: </h3> """, unsafe_allow_html=True)
    user_input = st.text_area("", value=st.session_state.user_input)
```

```
col1, col2 = st.columns([1, 10])
with col1:
    check = st.button("  Check 🚨")
with col2:
    if st.button("  Sample Hate Text 🎲 "):
        if sample_texts:
            st.session_state.user_input = random.choice(sample_texts)
            st.rerun()
        else:
            st.warning("No hate speech samples found in the dataset.")
if check:
    if user_input.strip():
        try:
            # After Text preprocessing
            processed_text = text_preprocessing(user_input)
            st.markdown("""<div style=" background-color: rgba(0, 0, 0, 0.5); padding: 15px; border-radius: 12px; margin-bottom: 20px;">
                                <h3 style="margin-top: 0; font-size: 20px;font-family: 'Arial', sans-serif;">After Text Preprocessing: </h3>
                                <p style="color: white;">{}</p>
                            </div> """.format(processed_text), unsafe_allow_html=True)

            result_1, result_2, prob_1, prob_2 = predict_hate(processed_text)

            # Add to history
            history_entry = {
                "sentence": user_input,
                "BiLSTM-SenticNet": f"{result_1}",
                "Prob_1": f"{round(prob_1*100, 2)}%",
                "Bi-GRU": f"{result_2}",
                "Prob_2": f"{round(prob_2*100, 2)}%",
                "timestamp": datetime.now().strftime("%H:%M:%S")
            }
            st.session_state.history.insert(0, history_entry)
            if len(st.session_state.history) > 10:
                st.session_state.history = st.session_state.history[:10]
```

```
            # Display prediction
            if result_1 == "Hate Speech":
                border_class_1 = "hate-border"
            else:
                border_class_1 = "nonhate-border"
            if result_2 == "Hate Speech":
                border_class_2 = "hate-border"
            else:
                border_class_2 = "nonhate-border"

            percent_1 = round(prob_1 * 100, 2)
            percent_2 = round(prob_2 * 100, 2)
            color_class_1 = "progress-hate" if result_1 == "Hate Speech" else "progress-nonhate"
            color_class_2 = "progress-hate" if result_2 == "Hate Speech" else "progress-nonhate"

            st.markdown(f"""
            <div class="result-container {border_class_1}" style="padding: 15px; border-radius: 10px; margin-bottom: 20px; background-color: rgba(0, 0, 0, 0.5);">
                <p style="margin: 0; font-size: 18px;">
                    ▶️ <b>BiLSTM + SenticNet Prediction:</b> <span style="color:white;">{result_1}</span>
                    <span style="float:right;"><b>{percent_1}%</b></span>
                </p>
                <div class="progress-bar" style="margin-top: 8px;">
                    <div class="progress {color_class_1}" style="width: {percent_1}%; height: 20px;">
                    </div>
                </div>
            </div>
            """, unsafe_allow_html=True)

            st.markdown(f"""
            <div class="result-container {border_class_2}" style="padding: 15px; border-radius: 10px; margin-bottom: 30px; background-color: rgba(0, 0, 0, 0.5);">
                <p style="margin: 0; font-size: 18px;">
                    ▶️ <b>Bi-GRU Prediction:</b> <span style="color:white;">{result_2}</span>
                    <span style="float:right;"><b>{percent_2}%</b></span>
                </p>
                <div class="progress-bar" style="margin-top: 8px;">
                    <div class="progress {color_class_2}" style="width: {percent_2}%; height: 20px;">
                    </div>
                </div>
            </div>
            """, unsafe_allow_html=True)
```

```
            except Exception as e:
                st.warning(f"Error 1: {e}")
        else:
            st.warning("Please enter text 💬")

    # History Table
    st.divider()
    st.markdown("""
    <h3 style=" font-family: 'Arial', sans-serif; font-size: 24px; color: white; font-weight: 600; margin-bottom: 10px; ">📄 Prediction History (Last 10):</h3>
    """, unsafe_allow_html=True)

    if st.session_state.history:
        history_df = pd.DataFrame(st.session_state.history)
        history_df.index = np.arange(1, len(history_df) + 1 )

        st.dataframe(
            history_df[["sentence", "BiLSTM-SenticNet", "Prob_1","Bi-GRU","Prob_2"]],
            column_config={
                "sentence": "Input Text",
                "BiLSTM-SenticNet": "BiLSTM-SenticNet",
                "Prob_1": "Probability",
                "Bi-GRU": "Bi-GRU",
                "Prob_2": "Probability",
            },
            use_container_width=True,
            hide_index=False
        )
        if st.button("Clear History"):
            st.session_state.history = []
            st.rerun()
    else:
        st.info("No prediction history yet.")
```

```
elif main_page == "Hate Speech Dataset Dashboard":
    run_dataset_dashboard()

elif main_page == "Model Preformances Dashboard":
    model_option = st.sidebar.radio("Select a Model", [
        "Bi-GRU", "BiLSTM-SenticNet", "Bi-LSTM" ])
    if model_option == "Bi-GRU":
        run_model_dashboard("Bi-GRU Hate Base Deep Learning Model","GRU")
    elif model_option == "BiLSTM-SenticNet":
        run_model_dashboard("BiLSTM-SenticNet Concept-Level Fusion Model ","SenticNet")
    elif model_option == "Bi-LSTM":
        run_model_dashboard("Bi-LSTM Hate Base Deep Learning Model","LSTM")
```

Predict Hate Speech Function.py

```
import tensorflow as tf
from tensorflow.keras.models import load_model
import pickle
import tensorflow.keras.backend as K
from tensorflow.keras.saving import register_keras_serializable
from nltk.tokenize import word_tokenize
from senticnet.senticnet import SenticNet
import numpy as np
import re
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
import emoji
from tensorflow.keras.preprocessing.sequence import pad_sequences

def text_preprocessing(text):
    text = text.lower()
    text = re.sub(r'(http|https|www)\S+', '', text)
    html_tags_pattern = r'<.*?>'
    text = re.sub(html_tags_pattern, '',text)
    text = re.sub(r'\S+html\b', '',text)
    text = text = emoji.demojize(text)
    text = text = re.sub(r':\w+:', '', text)

    replace_dict = {
        '@': 'a',
        '$': 's',
        '$$':'ss',
        '0': 'o',
        '3': 'e',
        '1': 'i',
        '5': 's',
        '7': 't',
        '4': 'a',
        '9' : 'g',
    }
    def replace_symbols(text):
        for symbol, letter in replace_dict.items():
            text = re.sub(rf'{re.escape(symbol)}', letter, text)
            print(text)
        return text
    text = replace_symbols(text)
```

```
abbreviations = {
    "it's":"it is",
    "we're":"were are",
    "let's":"let us",
    "i'll":"i will",
}
def replace_abbreviations(text):
    text = str(text)
    for abbr, full_form in abbreviations.items():
        text = re.sub(r'\b' + re.escape(abbr) + r'\b', full_form, text)
    return str(text)
text = replace_abbreviations(text)

text = text = re.sub(r'[^A-Za-z0-9\s]', ' ', text)
text = text = re.sub(r'\s+', ' ', text).strip()
text = text = re.sub(r'(.)\1{2,}', r'\1\1', text)
```

```
abbreviations = {
    "auser" : "",
    "werent":"were not","arent": "are not",
    "isnt": "is not",
    "cant": "can not",
    "shes": "she is","hes": "he is",
    "youre": "you are",
    "youll": "you will",
    "youve": "you have",
    "weve": "we have",
    "yall":"you all",
    "theyre": "they are",
    "theyve": "they have",
    "doesnt": "does not",
    "dont":"do not",
    "didnt": "did not",
    "wont": "will not",
    "wouldnt": "would not",
    "shouldnt": "should not",
    "couldnt": "could not",
    "im": "i am",
    "iam": "i am",
    "ive": "i have",
    "id": "i would",
    "wth":"what the heal", "wtf":"what the fuck",
    "fk":"fuck", "f**k":"fuck","fu*k":"fuck", "f*ck":"fuck","fck":"fuck","fcking":"fucking","fking":"fucking",
    "cuz":"because", "bcuz":"because","becuz":"because",
    "bihday":"birthday",
    "etc":"et cetera",
    "selfie":"self portrait photograph",
    "lol":"laughing out loud",
    "lmao":"laughing my ass off",
    "forex":" foreign exchange",
    "lgbt":"transgender",
    "blm":"black lives matter",
    "obama":"Barack Obama",
    "omg":"oh my god",
    "ppl":"people",
    "fathersday":"father day",
}
def replace_abbreviations(text):
    text = str(text)
    for abbr, full_form in abbreviations.items():
        text = re.sub(r'\b' + re.escape(abbr) + r'\b', full_form, text)
    return str(text)
text = replace_abbreviations(text)

text = nltk.tokenize.word_tokenize(text)
```

```
stopTokens = nltk.corpus.stopwords.words("english")
stopTokens.remove('not')
stopTokens.remove('no')
def removeStopWord(words):
    return [word for word in words if word.lower() not in stopTokens]
text = removeStopWord(text)

def get_pos_tagging(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V": wordnet.VERB, "R": wordnet.ADV} #need this for wordnet cuz wordnet only have 4 postag
    return tag_dict.get(tag, wordnet.NOUN)
lemmatizer = WordNetLemmatizer()
def lemmatize_text(text):
    lemmatized_words = [lemmatizer.lemmatize(word, get_pos_tagging(word)) for word in text]
    return ' '.join(lemmatized_words)
text = lemmatize_text(text)
return text
```

```python
# Predict Hate Function
MAX_SEQUENCE_LENGTH = 100
NUM_SENTIC_FEATURES = 24

@register_keras_serializable(package="CustomLoss")
def focal_loss(y_true, y_pred, gamma=2.0, alpha=0.55):
    epsilon = K.epsilon()
    y_pred = K.clip(y_pred, epsilon, 1.0 - epsilon)
    pt = tf.where(K.equal(y_true, 1), y_pred, 1 - y_pred)
    alpha_factor = K.ones_like(y_true) * alpha
    alpha_t = tf.where(K.equal(y_true, 1), alpha_factor, 1 - alpha_factor)
    cross_entropy = -K.log(pt)
    weight = alpha_t * K.pow((1 - pt), gamma)
    loss = weight * cross_entropy
    return K.mean(loss, axis=1)

@register_keras_serializable(package="CustomLoss")
def focal_loss_1(y_true, y_pred, gamma=2.0, alpha=0.55):
    epsilon = K.epsilon()
    y_pred = K.clip(y_pred, epsilon, 1.0 - epsilon)
    pt = tf.where(K.equal(y_true, 1), y_pred, 1 - y_pred)
    alpha_factor = K.ones_like(y_true) * alpha
    alpha_t = tf.where(K.equal(y_true, 1), alpha_factor, 1 - alpha_factor)
    cross_entropy = -K.log(pt)
    weight = alpha_t * K.pow((1 - pt), gamma)
    loss = weight * cross_entropy
    return K.mean(loss, axis=1)

# Load Model and Tokenizer
model = load_model(
    r"../Model Code/Saved Model Hate/EmoHate/4_BiLSTM_FastText_SenticNet_FocalLoss_Model.keras",
    custom_objects={'focal_loss_fixed': focal_loss}
)
model_GRU = load_model(
    r"../Model Code/Saved Model Hate/GRU/6_GRU_Hate_FineTuning_Model_1.keras",
    custom_objects={'focal_loss_1': focal_loss_1}
)
with open('tokenizer.pkl', 'rb') as handle:
    tokenizer = pickle.load(handle)
```

```python
# Sentic Feature Extractor
def extract_sentic_features(text, tokenizer, max_len):
    sn = SenticNet()
    sentic_matrix = np.zeros((max_len, NUM_SENTIC_FEATURES), dtype=np.float32)
    tokens = word_tokenize(text)
    token_indices = tokenizer.texts_to_sequences([tokens])[0]
    for i, token_idx in enumerate(token_indices[:max_len]):
        word = tokenizer.index_word.get(token_idx, '')
        try:
            sentic_values = [float(s) for s in sn.concept(word)[:NUM_SENTIC_FEATURES]]
            sentic_matrix[i] = sentic_values
        except:
            pass  # OOV or missing sentic entry → all zeros
    return sentic_matrix

# Prediction Function
def predict_hate(text):
    sequence = tokenizer.texts_to_sequences([text])
    padded_sequence = pad_sequences(sequence, maxlen=MAX_SEQUENCE_LENGTH, dtype="float32")

    sentic_input = np.array([extract_sentic_features(text, tokenizer, MAX_SEQUENCE_LENGTH)])

    # Model 1 prediction
    predictions = model.predict([padded_sequence, sentic_input])
    prob_1 = float(predictions.flatten()[0])
    hate_label = int(prob_1 > 0.5)
    result1 = "Hate Speech" if hate_label == 1 else "Non-Hate Speech"

    # Model 2 prediction
    predictions_2 = model_GRU.predict([padded_sequence, sentic_input])
    prob_2 = float(predictions_2.flatten()[0])
    hate_label_2 = int(prob_2 > 0.5)
    result2 = "Hate Speech" if hate_label_2 == 1 else "Non-Hate Speech"

    return result1, result2, prob_1, prob_2
```

Dataset Dashboard.py

```
import re
import pandas as pd
import numpy as np
import streamlit as st
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import words, wordnet
from sklearn.feature_extraction.text import CountVectorizer
from wordcloud import WordCloud
from collections import Counter

def run_dataset_dashboard():
    # nltk.download('punkt')
    nltk.download('wordnet')
    nltk.download('words')

    file_path = r"..\Pre_Hate_Dataset\20_Hate_Final.csv"
    data = pd.read_csv(file_path, encoding='ISO-8859-1', low_memory=False)

    with st.container():
        st.markdown("""
            <h1 style="text-align: center; font-size: 36px;">Hate Speech Dataset Dashboard</h1>
            <hr style="height:2px; border:none; background-color:#ccc;" />
        """, unsafe_allow_html=True)
```

```
    with st.container():
        st.markdown("""
            <h1 style="text-align: center; font-size: 36px;">Hate Speech Dataset Dashboard</h1>
            <hr style="height:2px; border:none; background-color:#ccc;" />
        """, unsafe_allow_html=True)

        col1, col2 = st.columns(2)
        with col1:
            # 1 Hate & Non-Hate Distribution
            st.subheader("📊 Hate & Non-Hate Distribution")
            def hate_NoHate_percenatge(label):
                if label == 1:
                    return "Hate"
                elif label == 0:
                    return "Non-Hate"
                return "Unknown"
            data['label'] = data['label'].apply(hate_NoHate_percenatge)
            label_counts = data['label'].value_counts().reset_index()
            label_counts.columns = ['Label', 'Count']

            fig_pie = px.pie(label_counts, names='Label', values='Count',
                             color='Label', color_discrete_map={'Hate': '#211C84', 'Non-Hate': '#7A73D1'},
                             hole=0.3)
            fig_pie.update_layout(
                paper_bgcolor='rgba(0,0,0,0.5)',   # chart transparent background
                legend=dict(
                    font=dict(size=18, family='Arial',),
                    orientation="h",
                    x=0.5,              # legend center
                    xanchor='center'  # pie chart center
                )
            )
            fig_pie.update_traces(textinfo='percent+label')
            fig_pie.update_traces(
                textinfo='percent+label',
                textfont=dict(size=18, family='Arial')
            )
            st.plotly_chart(fig_pie, use_container_width=True)
```

```
        with col2:
            # 2. Enhanced Word Cloud
            st.subheader("☁️ Word Cloud (Hate only)", divider="blue")
            # Filter only hate speech for word cloud
            hate_text = ' '.join(data[data['label'] == 'Hate']['text'].astype(str))
            words_1 = word_tokenize(hate_text)
            word_counts = Counter(words_1)

            wordcloud = WordCloud(
                width=800,
                height=500,
                background_color=None,
                mode='RGBA',
                max_words=150,
                colormap='Set3',
                prefer_horizontal=0.9,
                min_font_size=8,
                max_font_size=150,
                relative_scaling=0.5,
                font_path=None,
                random_state=42,
                collocations=False,
            ).generate_from_frequencies(word_counts)

            fig_wc, ax_wc = plt.subplots(figsize=(12, 8), facecolor='none') #facecolor -> Background Transparent
            ax_wc.imshow(wordcloud, interpolation='bilinear')
            ax_wc.axis('off')
            plt.tight_layout(pad=0)
            fig_wc.patch.set_alpha(0)

            st.pyplot(fig_wc, use_container_width=True)
```

```
col1, col2 = st.columns(2)
with col1:
    # 3 Bigrams
    st.subheader("🔗 Most Frequency Bigrams")
    label_option = st.selectbox( "Choose the label to show bigrams:", options=["Hate", "Non-Hate"])

    data_label_1 = data[data['label'] == label_option]
    vectorizer = CountVectorizer(ngram_range=(2, 2))
    X = vectorizer.fit_transform(data_label_1['text'].astype(str))
    bigram_counts = np.array(X.sum(axis=0)).flatten()
    bigram_freq = pd.DataFrame({
        'bigram': vectorizer.get_feature_names_out(),
        'frequency': bigram_counts
    }).sort_values(by='frequency', ascending=False).head(20)

    fig_bigram = go.Figure()
    fig_bigram.add_trace(go.Bar(
        x=bigram_freq['frequency'],
        y=bigram_freq['bigram'],
        orientation='h',
        marker=dict(color='#48A6A7', line=dict(color='rgba(0,0,0,0.1)', width=1)),
        text=bigram_freq['frequency'],
        textposition='auto',
        textfont=dict(family="Arial", size=14, color="white", weight="bold")  # Adding font style here
    ))

    fig_bigram.update_layout(
        xaxis_title='Frequency',
        yaxis_title='Bigram',
        yaxis=dict(autorange="reversed"),  # Most Frequent on top
        paper_bgcolor='rgba(0,0,0,0.5)',
        font=dict(size=14, family='Arial'),
        margin=dict(t=60, l=80, r=40, b=50),
        height=700
    )
    st.plotly_chart(fig_bigram, use_container_width=True)
```

```
with col2:
    # 4 Unknown Word
    st.subheader("🔍 Top 20 Unknown Words")
    label_option = st.selectbox( "Choose a label:", options=["Before Text Preprocessing", "After Text Preprocessing"])

    if label_option == "Before Text Preprocessing":
        file_path = r"..\Pre_Hate_Dataset\ForDashBoardUsed.csv"
        data = pd.read_csv(file_path, encoding='ISO-8859-1', low_memory=False)

    combined_vocab = set(words.words()) | set(wordnet.words())
    def find_unknown_words_combined(text):
        if not isinstance(text, str): return []
        words_in_text = re.findall(r'\b\w+\b', text.lower())
        return [word for word in words_in_text if word not in combined_vocab]

    data['unknown_words'] = data['text'].apply(find_unknown_words_combined)
    all_unknown_words = [word for word_list in data['unknown_words'] for word in word_list]
    unknown_word_freq = Counter(all_unknown_words)
    freq_df = pd.DataFrame(unknown_word_freq.items(), columns=['word', 'frequency']).sort_values(by='frequency', ascending=False)
    top_unknown = freq_df.head(20)

    # Horizontal bar chart
    fig_unknown = go.Figure()
    fig_unknown.add_trace(go.Bar(
        x=top_unknown['frequency'],
        y=top_unknown['word'],
        orientation='h',
        marker=dict(color='#A97DCE', line=dict(color='rgba(0,0,0,0.1)', width=1)),
        text=top_unknown['frequency'],
        textposition='auto',
        textfont=dict(family="Arial", size=14, color="white")
    ))

    fig_unknown.update_layout(
        xaxis_title='Frequency',
        yaxis_title='Word',
        yaxis=dict(autorange="reversed"),
        paper_bgcolor='rgba(0,0,0,0.5)',
        font=dict(size=14, family='Arial'),
        margin=dict(t=60, l=80, r=40, b=50),
        height=700
    )
    st.plotly_chart(fig_unknown, use_container_width=True)
```

Model Performance Dashboard.py

```
import streamlit as st
from modelGraph import confusion_matrix ,PR_curve, learning_curve, ROC_curve

def get_color_class(score):
    score = float(score)
    if score >= 0.80:
        return "green"
    elif score >= 0.70:
        return "orange"
    else:
        return "red"

def run_model_dashboard(title,model):
    # CSS design
    st.markdown(f"""···

    if model == "GRU":
        metrics = {
            "Train Accuracy": {"value": "0.8283", "eval": ""},
            "Test Accuracy": {"value": "0.7993", "eval": ""},
            "Precision": {"value": "0.6353", "eval": ""},
            "Recall": {"value": "0.8490", "eval": ""},
            "Weighted F1": {"value": "0.8054", "eval": ""},
            "Micro F1": {"value": "0.7993", "eval": ""},
            "Macro F1": {"value": "0.7841", "eval": ""}
        }
    elif model == "LSTM":
        metrics = {···
    elif model == "SenticNet":
        metrics = {···
```

```
    st.markdown(f"<h1 style='text-align: center;'>{title}</h1>", unsafe_allow_html=True)
    st.divider()
    # Flip Card - Evaluation Matrix
    col1, col2, col3, col4, col5 = st.columns(5)
    with col1:
        st.markdown('<div class="nav-card"><a href="#confusion-matrix">📌 Confusion Matrix</a></div>', unsafe_allow_html=True)
    with col2:
        st.markdown('<div class="nav-card"><a href="#model-metrics-overview">📊 Metrics</a></div>', unsafe_allow_html=True)
    with col3:
        st.markdown('<div class="nav-card"><a href="#roc-auc-curve">🌊 ROC Curve</a></div>', unsafe_allow_html=True)
    with col4:
        st.markdown('<div class="nav-card"><a href="#precision-recall-curve">🎯 PR Curve</a></div>', unsafe_allow_html=True)
    with col5:
        st.markdown('<div class="nav-card"><a href="#learning-curve">📈 Learning Curve</a></div>', unsafe_allow_html=True)
    st.divider()
```

```
    with st.container():
        col1, col2 = st.columns(2)
        # Confusion Matrix
        with col1:
            st.markdown("<div id='confusion-matrix'></div>", unsafe_allow_html=True)
            st.subheader("Confusion Matrix")
            if model == "GRU":
                confusion_matrix(1169, 6572,13108, 3772, 22.35,15.10)
            elif model == "LSTM":
                confusion_matrix(1219, 6522,13098, 3782, 22.41,15.75)
            elif model == "SenticNet":
                confusion_matrix(893, 6848,12531, 4349, 25.76,11.54)
        # Matrix
        with col2:
            st.markdown("<div id='model-metrics-overview'></div>", unsafe_allow_html=True)
            st.subheader("📊 Model Metrics Overview")
            with st.container():
                metric_names = ["Train Accuracy", "Test Accuracy", "Precision", "Recall", "Weighted F1", "Micro F1", "Macro F1"]
                for metric in metric_names:
                    st.markdown(f"""
                        <div class="flip-card">
                            <div class="flip-card-inner">
                                <div class="flip-card-front">
                                    <div class="metric-title">{metric}</div>
                                    <div class="metric-value {get_color_class(metrics[metric]['value'])}">{metrics[metric]['value']}</div>
                                </div>
                                <div class="flip-card-back">
                                    <div class="back-text">{metrics[metric]['eval']}</div>
                                </div>
                            </div>
                        </div>
                    """, unsafe_allow_html=True)
                    st.divider()
```

```
    col1, col2 = st.columns(2)
    # ROC Curve
    with col1:
        st.markdown("<div id='roc-auc-curve'></div>", unsafe_allow_html=True)
        st.subheader("🌊 ROC-AUC Curve")
        if model == "GRU":
            ROC_curve(r"./json/Bi-GRU_ROC_Curve.json")
        elif model == "LSTM":
            ROC_curve(r"./json/Bi-LSTM_ROC_Curve.json")
        elif model == "SenticNet":
            ROC_curve(r"./json/Bi-LSTM_SenticNet_ROC_Curve.json")
    # PR Curve
    with col2:
        st.markdown("<div id='precision-recall-curve'></div>", unsafe_allow_html=True)
        st.subheader("🎯 Precision-Recall Curve")
        if model == "GRU":
            PR_curve(r"./json/Bi-GRU_PR_Curve.json")
        elif model == "LSTM":
            PR_curve(r"./json/Bi-LSTM_PR_Curve.json")
        elif model == "SenticNet":
            PR_curve(r"./json/Bi-LSTM_SenticNet_PR_Curve.json")
    # Learning Curve
    st.markdown("<div id='learning-curve'></div>", unsafe_allow_html=True)
    if model == "GRU":
        learning_curve(r"../Model Code/Saved Model Hate/GRU/6_GRU_Hate_FineTuning_History_1.pkl",3)
    elif model == "LSTM":
        learning_curve(r"../Model Code/Saved Model Hate/LSTM/FineTune_1_LSTM_FastText__History.pkl",8)
    elif model == "SenticNet":
        learning_curve(r"../Model Code/Saved Model Hate/EmoHate/4_BiLSTM_FastText_SenticNet_FocalLoss_History.pkl",7)
```

Model Graph Function.py

```
import plotly.graph_objects as go
import numpy as np
import streamlit as st
import json
import joblib

def confusion_matrix(TN,TP,FN,FP,FPR,FNR):
    cm = np.array([[TN, TP],[FN, FP]])
    fig = go.Figure(data=go.Heatmap(
        z=cm,
        x=['Predicted: Non-Hate', 'Predicted: Hate'],
        y=['Actual: Hate','Actual: Non-Hate'],
        colorscale='Purpor',
        showscale=True, # Color Bar
        opacity=0.8
    ))
    annotations = []
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            annotations.append(
                dict(
                    text=str(cm[i][j]),
                    x=['Predicted: Non-Hate', 'Predicted: Hate'][j],
                    y=['Actual: Hate','Actual: Non-Hate'][i],
                    font=dict(color='white', size=15, family='Arial Black'),
                    showarrow=False
                )
            )
    FPR = FPR
    FNR = FNR
    fig.update_layout(
        title=dict(
            text=f"FPR: {FPR:.2f}% | FNR: {FNR:.2f}%",
            font=dict(size=15, family='Arial'),
            x=0.5,
            xanchor='center'
        ),
        annotations=annotations,
        paper_bgcolor='rgba(0,0,0,0.5)',
        xaxis=dict(showgrid=False),
        yaxis=dict(showgrid=False),
        margin=dict(l=50, r=50, t=80, b=50)
    )
    st.plotly_chart(fig, use_container_width=True)
```

```python
def PR_curve(json_path):
    with open(json_path, "r", encoding="utf-8") as f: pr_data = json.load(f)

    precision = pr_data["precision"]
    recall = pr_data["recall"]
    pr_auc = pr_data["PR_AUC"]
    baseline= pr_data["baseline"]

    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=recall,
        y=precision,
        mode='lines',
        name=f'PR Curve (AUC = {pr_auc:.4f})',
        line=dict(color='orange', width=3)
    ))

    fig.add_trace(go.Scatter(
        x=[0, 1],
        y=[baseline, baseline],
        mode='lines',
        name='Random Guess',
        line=dict(color='white', width=2, dash='dash')
    ))

    fig.update_layout(
        xaxis=dict(title='Recall', range=[0, 1]),
        yaxis=dict(title='Precision', range=[0, 1.02]),
        legend=dict(
            font=dict(size=12),
            orientation="h",
            x=0.5,
            xanchor='center',
            y=-0.2
        ),
        paper_bgcolor='rgba(0,0,0,0.5)',
        margin=dict(l=40, r=40, t=80, b=80),
        title=dict(
            text=f"PR-AUC Score: {pr_auc:.4f}",
            font=dict(size=15, family='Arial'),
            x=0.5,
            xanchor='center'
        )
    )
    st.plotly_chart(fig, use_container_width=True)
```

```python
def ROC_curve(json_path):
    with open(json_path, "r", encoding="utf-8") as f:
        roc_data = json.load(f)
    fpr = roc_data["fpr"]
    tpr = roc_data["tpr"]
    roc_auc = roc_data["roc_auc"]

    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=fpr,
        y=tpr,
        mode='lines',
        name=f'ROC Curve (AUC = {roc_auc:.4f})',
        line=dict(color='cyan', width=3)
    ))

    fig.add_trace(go.Scatter(
        x=[0, 1],
        y=[0, 1],
        mode='lines',
        name='Random Guess',
        line=dict(color='white', width=2, dash='dash')
    ))

    fig.update_layout(
        xaxis=dict(title='False Positive Rate', range=[0, 1]),
        yaxis=dict(title='True Positive Rate', range=[0, 1.02]),
        legend=dict(
            font=dict(size=12),
            orientation="h",
            x=0.5,
            xanchor='center',
            y=-0.2
        ),
        paper_bgcolor='rgba(0,0,0,0.5)',
        margin=dict(l=40, r=40, t=80, b=80),
        title=dict(
            text=f"AUC Score: {roc_auc:.4f}",
            font=dict(size=15, family='Arial'),
            x=0.5,
            xanchor='center'
        )
    )
    st.plotly_chart(fig, use_container_width=True)
```

```python
def learning_curve(history_path, patience):
    history = joblib.load(history_path)
    accuracy = history.get('accuracy', [])[:patience]
    val_accuracy = history.get('val_accuracy', [])[:patience]
    loss = history.get('loss', [])[:patience]
    val_loss = history.get('val_loss', [])[:patience]
    epochs = list(range(1, len(accuracy) + 1))

    col1, col2 = st.columns(2)
    # Accuracy Curve
    with col1:
        st.subheader("🦇 Accuracy Curve")
        fig_acc = go.Figure()
        fig_acc.add_trace(go.Scatter(x=epochs, y=accuracy, mode='lines+markers', name='Train Accuracy',
                                     line=dict(color='lightgreen', width=3)))
        fig_acc.add_trace(go.Scatter(x=epochs, y=val_accuracy, mode='lines+markers', name='Test Accuracy',
                                     line=dict(color='red', width=3)))

        for i in range(len(epochs)):
            fig_acc.add_annotation(x=epochs[i], y=accuracy[i], text=f"{accuracy[i]:.4f}", showarrow=False,
                                   font=dict(size=10, color="lightgreen"), yshift=10)
            fig_acc.add_annotation(x=epochs[i], y=val_accuracy[i], text=f"{val_accuracy[i]:.4f}", showarrow=False,
                                   font=dict(size=10, color="red"), yshift=-10)

        fig_acc.update_layout(
            xaxis_title="Epoch",
            yaxis_title="Accuracy",
            xaxis=dict(tickmode='linear'),
            # yaxis=dict(range=[0.6, 0.9]),
            legend=dict(orientation="h", x=0.5, xanchor='center', y=-0.2),
            paper_bgcolor='rgba(0,0,0,0.5)',
            margin=dict(l=40, r=40, t=80, b=80)
        )
        st.plotly_chart(fig_acc, use_container_width=True)
```

```python
    # Loss Curve
    with col2:
        st.subheader("🦇 Loss Curve")

        fig_loss = go.Figure()
        fig_loss.add_trace(go.Scatter(x=epochs, y=loss, mode='lines+markers', name='Train Loss',
                                      line=dict(color='lightblue', width=3)))
        fig_loss.add_trace(go.Scatter(x=epochs, y=val_loss, mode='lines+markers', name='Test Loss',
                                      line=dict(color='red', width=3)))

        for i in range(len(epochs)):
            fig_loss.add_annotation(x=epochs[i], y=loss[i], text=f"{loss[i]:.4f}", showarrow=False,
                                    font=dict(size=10, color="lightblue"), yshift=10)
            fig_loss.add_annotation(x=epochs[i], y=val_loss[i], text=f"{val_loss[i]:.4f}", showarrow=False,
                                    font=dict(size=10, color="red"), yshift=-10)

        fig_loss.update_layout(
            xaxis_title="Epoch",
            yaxis_title="Loss",
            xaxis=dict(tickmode='linear'),
            # yaxis=dict(range=[0.04, 0.08]),
            legend=dict(orientation="h", x=0.5, xanchor='center', y=-0.2),
            paper_bgcolor='rgba(0,0,0,0.5)',
            margin=dict(l=40, r=40, t=80, b=80)
        )

        st.plotly_chart(fig_loss, use_container_width=True)
```

Appendix G: Turnitin Similarity Report

TANG YU TONG-TP069018-APU3F2408CSDA_Final.docx

ORIGINALITY REPORT

15%	10%	11%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Asia Pacific University College of Technology and Innovation (UCTI) Student Paper	2%
2	eitca.org Internet Source	<1%
3	Submitted to Liverpool John Moores University Student Paper	<1%
4	arxiv.org Internet Source	<1%
5	ebin.pub Internet Source	<1%
6	Wu, Xian. "Deep Learning for Sensory and Behavioral Time Series Analysis", University of Notre Dame, 2022 Publication	<1%
7	Submitted to University of North Texas Student Paper	<1%
8	fastercapital.com Internet Source	

		<1%
9	Submitted to Gitam University Student Paper	<1%
10	link.springer.com Internet Source	<1%
11	dokumen.pub Internet Source	<1%
12	R. N. V. Jagan Mohan, B. H. V. S. Rama Krishnam Raju, V. Chandra Sekhar, T. V. K. P. Prasad. "Algorithms in Advanced Artificial Intelligence - Proceedings of International Conference on Algorithms in Advanced Artificial Intelligence (ICAAAI-2024)", CRC Press, 2025 Publication	<1%
13	www.coursehero.com Internet Source	<1%
14	"Social, Cultural, and Behavioral Modeling", Springer Science and Business Media LLC, 2023 Publication	<1%
15	Ahmad Nahid Ma'aly, Dita Pramesti, Ariadani Dwi Fathurahman, Hanif Fakhrurroja. "Exploring Sentiment Analysis for the Indonesian Presidential Election Through	<1%