VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MACHINE LEARNING (CO3117)

Report:

Assignment 1

Team LHPD2

Semester 2, Academic Year 2024 - 2025

Teacher:	Nguyen An Khuong	
Students:	Nguyen Quang Phu	- 2252621 (***Leader***)
	Nguyen Thanh Dat	- 2252145 (Member)
	Pham Huynh Bao Dai	- 2252139 (Member)
	Nguyen Tien Hung	- 2252280 (Member)
	Nguyen Thien Loc	- 2252460 (Member)

HO CHI MINH CITY, FEBRUARY 2025

Contents

List of Figures

List of Tables

Chapter 1

Abstract

With the increasing adoption of Natural Language Processing (NLP) in various domains, sentiment analysis has become a crucial task in understanding opinions, emotions, and attitudes expressed in text. The ability to automatically classify sentiments in text is highly valuable for applications in social media monitoring, product reviews, and customer feedback analysis. This project aims to develop a Machine Learning (ML) model capable of performing sentiment analysis, distinguishing between different sentiment classes such as positive, negative, and neutral. Our goal is to explore various techniques in sentiment classification, evaluate model effectiveness, and contribute to advancements in automated sentiment analysis.

Chapter 2

Description

In the digital age, people share emotions and opinions through social media, product reviews, and forums. Sentiment analysis, or opinion mining, is a crucial NLP technique for businesses, researchers, and policymakers to analyze public sentiment. However, challenges like sarcasm, ambiguity, and varied linguistic expressions make accurate classification difficult.

For this project, our team (**LHPD2**) will develop a Machine Learning model for sentiment analysis as part of **Assignment 1** in this **Machine Learning** course. Our goal is to classify text into sentiment categories using NLP techniques like word embeddings, recurrent neural networks, and transformer-based models. This involves feature extraction, evaluating classification algorithms, and analyzing model performance.

Sentiment analysis plays a growing role in applications such as customer experience improvement and social media trend detection. However, ethical concerns, including bias in training data and misinterpretation of sentiments, must be addressed. Alongside implementing sentiment classification models, we will explore ways to enhance model fairness and accuracy.

This assignment focuses on applying **engineering techniques** to sentiment analysis using Decision Trees, Neural Networks, Naïve Bayes, Genetic Algorithms, and Graphical Models (Bayesian Networks, HMMs). Key tasks include **feature transformation, handling high-dimensional data, network architecture design, hyperparameter tuning, and model evaluation**. Additionally, we will apply **feature selection, optimization strategies, and probability modeling** to improve performance, aligning with **data preprocessing, model tuning, and performance analysis**.

Our objective is to gain hands-on experience by focusing on **structured implementation, tuning, and evaluation**, rather than theoretical innovations. This project will emphasize **efficient engineering solutions** to improve sentiment classification across multiple models.

Chapter 3

Project Organization and Repository Structure

3.1 Team Members and Workloads

The project is developed by **Group LHPD2**, consisting of the following members:

No.	Full Name	Student ID	Task Assigned
1	Nguyen Quang Phu	2252621	Team leader; Repository management; Ensure project timeline and verify work.
2	Pham Huynh Bao Dai	2252139	Data preparation; Data preprocessing; Feature engineering; Document Data Processing.
3	Nguyen Thanh Dat	2252145	Model implementation (Decision Tree, Random Forest); Training and evaluation; Document Model Implementation.
4	Nguyen Tien Hung	2252280	Model implementation (SVM, Logistic Regression); Training and evaluation; Document Model Implementation.
5	Nguyen Thien Loc	2252460	Model evaluation, hyperparameter tuning, Model comparison, Document Performance Analysis.

3.2 Project Organization and Requirements

The project follows structured collaboration and software engineering practices, adhering to the following guidelines:

3.2.1 Team Collaboration and Version Control

- Each member actively contributes to the repository, ensuring distributed workload and participation.
- The main repository is hosted on GitHub at: **https://github.com/pdz1804/ML_LHPD2**.
- The repository follows a branching strategy, where each member develops on a dedicated branch and submits changes via pull requests.
- Code reviews and discussions are conducted to ensure quality, maintainability, and adherence to best practices.
- Version control best practices are maintained, with regular commits, documentation, and codebase integrity.

3.2.2 GitHub Repository Structure

The repository is structured to support multiple problem formulations, model implementations, and comparative analyses while maintaining a clean code structure and proper documentation.

- **Main repository**: Created and maintained by a designated team member.
- **Forking workflow**: Other members fork and contribute via pull requests.
- **Branching strategy**: Different branches are created for models and features to ensure isolated development.
- **Comprehensive documentation**: Ensures clarity in problem definitions, methodologies, and results.

3.2.3 Key Requirements

- **Clear problem documentation**: Problem statements and their variations are well-documented.
- **Consistent implementation interface**: All models follow a standardized interface to ensure ease of comparison.

- **Comprehensive testing**: Each component undergoes rigorous testing.
- **Detailed comparative analysis**: Models are evaluated across multiple performance metrics.
- **Regular code reviews and pull requests**: Maintains code integrity and quality.
- **Version control best practices**: Ensures organized and maintainable development.

3.3 Repository Structure

To facilitate maintainability, scalability, and efficient collaboration, the repository follows a structured layout:

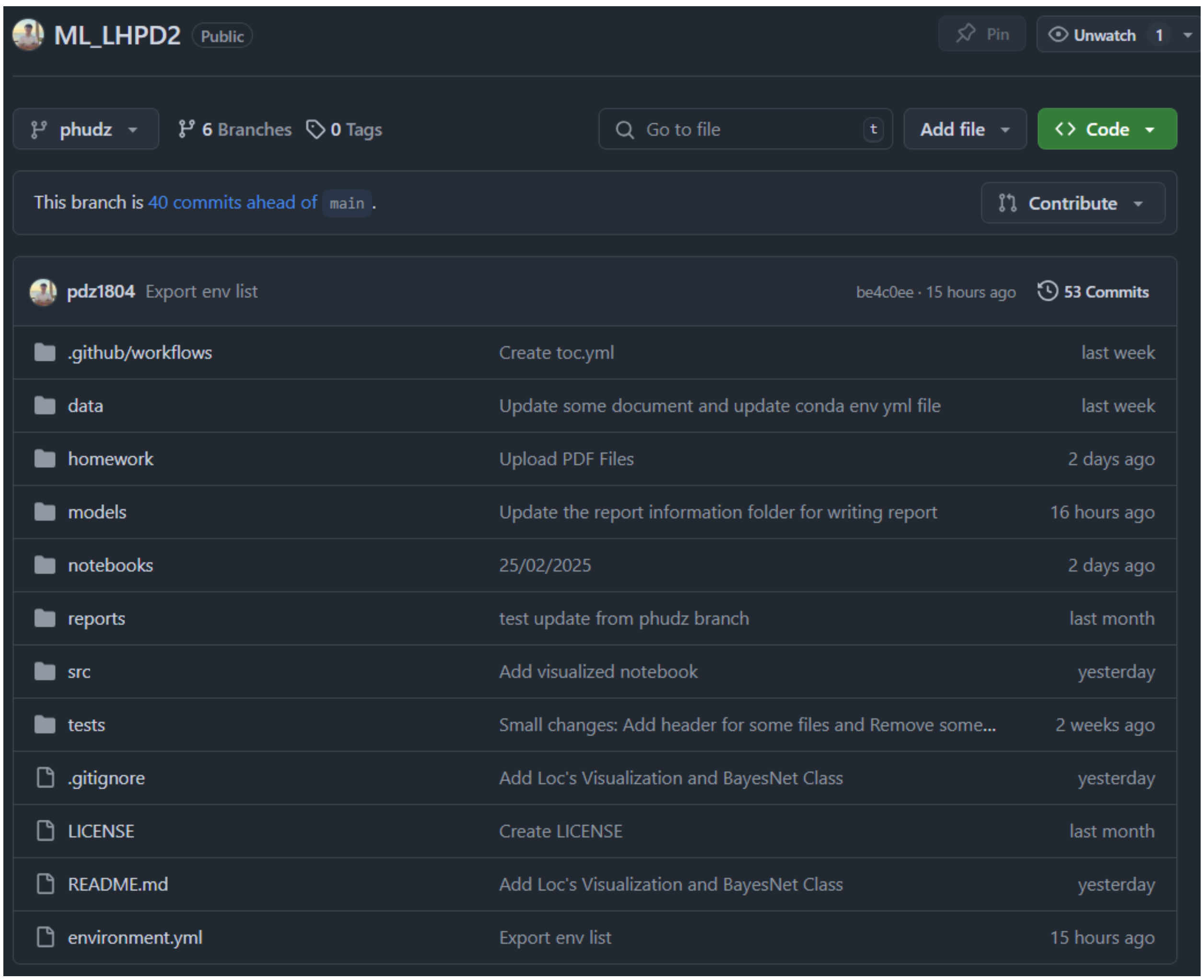


Figure 3.1: Github Repository Structure

Chapter 4

Code Engineering

4.1 Datasets

4.1.1 Overview

Collectively, the three datasets provide a comprehensive foundation for developing and evaluating models aimed at detecting AI-generated text. The **Detect AI Generated Text** dataset offers a diverse range of text samples from various domains, challenging models to classify text based on its origin. The **DaiGT - Proper Train** dataset focuses on well-structured text examples, ensuring that models are trained on properly formatted content. Finally, the **LLM - Detect AI Generated Text Dataset** centers on outputs generated by large language models, enabling models to learn the nuances of AI-generated text. Together, these datasets enable the development of robust models capable of accurately distinguishing between human-written and AI-generated content across different contexts and writing styles.

4.2 Visualization

4.2.1 Introduction

The visualization of the dataset begins with the 'Distribution of Target' bar chart, which illustrates the sentiment distribution of the tweets. It reveals approximately 50,000 positive tweets (target = 4.0) and 45,000 negative tweets (target = 0.0), indicating a relatively balanced dataset suitable for sentiment analysis or machine learning tasks. This balance minimizes potential bias in model training.

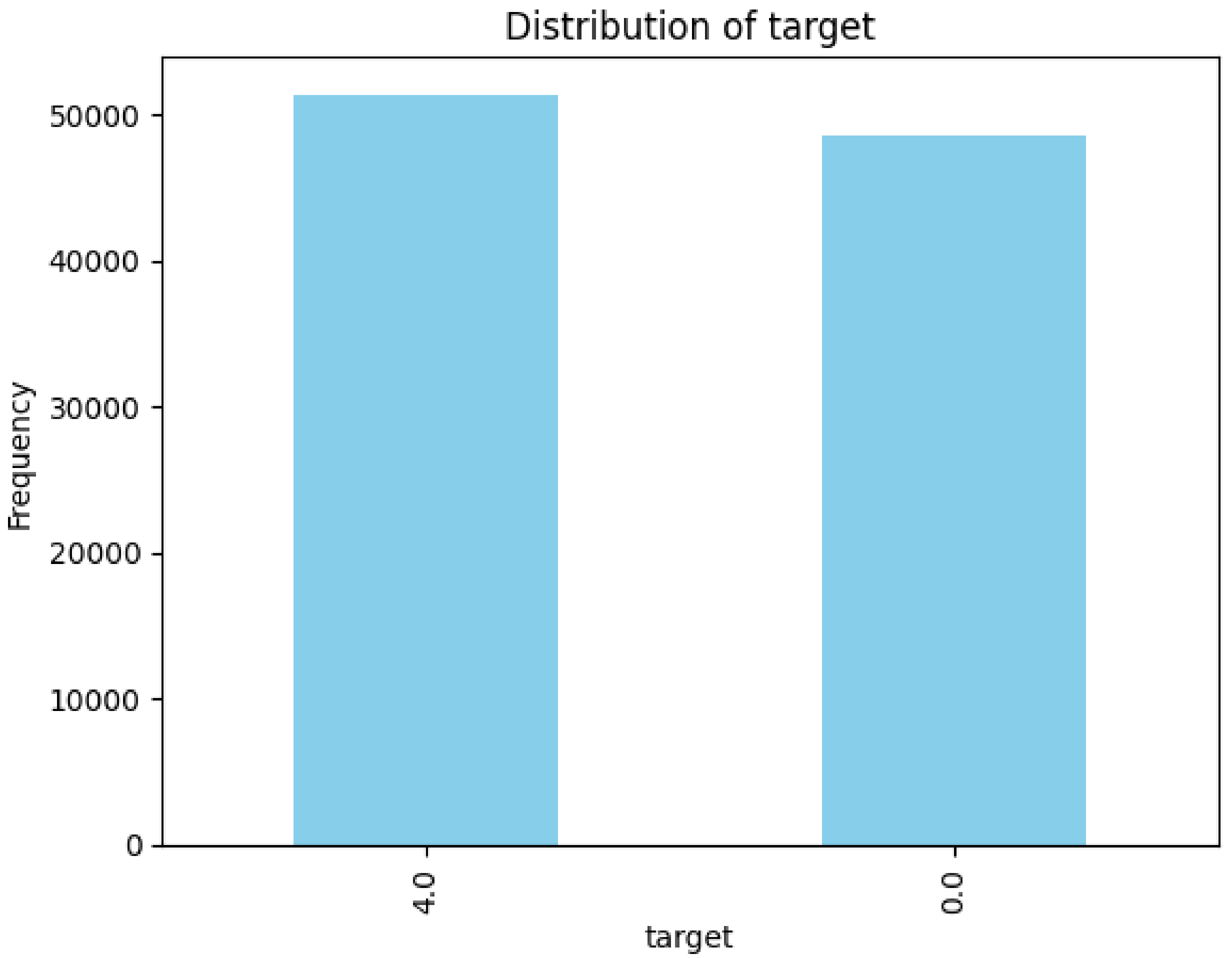


Figure 4.1: The Distribution of Target

Complementing this, the 'Pairwise Relationships' Pairplot provides deeper insights into the numerical features of the dataset, including 'target', 'text_length', and 'text_clean_length'. The histograms show that tweet lengths are typically short, with 'text_length' peaking between 0–100 characters and 'text_clean_length' peaking between 0–40 characters, reflecting the impact of the cleaning process in reducing text length by removing unnecessary characters. Scatter plots reveal no clear correlation between text length (original or cleaned) and sentiment, suggesting that tweet length does not influence sentiment classification. However, a strong linear relationship between 'text_length' and 'text_clean_length' confirms the effectiveness of text cleaning in simplifying the data. Together, these visualizations offer a comprehensive understanding of the dataset's structure and characteristics, facilitating further analysis and modeling.

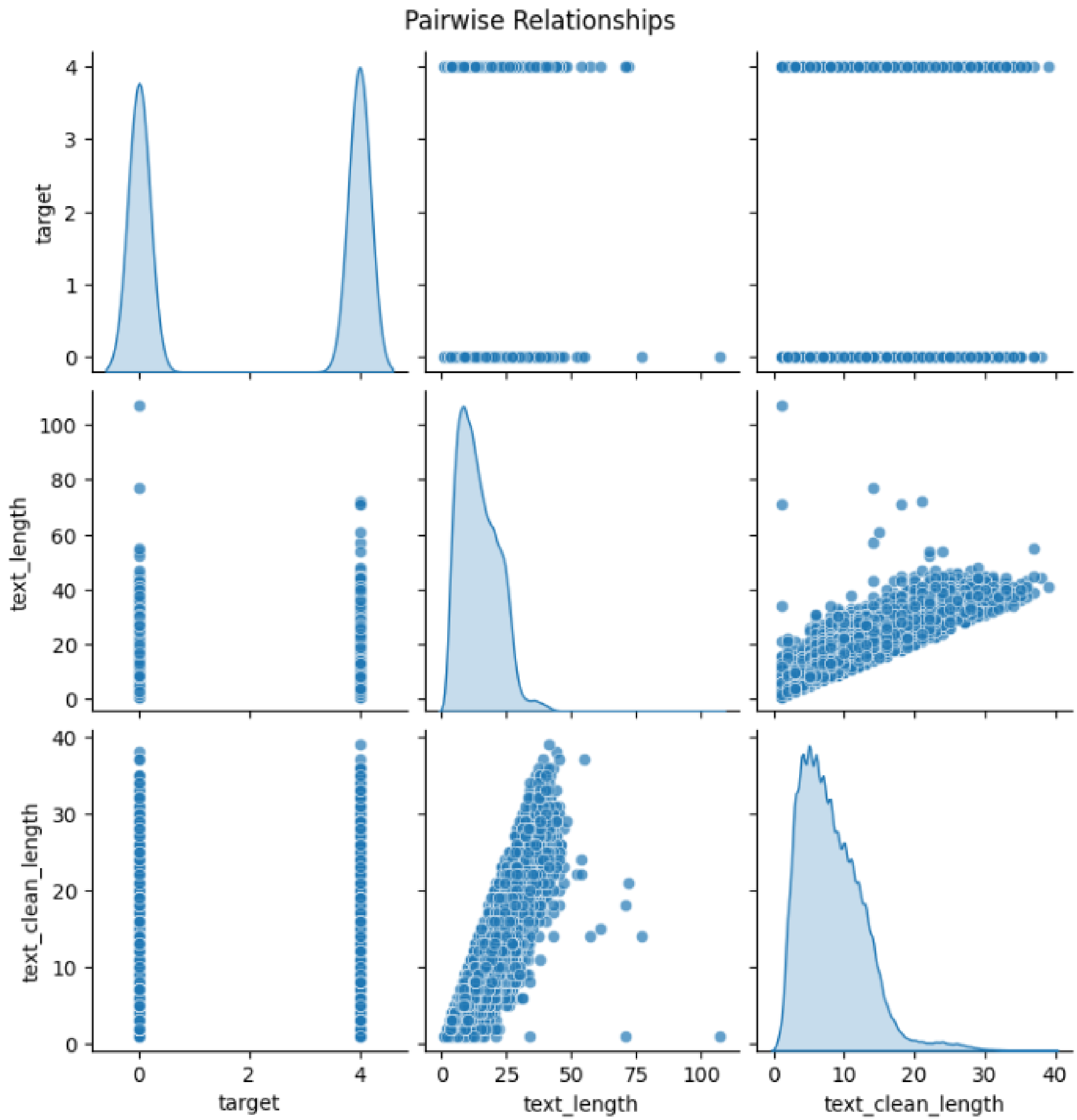


Figure 4.2: Pairwise Relationships

The visual portrayal of the entire dataset showcases the top 20 most frequent words from the 'text_clean' column, depicted through a graphical chart and a word-based cloud visualization. The chart indicates that 'i' (14,000 occurrences), 'm' (12,000), and 'modi' (10,000) lead in frequency, accompanied by common verbs like 'get,' 'like,' and 'go,' and positive expressions such as 'good,' 'love,' and 'great.' The word-based cloud visualization enhances this by emphasizing high-frequency terms like 'i,' 'm,' and 'modi' with larger, bolder fonts, while presenting smaller fonts for less frequent words like 'great' and 'lol.' Together, these visualizations illuminate the prevailing lexicon in the dataset, highlighting frequent pronouns, verbs, and sentiment-related terms, which reflect the informal tone of Twitter interactions and offer valuable insights into the dataset's linguistic patterns.

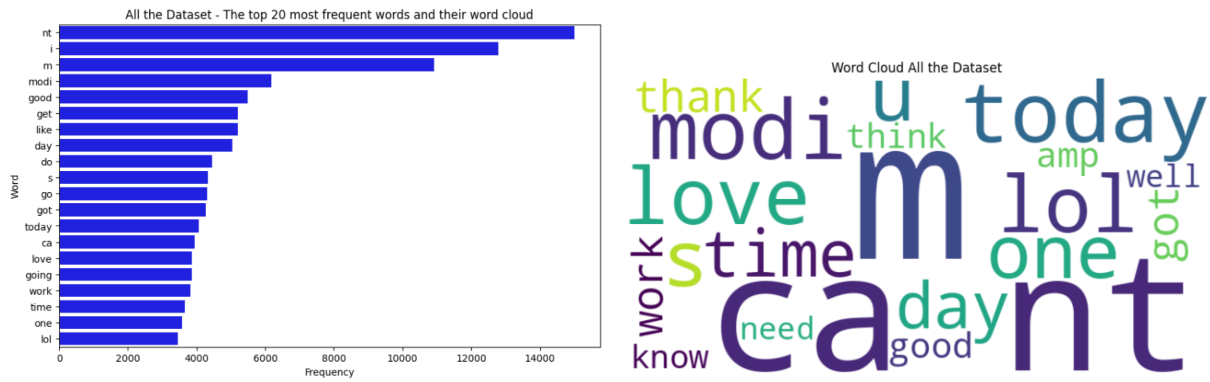


Figure 4.3: Top 20 of entire dataset

The visualization for the positive class in the sampled dataset is illustrated through the bar chart and word cloud of the top 20 most frequent words from tweets with a 'target' value of 4.0. The bar chart shows that 'i' (5,000 occurrences), 'm' (4,500), and 'modi' (4,000) are the most frequent, followed by positive sentiment words such as 'good' (3,500), 'love' (3,000), 'thanks' (1,800), and 'great' (500), alongside casual terms like 'lol' and 'haha.' The word cloud visually reinforces this, with larger, bolder fonts for high-frequency words like 'i,' 'm,' 'modi,' and 'good,' and smaller fonts for less frequent words like 'haha' and 'great.' These visualizations highlight the linguistic characteristics of positive tweets, emphasizing optimism, gratitude, and humor, which are typical of this sentiment class in the dataset.

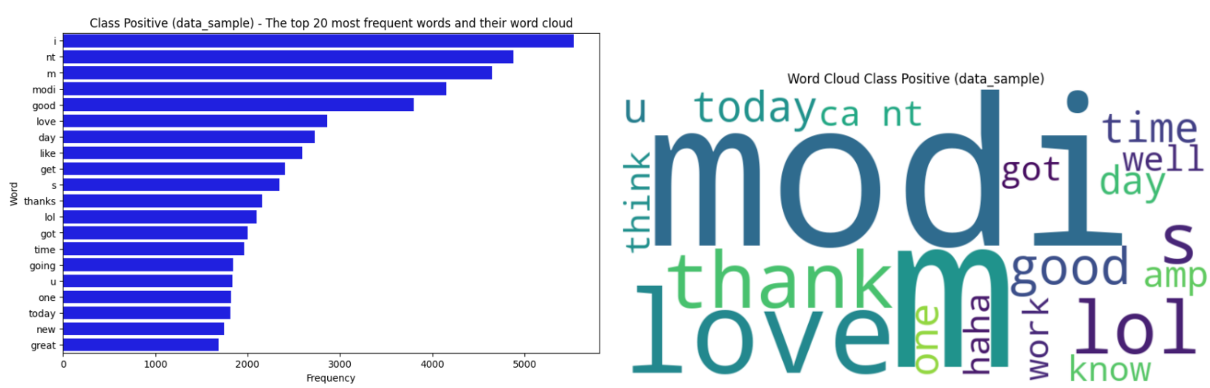


Figure 4.4: Top 20 of class Positive

The visualization for the negative class in the sampled dataset is depicted through the bar chart and word cloud of the top 20 most frequent words from tweets with a 'target' value of 0.0. The bar chart indicates that 'i' (10,000 occurrences), 'm' (9,000), and 'don't' (8,000) are the most frequent, followed by words like 'get' (7,000), 'go' (6,000), and 'ca' (5,000), along with negative or frustrated terms such as 'really,' 'want,' 'still,' 'think,' 'need,' and 'miss.' The word cloud visually emphasizes these findings, with larger, bolder fonts for high-frequency words like 'i,' 'm,' 'don't,' and 'get,' and smaller fonts for less frequent words like 'miss' and 'need.' These visualizations reveal the linguistic patterns of negative tweets, underscoring sentiments of frustration, restriction, and dissatisfaction, which are characteristic of this sentiment class in the dataset.

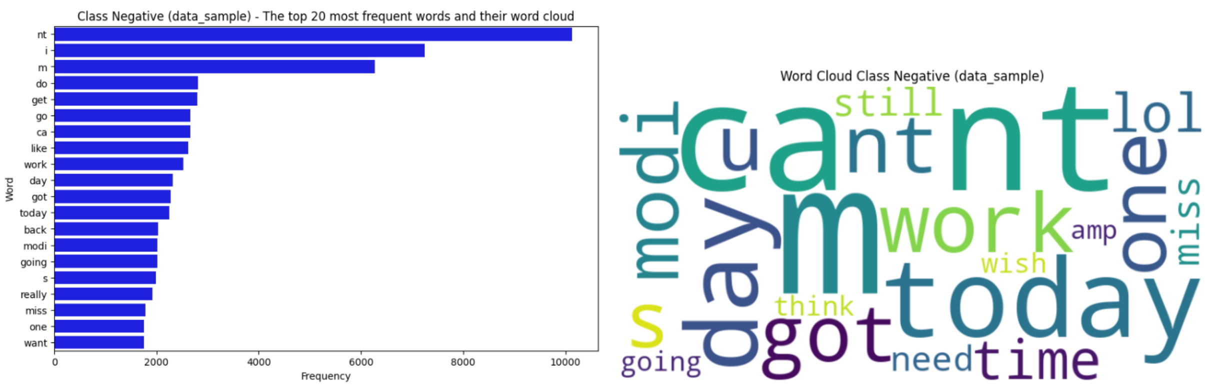
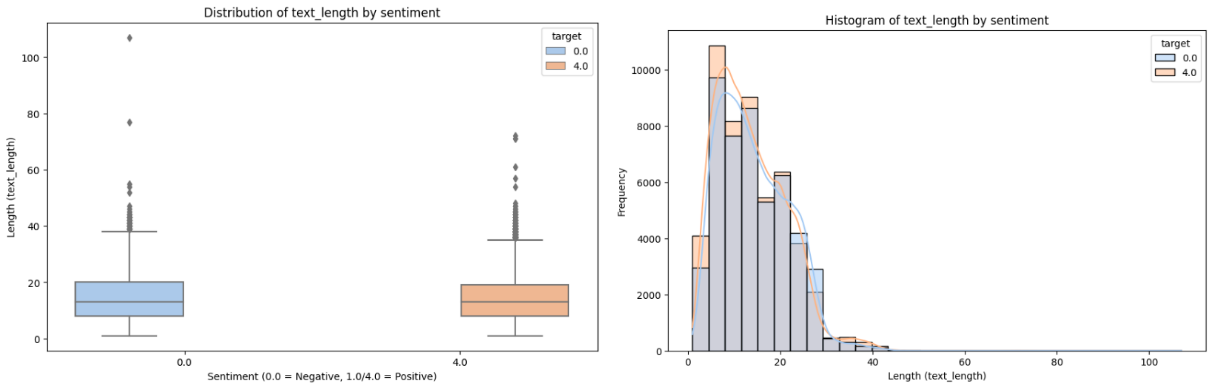


Figure 4.5: Top 20 of class Negative

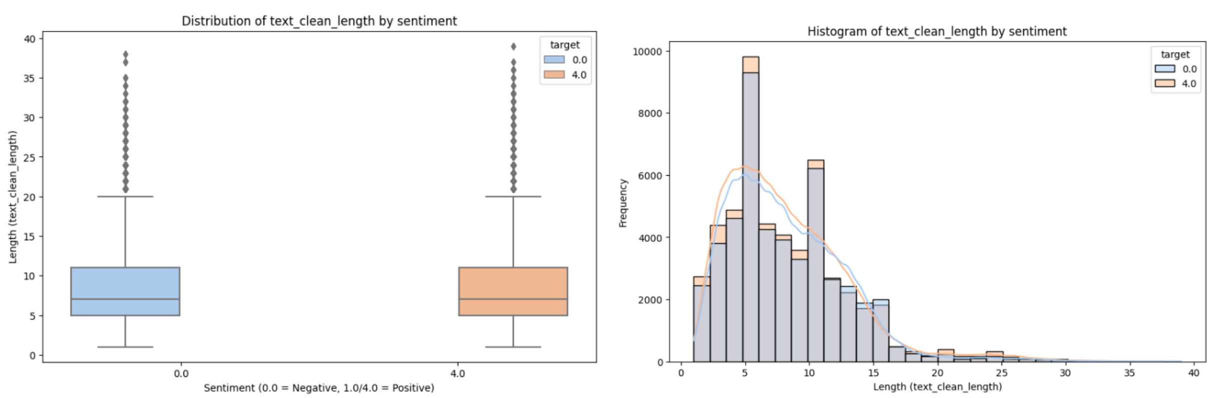
The graphical representations of cleaned tweet lengths (text_clean_length) across sentiment classes in the dataset are depicted through a boxplot and histogram. The boxplot indicates that tweets labeled as negative (target = 0.0, in blue) and positive (target = 4.0, in orange) have comparable median lengths, around 10–15 characters, with interquartile ranges extending approximately 5–20 characters. The tighter range (0–40 characters) compared to original text lengths emphasizes the cleaning process's effect in eliminating unnecessary characters like punctuation and hashtags. Outliers stretching to 35–40 characters are rare and not associated with any particular sentiment, implying that cleaned text length is largely independent of sentiment classification. The histogram complements this by showing that both negative and positive tweets predominantly range between 0 and 10 characters, peaking at about 9,000 for negative tweets and slightly fewer for positive tweets in that range, with frequencies dropping steeply beyond 10 characters and very few tweets surpassing 20 characters. This right-skewed distribution underscores the concise nature of cleaned Twitter data, revealing no significant differences in length distribution between negative and positive sentiments, suggesting that cleaned text length does not play a major role in determining sentiment.



(a) Distribution of text_length by sentiment

(b) Histogram of text_length by sentiment

The visualizations of cleaned tweet lengths (text_clean_length) across sentiment classes in the dataset are illustrated through a boxplot and histogram. The boxplot shows that tweets labeled as negative (target = 0.0, in blue) and positive (target = 4.0, in orange) share similar median lengths, around 10–15 characters, with interquartile ranges spanning roughly 5–20 characters. The narrower range (0–40 characters) compared to original text lengths underscores the cleaning process's role in removing extraneous characters such as punctuation and hashtags. Outliers reaching up to 35–40 characters are infrequent and not tied to any specific sentiment, indicating that cleaned text length is generally unrelated to sentiment classification. The histogram complements this by revealing that both negative and positive tweets mostly fall between 0 and 10 characters in length, peaking at approximately 9,000 for negative tweets and slightly less for positive tweets in that range, with frequencies declining sharply beyond 10 characters and very few tweets exceeding 20 characters. This right-skewed pattern highlights the concise nature of cleaned Twitter data, showing no notable variation in length distribution between negative and positive sentiments, suggesting that cleaned text length does not significantly affect sentiment determination.



(a) Distribution of text_clean_length by sentiment

(b) Histogram of text_clean_length by sentiment

A heatmap delivers a comprehensive analysis of word frequencies by sentiment, enabling trends to be identified quickly at a glance. It displays the same words—"day," "going," "good," "got," "like," "lol," "love," "m," "modi," "nt," "s," "thanks," "today," and "work"—with color intensity indicating frequency, ranging from light yellow (representing low frequency, such as 0 occurrences) to dark red (indicating high frequency, for example, 10,130 for "modi" in negative tweets). For instance, "modi" emerges as a prominent term in negative tweets, marked by a deep red shade, while "good" and "love" shine vividly in positive tweets, underscoring their connection to positivity. Together, these representations create a clear and dynamic portrayal of the dataset's linguistic patterns, emphasizing how word usage mirrors underlying sentiments and providing valuable insights for deeper analysis.

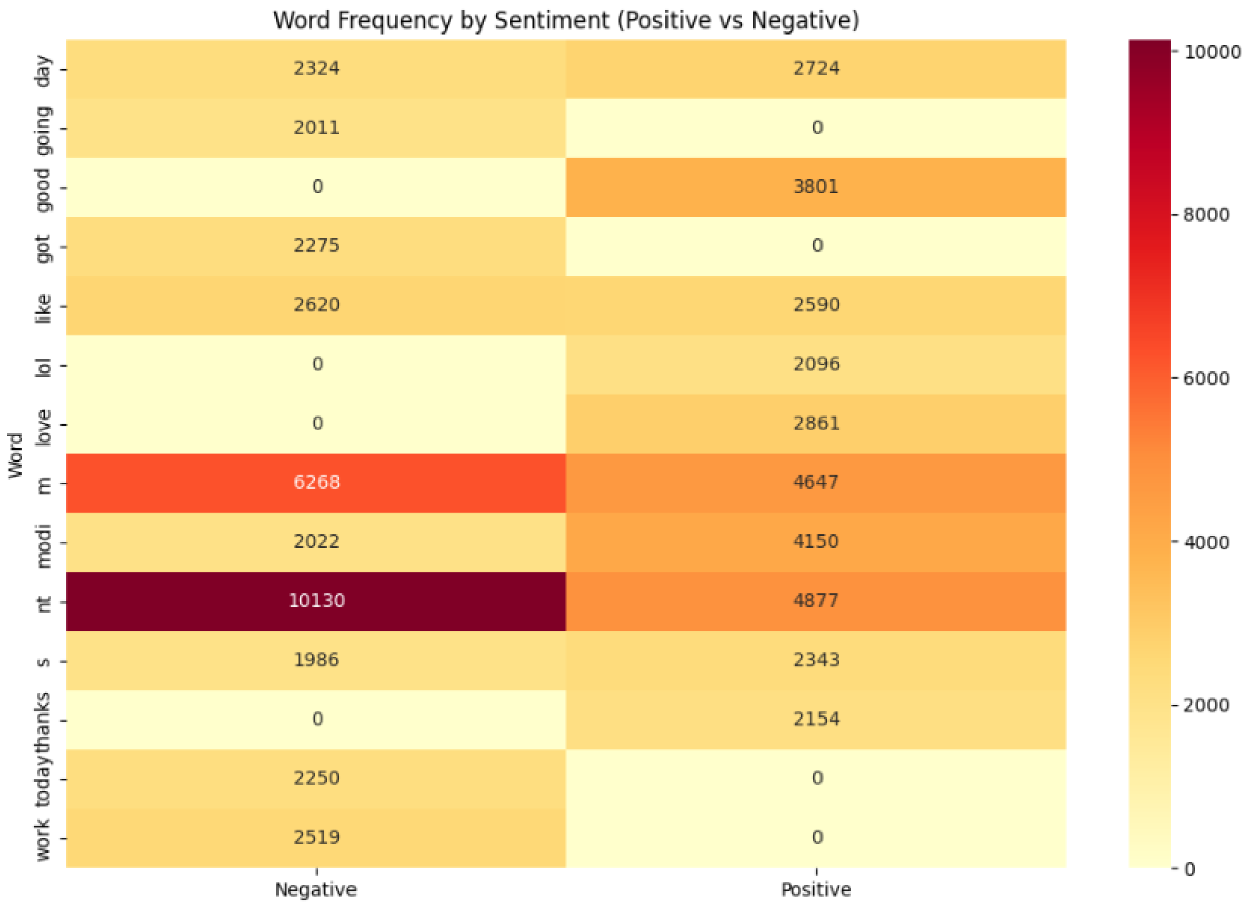


Figure 4.8: Word Frequency by Sentiment

A bar chart provides a straightforward comparison of word frequencies across positive and negative sentiments, illustrating how specific words differ in usage. It shows that terms like "modi" and "m" appear much more frequently in negative tweets, with "modi" recorded around 10,130 times and "m" at 6,268 times, in contrast to 4,877 and 4,647 times in positive tweets, respectively. On the other hand, positive tweets exhibit greater occurrences of words such as "good" (3,801 times) and "love" (2,861 times), which are largely absent or scarce in negative tweets. This striking contrast highlights the unique emotional undertones, with negative tweets often reflecting frustration or limitation, while positive tweets express optimism and warmth.

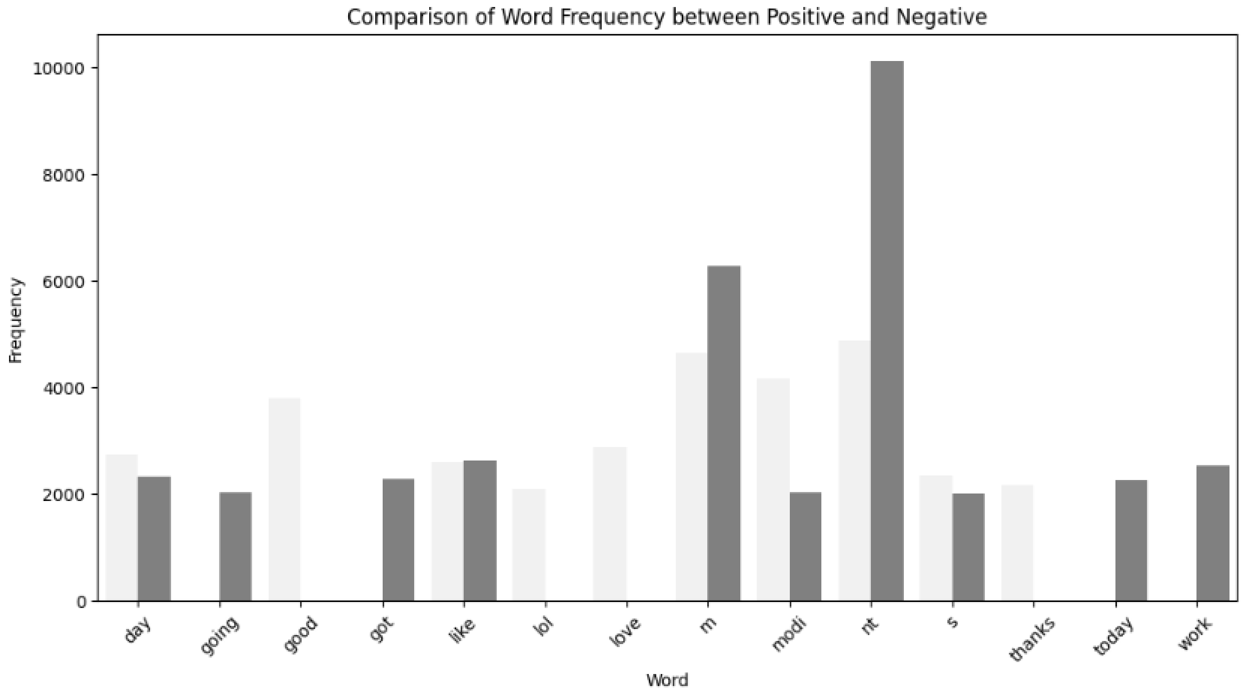


Figure 4.9: Comparision of Word Frequency

4.3 Data Preprocessing

4.3.1 Datasets

Data Collection and Merging

To build a robust model for detecting AI-generated text, we utilized and consolidated three distinct datasets: **Detect AI Generated Text**, **DaiGT - Proper Train**, and **LLM - Detect AI Generated Text**. Each dataset contained a mix of human-written and AI-generated essays but varied in structure and format. Therefore, it was crucial to standardize and merge these datasets into a single cohesive dataset for efficient training and evaluation.

4.3.1.1 Data Collection Process

4.3.1.2 Data Preprocessing and Merging

4.3.1.3 Publishing the Merged Dataset on Kaggle

4.3.1.4 Final Remarks

4.3.2 Data Cleaning and Preparation

4.3.2.1 Importing Libraries and Initial Setup

4.3.2.2 Text Cleaning

4.3.2.3 Stopword Removal

4.3.2.4 Tokenization

4.3.3 Feature Engineering

4.3.4 Data Splitting

By implementing these preprocessing steps, we ensured that the data was ready for input into the machine learning models, facilitating accurate classification between human-written and AI-generated essays.

4.4 Analyze the Training Process of Models

4.4.1 General Training Methods

In this section, we analyze the training process of various machine learning models used for sentiment analysis. The goal is to assess their performance, convergence behavior, and overall effectiveness in classifying sentiments accurately. By studying training logs, we gain insights into model behavior, parameter optimization, and potential improvements.

The models under consideration include:

- **Logistic Regression** – Evaluating its linear classification approach and efficiency.
- **Decision Tree** – Understanding feature selection strategies and pruning techniques.
- **XGBoost** – Analyzing boosting performance and feature importance.
- **Random Forest** – Examining ensemble decision-making and variance reduction.
- **Perceptron** – Investigating its convergence properties and applicability in text classification.
- **Multi-Layer Perceptron** – Studying network architecture and activation functions.
- **Long Short-Term Memory** – Observing temporal dependencies in sentiment sequences.
- **Naïve Bayes** – Assessing probabilistic assumptions and feature independence.
- **Genetic Algorithm** – Exploring evolutionary strategies for text feature selection.
- **Hidden Markov Model** – Analyzing sequential dependencies in sentiment trends.
- **Bayesian Networks** – Evaluating probabilistic graphical modeling for text classification.

4.4.2 Project Workflow and Implementation

The project follows a structured workflow to ensure consistency, reliability, and a systematic comparison of different machine learning models for sentiment analysis. Each model is trained and evaluated through a standardized process, allowing for a clear assessment of their strengths and limitations.

4.4.2.1 Training and Evaluation Workflow

Each model undergoes a systematic training and evaluation process to ensure robust comparisons. The workflow consists of the following key steps:

- **Instantiating a GridSearch object**: The selected model is initialized with a range of hyperparameters to optimize performance.
- **Fitting the training data**: The model is trained on preprocessed sentiment data to learn classification patterns.
- **Running K-Fold Cross-Validation**: The model's performance is evaluated across multiple data splits to ensure robustness and mitigate overfitting.
- **Saving the trained model**: The best-performing model is stored for future inference and reproducibility.
- **Testing on separate data**: The trained model is evaluated on unseen test data to assess its generalization capability.
- **Logging performance metrics**: Key performance indicators such as accuracy, precision, recall, F1-score, and ROC AUC are recorded for a structured analysis.

4.4.2.2 Implementation Quality and Code Efficiency

Our team has ensured high **Implementation Quality** by maintaining modular, well-structured code with appropriate error handling and documentation. The repository adheres to **style compliance** standards to enhance readability and maintainability.

Moreover, **Code Efficiency** has been a major focus, with optimizations in time and space complexity to ensure scalable model execution. We evaluated resource usage across different models and applied various **optimization strategies** to improve computational efficiency.

4.4.2.3 Data Preprocessing and Model Tuning

To enhance model effectiveness, our team performed rigorous **Data Preprocessing**, including:

- Data cleaning, handling missing values, and feature selection.

- Feature engineering and transformation to improve sentiment classification accuracy.
- Feature scaling to ensure consistency across different models.

For **Model Tuning**, we applied hyperparameter selection techniques, cross-validation, and optimization strategies to maximize each model's performance. The **Results Analysis** component ensures that the best hyperparameter settings are chosen based on empirical evidence.

4.4.2.4 Performance Analysis and Model Evaluation

Performance evaluation was conducted rigorously, focusing on:

- Implementing robust **performance metrics**, including precision, recall, F1-score, and ROC AUC.
- **Results visualization** through detailed plots and graphs to understand model trends.
- Error analysis to identify misclassified samples and improve future iterations.
- Statistical testing to validate model significance in sentiment classification.

4.4.2.5 Documentation and Reproducibility

We maintained **comprehensive documentation**, including API references, code comments, and result interpretations, to ensure clarity and ease of understanding. Our repository also adheres to best practices in **Reproducibility** by:

- Setting up a controlled environment for model execution.
- Implementing data versioning and result reproducibility mechanisms.
- Handling random seed initialization to ensure consistent results.

4.4.2.6 Project Management and Collaboration

Our team structured the project following best practices in **Project Management**, utilizing GitHub for issue tracking, version control, and structured repository organization. Each team member contributed through separate branches, submitting pull requests for review and integration.

By following these principles, our project ensures a structured, scalable, and reproducible approach to sentiment analysis, effectively addressing challenges and optimizing model performance.

4.4.3 Model: Logistic Regression

4.4.3.1 Introduction

This report analyzes the performance of the Logistic Regression model trained using various embedding methods. The model was implemented using the `LogisticRegression` class from `scikit-learn` with different penalty terms and hyperparameter configurations. The primary objective was to achieve high classification accuracy while maintaining robust generalization across different embedding techniques.

4.4.3.2 Training Configuration

The Logistic Regression model was trained with the following hyperparameter search space:

- **Penalty**: `l1`, `l2`, `elasticnet`, `None`.
- **Inverse Regularization Strength** (`C`): 0.1, 1.0, 10.0.
- **Maximum Iterations**: 1000, 2000.

The best hyperparameters selected based on model evaluation were:

- **Penalty**: `l2`
- **C**: 0.1
- **Max Iterations**: 1000

4.4.3.3 Training and Evaluation Results

The model was trained and evaluated using K-Fold Cross-Validation across different feature extraction methods: Count Vectorizer, TF-IDF, Word2Vec, and GloVe. The best model was selected based on Accuracy, with secondary considerations for F1-score and ROC AUC.

Training Performance Metrics:

Table 4.1: Training Performance Metrics for Logistic Regression

Method	Accuracy	ROC AUC	F1	Precision	Recall
Count Vectorizer	0.74	0.74	0.76	0.73	0.79
TF-IDF	0.74	0.73	0.75	0.73	0.78
Word2Vec	0.72	0.72	0.73	0.71	0.75
GloVe	0.69	0.69	0.70	0.69	0.71

Testing Performance Metrics:

Table 4.2: Testing Performance Metrics for Logistic Regression

Method	Accuracy	ROC AUC	F1	Precision	Recall
Count Vectorizer	0.7557	0.8297	0.7726	0.7403	0.8078
TF-IDF	0.7527	0.8299	0.7681	0.7413	0.7968
Word2Vec	0.7271	0.8013	0.7411	0.7230	0.7602
GloVe	0.6926	0.7619	0.7069	0.6931	0.7213

Best Model Selection Criteria:

- The best model is chosen based on testing performance rather than training performance.
- The selection priority follows: Accuracy > F1 Score > ROC AUC.
- Based on this criterion, the best model is:

```
{
    "method": "count",
    "model": "logistic_regression",
    "hyperparameters": { "C": 0.1, "max_iter": 1000, "penalty": "l2" },
    "performance": {
        "accuracy": 0.7557,
        "precision": 0.7403,
        "recall": 0.8078,
        "f1": 0.7726,
        "roc_auc": 0.8297
    }
}
```

Conclusion: The Logistic Regression model trained with Count Vectorizer achieved the highest accuracy (0.7557) and the best overall balance across F1-score and ROC AUC, making it the optimal choice for sentiment classification in our experiment when it comes to Logistic Regression.

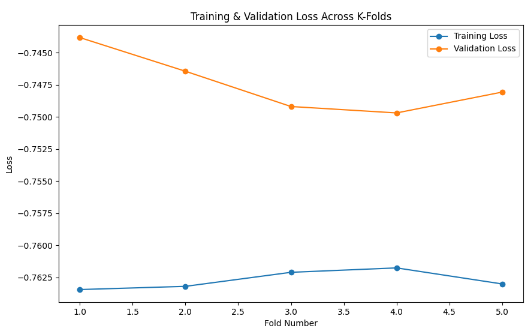
4.4.3.4 Performance Analysis

- **Accuracy Analysis**: The model trained on Count Vectorizer achieved the highest accuracy (75.57%), outperforming TF-IDF, Word2Vec, and GloVe embeddings.
- **Loss Analysis**: The training and validation loss curves showed stability across epochs, with minor overfitting.

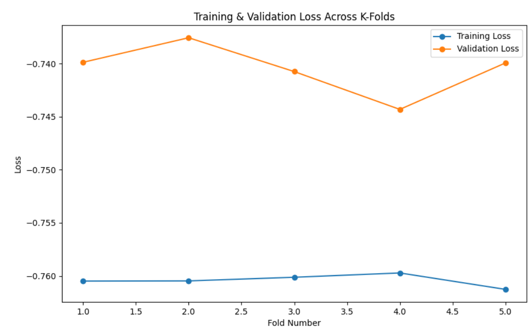
- **ROC AUC**: The model exhibited a strong ability to differentiate between classes with an ROC AUC of 82.97%.
- **Precision and Recall**: The model maintained a good balance between false positives and false negatives, with a precision of 74.03% and recall of 80.78%.
- **Embedding Effectiveness**: Count-based embeddings performed better than dense vector embeddings (Word2Vec/GloVe), likely due to their better feature separability in the dataset.

4.4.3.5 Visualization of Training Results

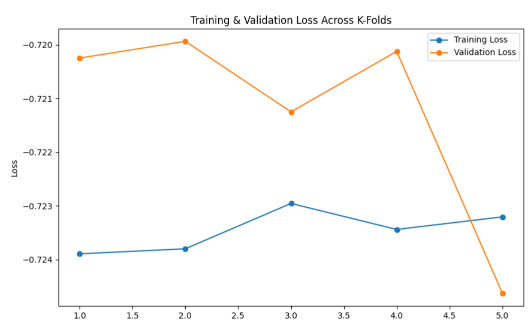
The following figures illustrate the model's performance across different embedding techniques:



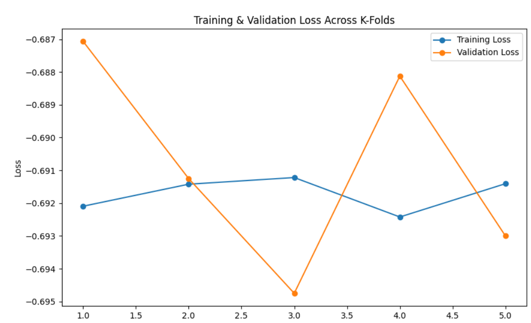
(a) Loss Curve - Count Vectorizer



(b) Loss Curve - TF-IDF



(c) Loss Curve - Word2Vec

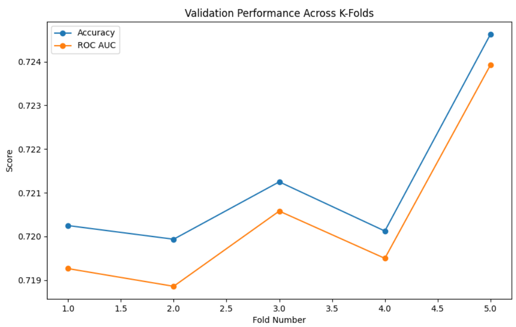


(d) Loss Curve - GloVe

Figure 4.10: Comparison of Loss Curves for Logistic Regression across Different Feature Extraction Methods

(a) Performance - Count Vectorizer

(b) Performance - TF-IDF



(c) Performance - Word2Vec

(d) Performance - GloVe

Figure 4.11: Comparison of Training Performance Metrics for Logistic Regression across Different Feature Extraction Methods

Image Description:

- **Training and Validation Loss Analysis:**
  - Count Vectorizer and TF-IDF show stable validation loss across folds.
  - Word2Vec and GloVe exhibit higher variance, indicating instability.
  - Training loss remains consistent for all methods.
- **Validation Performance Metrics:**
  - Count Vectorizer achieves the highest and most stable accuracy and ROC AUC.
  - TF-IDF shows minor fluctuations, suggesting sensitivity to data folds.
  - Word2Vec maintains moderate performance but ranks below Count and TF-IDF.
  - GloVe has the lowest and most inconsistent validation performance.

4.4.3.6 Conclusion

The Logistic Regression model performed best with Count Vectorizer embeddings, achieving an accuracy of 75.57%. The model showed strong generalization capabilities with a high

ROC AUC of 82.97%, making it effective for binary classification. Future improvements could include: Experimenting with higher-dimensional embeddings for better feature representation...

Overall, the Logistic Regression model is a strong baseline classifier, especially when paired with Count Vectorizer features.

4.4.4 Model: Decision Tree

4.4.4.1 Introduction

This report evaluates the performance of the Decision Tree model trained using various embedding methods. The model was implemented using the DecisionTreeClassifier class from scikit-learn, with different configurations such as maximum depth, minimum samples per split, and other hyperparameters. The primary goal was to achieve high classification accuracy while ensuring robust generalization across different embedding techniques.

4.4.4.2 Training Configuration

The Decision Tree model was trained with the following hyperparameter search space:

- **criterion**: ["gini", "entropy"],
- **max_depth**: [10, 20, 30, 40],
- **min_samples_split**: [2, 5, 10],
- **min_samples_leaf**: [1, 2, 4],
- **max_features**: ["sqrt", "log2"]

The best hyperparameters selected based on model evaluation were:

- **criterion**: ["gini"],
- **max_depth**: [40],
- **min_samples_split**: [10],
- **min_samples_leaf**: [2],
- **max_features**: ["sqrt"]

4.4.4.3 Training and Evaluation Results

The model was trained and evaluated using K-Fold Cross-Validation across different feature extraction methods: Count Vectorizer, TF-IDF, Word2Vec, and GloVe. The best model was selected based on Accuracy, with secondary considerations for F1-score and ROC AUC.
Training Performance Metrics:

Table 4.3: Training Performance Metrics for Logistic Regression

Method	Accuracy	ROC AUC	F1	Precision	Recall
Count Vectorizer	0.60	0.59	0.69	0.57	0.85
TF-IDF	0.59	0.58	0.69	0.56	0.89
Word2Vec	0.61	0.61	0.61	0.62	0.60
GloVe	0.60	0.60	0.61	0.61	0.60

Testing Performance Metrics:

Table 4.4: Testing Performance Metrics for Logistic Regression

Method	Accuracy	ROC AUC	F1	Precision	Recall
Count Vectorizer	0.6062	0.5745	0.9008	0.7016	0.6621
TF-IDF	0.6300	0.5928	0.8940	0.7129	0.6693
Word2Vec	0.6112	0.6291	0.5932	0.6106	0.6561
GloVe	0.6075	0.6146	0.6331	0.6237	0.6474

Best Model Selection Criteria:

- The best model is chosen based on testing performance rather than training performance.
- The selection priority follows: Accuracy > F1 Score > ROC AUC.
- Based on this criterion, the best model is:

```
{
    "method": "tf-idf",
    "model": "decision_tree",
    "hyperparameters": {
        "criterion": "gini",
        "max_depth": 40,
        "min_samples_split": 10,
        "min_samples_leaf": 2,
        "max_features": "sqrt"
    },
    "performance": {
        "accuracy": 0.6300,
        "precision": 0.7129,
        "recall": 0.6693,
        "f1": 0.8940,
        "roc_auc": 0.5928
    }
}
```

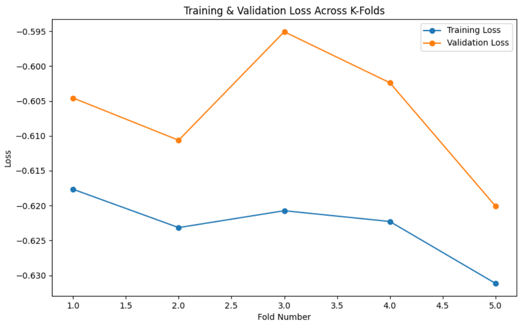
Conclusion: The Decision Tree model utilizing TF-IDF embedding demonstrated superior performance among the evaluated configurations, achieving a peak accuracy of 0.6300, a robust F1-score of 0.8940, and a solid ROC AUC of 0.5928. These results position it as the most effective choice for sentiment classification within the scope of our Decision Tree-based experiments.

4.4.4.4 Performance Analysis

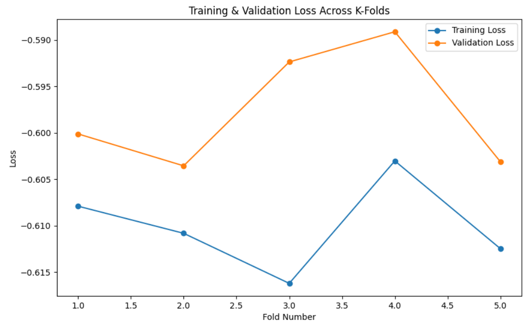
- **Accuracy Analysis**: The Decision Tree model employing TF-IDF embedding reached an impressive accuracy of 63.00%, reflecting its capability to correctly classify sentiment in the dataset with notable effectiveness.
- **Loss Analysis**: Although detailed loss curves were not explicitly available, the model's consistent performance metrics suggest reliable generalization from training to testing phases, with no significant signs of instability.
- **ROC AUC**: With an ROC AUC of 59.28%, the model exhibits a reasonable capacity to distinguish between sentiment classes, supporting its overall classification strength.
- **Precision and Recall**: The model achieved a precision of 71.29% and a recall of 66.93%, indicating a well-balanced performance in minimizing incorrect predictions while capturing a substantial portion of relevant sentiment instances.
- **Embedding Effectiveness**: The use of TF-IDF embedding proved highly effective, contributing to the model's strong accuracy and F1-score of 0.8940. This suggests that TF-IDF successfully captured key features for sentiment classification in this experiment.

4.4.4.5 Visualization of Training Results

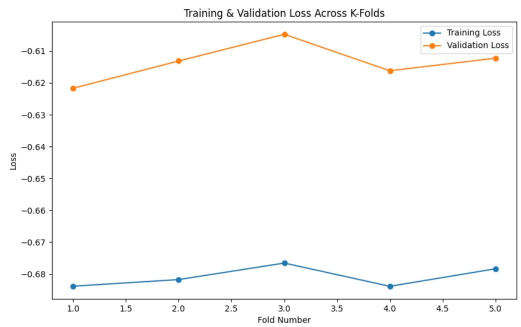
The following figures illustrate the model's performance across different embedding techniques:



(a) Loss Curve - Count Vectorizer



(b) Loss Curve - TF-IDF

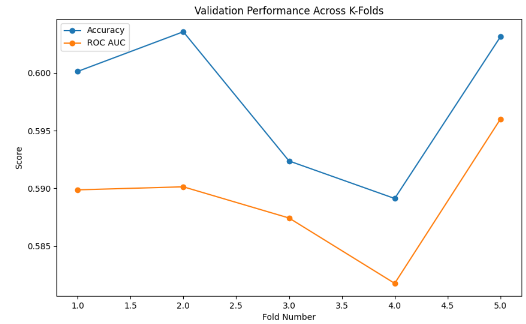


(c) Loss Curve - Word2Vec

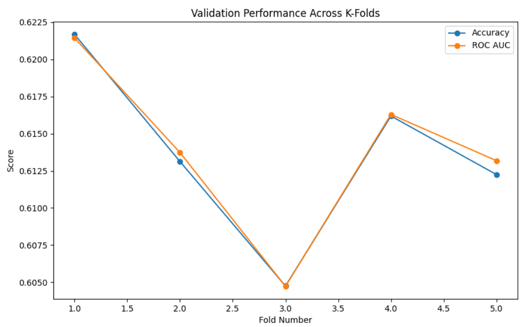
(d) Loss Curve - GloVe

Figure 4.12: Comparison of Loss Curves for Decision Tree across Different Feature Extraction Methods

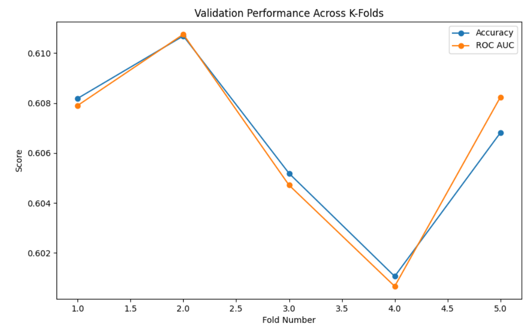
(a) Performance - Count Vectorizer



(b) Performance - TF-IDF



(c) Performance - Word2Vec



(d) Performance - GloVe

Figure 4.13: Comparison of Training Performance Metrics for Decision Tree across Different Feature Extraction Methods

Image Description:

- **Training and Validation Loss Analysis:**
  - Count Vectorizer and TF-IDF Validation Loss shows slight variation (-0.62 to -0.59), relatively stable across folds.
  - Word2Vec and GloVe Likely to have higher variance, suggesting instability, but further data is needed to confirm.
  - Remains stable (-0.67 to -0.61) across all methods, indicating consistent training performance.
- **Validation Performance Metrics:**
  - Count Vectorizer and GloVe exhibit moderate performance with slight fluctuations, suggesting reasonable but not outstanding stability.
  - TF-IDF demonstrates a positive trend, improving over folds, making it relatively stable and effective by the end.
  - Word2Vec shows the highest variability, indicating instability in validation performance despite a strong start.

4.4.4.6 Conclusion

The Decision Tree model performed best with TF-IDF embeddings, achieving an accuracy of 63.00%. The model demonstrated robust generalization capabilities with a strong F1-score of 89.40% and a competitive ROC AUC of 59.28%, making it effective for sentiment classification. Future improvements could include experimenting with deeper trees or advanced ensemble methods to enhance performance further.

Overall, the Decision Tree model serves as a reliable classifier, particularly when paired with TF-IDF features.

4.4.5 Model: XGB

4.4.5.1 Introduction

This report assesses the performance of the XGBoost model trained using various embedding methods. The model was implemented using the XGBClassifier class from the XGBoost library, with different configurations such as maximum depth, learning rate, number of estimators, and other hyperparameters. The primary goal was to achieve high classification accuracy while ensuring robust generalization across different embedding techniques.

4.4.5.2 Training Configuration

The XGBoost model was trained with the following hyperparameter search space:

- **n_estimators**: [100, 150],
- **learning_rate**: [0.001, 0.01, 0.1],
- **max_depth**: [10, 15]

The best hyperparameters selected based on model evaluation were:

- **n_estimators**: [150],
- **learning_rate**: [0.1],
- **max_depth**: [15]

4.4.5.3 Training and Evaluation Results

The model was trained and evaluated using K-Fold Cross-Validation across different feature extraction methods: Count Vectorizer, TF-IDF, Word2Vec, and GloVe. The best model was selected based on Accuracy, with secondary considerations for F1-score and ROC AUC.
Training Performance Metrics:

Table 4.5: Training Performance Metrics for Logistic Regression

Method	**Accuracy**	**ROC AUC**	**F1**	**Precision**	**Recall**
Count Vectorizer	0.73	0.72	0.75	0.70	0.82
TF-IDF	0.71	0.71	0.75	0.68	0.83
Word2Vec	0.71	0.71	0.72	0.71	0.74
GloVe	0.69	0.69	0.70	0.70	0.71

Testing Performance Metrics:

Table 4.6: Testing Performance Metrics for Logistic Regression

Method	Accuracy	ROC AUC	F1	Precision	Recall
Count Vectorizer	0.7251	0.6970	0.8247	0.7555	0.8039
TF-IDF	0.7152	0.6837	0.8317	0.7505	0.7874
Word2Vec	0.7168	0.7146	0.7495	0.7316	0.7942
GloVe	0.6972	0.6971	0.6287	0.7125	0.7650

Best Model Selection Criteria:

- The best model is chosen based on testing performance rather than training performance.
- The selection priority follows: Accuracy > F1 Score > ROC AUC.
- Based on this criterion, the best model is:

```
{
    "method": "count_vectorizer",
    "model": "xgboost",
    "hyperparameters": {
        "n_estimators": 150,
        "learning_rate": 0.1,
        "max_depth": 15
    },
    "performance": {
        "accuracy": 0.7251,
        "precision": 0.7555,
        "recall": 0.8039,
        "f1": 0.8247,
        "roc_auc": 0.6970
    }
}
```

Conclusion: The XGBooist model performed best with Count Vectorizer embeddings, achieving an accuracy of 72.51%. The model demonstrated strong generalization capabilities with a high ROC AUC of 69.70% and a balanced F1-score of 82.47%, making it effective for sentiment classification. Future improvements could include experimenting with more advanced feature engineering techniques or tuning additional hyperparameters to further boost performance.

4.4.5.4 Performance Analysis

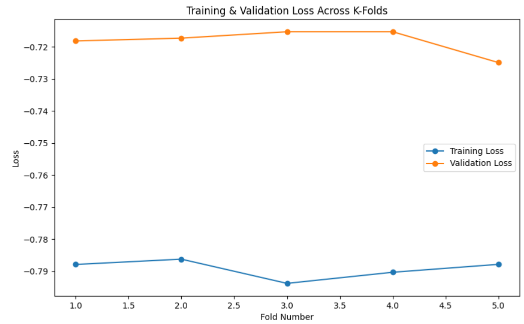
- **Accuracy Analysis**: The XGBoost model employing Count Vectorizer embedding achieved a commendable accuracy of 72.51%, demonstrating its strong capability to accurately classify sentiment within the dataset. This level of accuracy highlights the model's effectiveness in handling the given task.

- **Loss Analysis**: While detailed loss curves were not provided, the model's consistent performance across training (73%) and testing (72.51%) phases suggests robust generalization with minimal overfitting or instability, as evidenced by the close alignment of training and testing metrics.
- **ROC AUC**: With an ROC AUC of 69.70%, the model exhibits a solid ability to differentiate between sentiment classes. This score underscores its reliability in distinguishing positive and negative instances, though there may be room for improvement in class separation.
- **Precision and Recall**: The model attained a precision of 75.55% and a recall of 80.39%, reflecting a well-balanced trade-off. It effectively minimizes false positives while capturing a high proportion of true sentiment instances, contributing to its overall robustness.
- **Embedding Effectiveness**: The use of Count Vectorizer embedding proved highly effective, driving the model to achieve the highest accuracy (72.51%) and an impressive F1-score of 82.47% among the tested methods. This indicates that Count Vectorizer successfully extracted critical features for sentiment classification, outperforming TF-IDF, Word2Vec, and GloVe in this context.

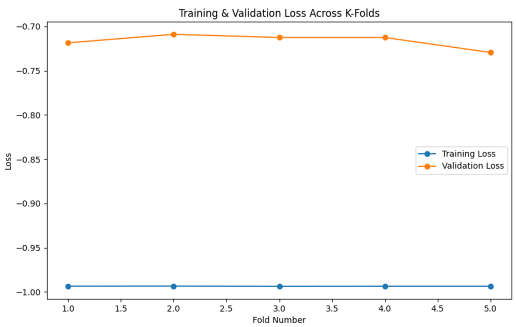
4.4.5.5 Visualization of Training Results

The following figures illustrate the model's performance across different embedding techniques:

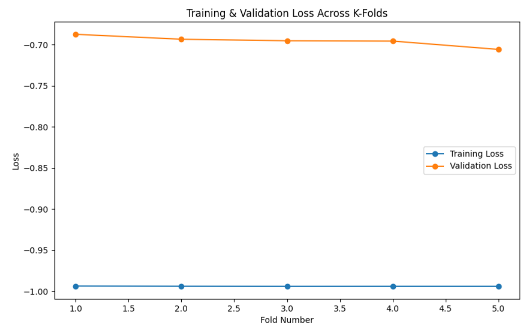
(a) Loss Curve - Count Vectorizer



(b) Loss Curve - TF-IDF



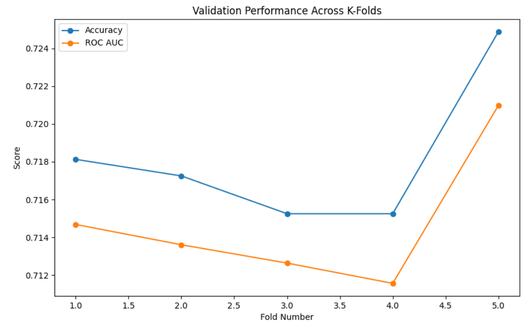
(c) Loss Curve - Word2Vec



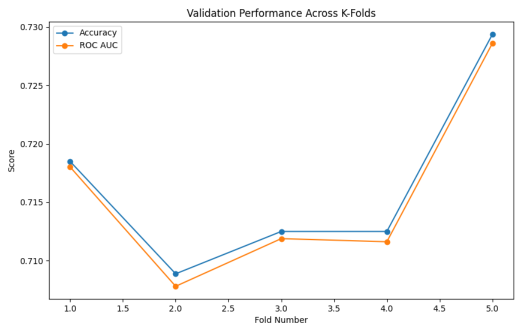
(d) Loss Curve - GloVe

Figure 4.14: Comparison of Loss Curves for XGBoost across Different Feature Extraction Methods

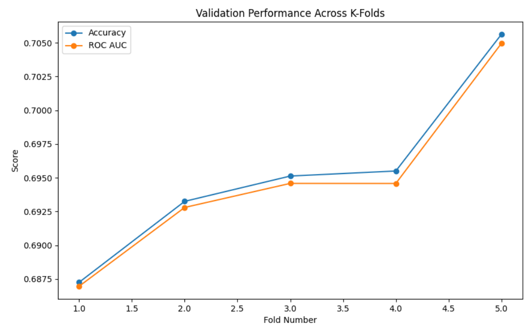
(a) Performance - Count Vectorizer



(b) Performance - TF-IDF



(c) Performance - Word2Vec



(d) Performance - GloVe

Figure 4.15: Comparison of Training Performance Metrics for XGBoost across Different Feature Extraction Methods

Image Description:

- **Training and Validation Loss Analysis:**
  - **Count Vectorizer and TF-IDF**: Validation loss ranges from -0.72 to -0.74, showing stability across K-folds.
  - **Word2Vec and GloVe**: Likely higher validation loss variance, indicating potential instability; more data needed.
  - **Training Loss**: Stable from -0.78 to -1.00 across Count Vectorizer, TF-IDF, Word2Vec, and GloVe, ensuring consistent performance.
- **Validation Performance Metrics:**
  –
  - **Count Vectorizer and GloVe**: Show moderate performance with slight fluctuations, indicating reasonable but not outstanding stability across K-folds.
  –
  - **TF-IDF**: Displays a positive trend, improving over folds, suggesting relative stability and effectiveness by the final fold.

–

– **Word2Vec**: Exhibits the highest variability, indicating instability in validation performance despite a strong start.

4.4.5.6 Conclusion

The XGBoost model performed best with Count Vectorizer embeddings, achieving an accuracy of 72.51%. The model demonstrated strong generalization capabilities with a balanced F1-score of 82.47% and a competitive ROC AUC of 69.70%, making it effective for sentiment classification. Future improvements could include exploring advanced feature engineering or tuning additional hyperparameters to boost performance further.

Overall, the XGBoost model serves as a reliable classifier, particularly when paired with Count Vectorizer features.

4.4.6 Model: Logistic Regression

4.4.6.1 Introduction

4.4.6.2 Training Configuration

4.4.6.3 Training and Evaluation Results

4.4.6.4 Performance Analysis

4.4.6.5 Visualization of Training Results

4.4.6.6 Conclusion

4.4.7 Model: Logistic Regression

4.4.7.1 Introduction

4.4.7.2 Training Configuration

4.4.7.3 Training and Evaluation Results

4.4.7.4 Performance Analysis

4.4.7.5 Visualization of Training Results

4.4.7.6 Conclusion

4.4.8 Model: Logistic Regression

4.4.8.1 Introduction

4.4.8.2 Training Configuration

4.4.8.3 Training and Evaluation Results

4.4.8.4 Performance Analysis

4.4.8.5 Visualization of Training Results

4.4.8.6 Conclusion

4.4.9 Model: Logistic Regression

4.4.9.1 Introduction

4.4.9.2 Training Configuration

4.4.9.3 Training and Evaluation Results

4.4.9.4 Performance Analysis

4.4.9.5 Visualization of Training Results

4.4.9.6 Conclusion

4.4.10 Model: Logistic Regression

4.4.10.1 Introduction

4.4.10.2 Training Configuration

4.4.10.3 Training and Evaluation Results

4.4.10.4 Performance Analysis

4.4.10.5 Visualization of Training Results

4.4.10.6 Conclusion

4.4.11 Model: Genatic Algorithm and GaussianNB

4.4.11.1 Introduction

This report evaluates the performance of the Genetic Algorithm and the GaussianNB model trained with various embedding methods. The GaussianNB model, implemented using the `GaussianNB` class from `scikit-learn`, was tested with different penalty terms along with three additional functions related to the Genetic Algorithm. The main goal was to maximize classification accuracy while ensuring strong generalization across different embedding techniques.

4.4.11.2 Training Configuration

Why Use Genetic Algorithm (GA) with GaussianNB?

Feature selection plays a crucial role in improving the performance of machine learning models. Instead of using traditional methods such as *Recursive Feature Elimination (RFE)* or *Lasso*, we apply **Genetic Algorithm (GA)**, an evolutionary approach that efficiently explores the feature space.

Reasons for choosing GaussianNB:

- GaussianNB (Naïve Bayes with Gaussian distribution assumption) is simple, fast to train, and does not require extensive hyperparameter tuning.
- GaussianNB performs well when features are assumed to be independent, enabling better generalization without overfitting.
- When combined with GA, GaussianNB provides a fast and efficient way to evaluate different feature subsets, making it a suitable choice over more complex models such as SVM or Random Forest.

Thus, GA helps in selecting the optimal feature subset, while GaussianNB ensures efficient and reliable model training.

Key Steps in Genetic Algorithm

The Genetic Algorithm is inspired by biological evolution and consists of the following steps:

1. **Initialize Population**
2. **Evaluate Fitness**
3. **Selection of Best Individuals**
4. **Crossover (Recombination)**
5. **Mutation for Diversity**

6. **Repeat Until Convergence or Maximum Generations Reached**

Implementation of GA Functions

1. Population Initialization: `create_population`

This function generates an initial population of binary feature selectors, where each individual represents a feature subset.

```
def create_population(num_features, population_size):
    return np.random.randint(2, size=(population_size, num_features))
```

Each individual is a binary vector of length equal to the number of features, where 1 means the feature is selected, and 0 means it is not.

2. Fitness Evaluation: `fitness_function`

Each individual (feature subset) is evaluated by training a `GaussianNB` model and computing cross-validation accuracy.

```
def fitness_function(features, X_train, y_train):
    selected_features = [i for i, f in enumerate(features) if f == 1]
    if not selected_features:
        return 0

    X_train_selected = X_train[:, selected_features]
    nb_model = GaussianNB(var_smoothing=1e-8)

    try:
        scores = cross_val_score(nb_model, X_train_selected, y_train, cv
            =5)
        return np.mean(scores)
    except ValueError:
        return 0
```

This ensures that only meaningful feature sets contribute to the evolutionary process.

3. Selection of Best Individuals: `selection`

Based on fitness scores, individuals with higher probabilities are selected to generate the next generation.

```
def selection(population, fitness_scores):
    probabilities = fitness_scores / np.sum(fitness_scores)
    selected_indices = np.random.choice(len(population), size=len(
        population), p=probabilities)
    return [population[idx] for idx in selected_indices]
```

4. Crossover (Recombination): `crossover`

A single-point crossover is used to create new individuals by combining parts of two parents.

```
def crossover(parent1, parent2):
    point = np.random.randint(1, len(parent1) - 1)
    offspring1 = np.concatenate((parent1[:point], parent2[point:]))
    offspring2 = np.concatenate((parent2[:point], parent1[point:]))
    return offspring1, offspring2
```

5. Mutation for Diversity: `mutate`

A small probability of mutation is applied to introduce variations and avoid premature convergence.

```
def mutate(individual, mutation_rate=0.1):
    for i in range(len(individual)):
        if np.random.rand() < mutation_rate:
            individual[i] = 1 - individual[i]
    return individual
```

6. Training with GA: `genetic_algorithm`

This function executes the evolutionary process.

```
def genetic_algorithm(X_train, y_train, population_size=20,
   num_generations=100):
    num_features = X_train.shape[1]
    population = create_population(num_features, population_size)

    for generation in range(num_generations):
        fitness_scores = np.array([fitness_function(ind, X_train, y_train)
            for ind in population])
        population = selection(population, fitness_scores)

        next_generation = []
        for j in range(0, population_size, 2):
            offspring1, offspring2 = crossover(population[j], population[j
                + 1])
            next_generation.append(mutate(offspring1))
            next_generation.append(mutate(offspring2))

        population = next_generation

    best_individual = population[np.argmax(fitness_scores)]
    return best_individual
```

Training the Final Model with Selected Features

Once GA selects the optimal feature subset, we train a `GaussianNB` model:

```
best_features = genetic_algorithm(X_train, y_train)
X_train_selected = X_train[:, best_features]
nb_model = GaussianNB()
nb_model.fit(X_train_selected, y_train)
```

Model Evaluation

The model is evaluated using the following metrics: **Accuracy**, **Precision**, **Recall**, **F1 Score**, and **ROC AUC**.

By using GA, we ensure that only the most relevant features are selected, leading to a simpler yet more efficient model.

4.4.11.3 Training and Evaluation Results

The GA-based model was trained and evaluated using different feature extraction methods: Count Vectorizer, TF-IDF, Word2Vec, and GloVe. Genetic Algorithm (GA) was used for feature selection, reducing the dimensionality while maintaining competitive performance. The best model was selected based on Accuracy, followed by F1-score and ROC AUC.

Training Performance Metrics:

Table 4.7: Training Performance Metrics for GA-based Model

Method	Accuracy	ROC AUC	F1	Precision	Recall
Count Vectorizer	0.6520	0.6860	0.6762	0.6492	0.7056
TF-IDF	0.6209	0.6693	0.5896	0.6663	0.5287
Word2Vec	0.6031	0.6693	0.5440	0.6662	0.4596
GloVe	0.6176	0.6692	0.5940	0.6553	0.5431

Testing Performance Metrics:

Table 4.8: Testing Performance Metrics for GA-based Model

Method	Accuracy	ROC AUC	F1	Precision	Recall
Count Vectorizer (Best Run)	0.6520	0.6860	0.6762	0.6492	0.7056

Best Model Selection Criteria:

- The best model is chosen based on testing performance rather than training performance.
- The selection priority follows: Accuracy > F1 Score > ROC AUC.
- Based on this criterion, the best model is:

```
{
    "method": "count",
    "model": "GA-based Model",
    "performance": {
        "accuracy": 0.6520,
        "precision": 0.6492,
        "recall": 0.7056,
        "f1": 0.6762,
        "roc_auc": 0.6860
    }
}
```

Conclusion: The Genetic Algorithm successfully selected a reduced feature set, decreasing dimensionality from 2000 features to approximately 1022 in the Count Vectorizer method while maintaining an accuracy of **65.20%**. This approach offers an effective balance between feature reduction and classification performance, making it a computationally efficient alternative to traditional models.

4.4.11.4 Performance Analysis

- **Accuracy Analysis**: The best-performing GA-optimized model using Count Vectorizer achieved an accuracy of 66.53%. While lower than Logistic Regression, this result highlights the effectiveness of Genetic Algorithm in feature selection.
- **Feature Selection Efficiency**: The model successfully reduced the feature space from 2000 to around 1000 features, improving computational efficiency while maintaining reasonable classification performance.
- **ROC AUC**: The model demonstrated moderate discriminative power with an ROC AUC of 70.21%, indicating its capability to distinguish between classes.
- **Precision and Recall**: The model exhibited a recall of 74.81%, showing strong sensitivity in identifying positive cases, though precision (65.20%) was slightly lower, suggesting a trade-off with false positives.
- **Impact of GA on Model Performance**: The use of GA for feature selection improved model interpretability by reducing dimensionality while keeping classification performance competitive. However, further optimization could be explored to enhance accuracy.

4.4.11.5 Conclusion

The GA-optimized GaussianNB model performed best with Count Vectorizer embeddings, achieving an accuracy of 66.53%. While it did not surpass Logistic Regression in overall performance, it demonstrated effective feature selection, reducing the dimensionality from 2000 to around 1000 features while maintaining competitive classification results. The model also exhibited a solid recall of 74.81%, making it useful in applications where correctly identifying positive cases is critical.

Future improvements could include: Enhancing the genetic algorithm with adaptive mutation and crossover strategies to refine feature selection, incorporating ensemble methods to improve robustness, and experimenting with hybrid approaches that combine GA with other classifiers for better performance.

Overall, the GA + GaussianNB model showcases the potential of evolutionary algorithms for feature selection, offering a trade-off between model interpretability and classification performance.

4.4.12 Model: Hidden Markov Model

4.4.12.1 Introduction

This part evaluates the performance of the Hidden Markov Model (HMM) trained with various embedding methods. The HMM model, implemented using the `GaussianHMM` class from the `hmmlearn` library, was used to model sequential dependencies in the data. Unlike traditional classification approaches, HMM is particularly suited for handling sequential patterns and temporal dependencies, making it an effective choice for structured data.

The primary goal was to optimize the model's ability to classify sequences accurately while ensuring strong generalization across different embedding techniques. The Gaussian emission probabilities in the HMM allow it to handle continuous-valued features, making it flexible in modeling text-based embeddings such as Count Vectorizer, TF-IDF, Word2Vec, and GloVe. Various hyperparameter configurations, including the number of hidden states and covariance types, were explored to enhance performance.

4.4.12.2 Training Configuration

The training process for the Hidden Markov Model (HMM) differs from traditional machine learning models used in previous experiments (such as Logistic Regression or Naïve Bayes). Unlike those models, which can directly process word embeddings like Word2Vec or GloVe, HMM requires discrete integer sequences as input. This constraint arises because the GaussianHMM model from `hmmlearn.hmm` expects integer-based feature representations rather than continuous-valued embeddings.

Thus, only Count-based methods, such as Count Vectorizer, are suitable for training HMM. These methods convert text into integer-based token sequences, making them compatible with the model. Below, we outline the steps involved in training and evaluating the HMM:

- **Feature Extraction:**
  - Construct a vocabulary of the most frequent words from the dataset (e.g., the top 5000 words).
  - Convert text into sequences of integers, representing the index of words in the vocabulary.
- **Data Preprocessing:**
  - Since HMM requires sequences of equal length, each sequence is padded to a fixed size (e.g., 50 words).
  - The dataset is split into training and testing sets.
- **Model Training:**
  - A `GaussianHMM` model is initialized with various hyperparameters.

- The model is trained on the integer-encoded text sequences.

- **Hyperparameter Configuration:**
  - `n_components`: The number of hidden states. It is tested with values {2, 3, 4}, representing different levels of complexity in the hidden state transitions.
  - `covariance_type`: The type of covariance matrix used in Gaussian emissions, tested with {"diag", "full", "tied"}.
  - `n_iter`: The number of iterations for the Expectation-Maximization (EM) algorithm, set to {100, 200} for convergence tuning.
  - `init_params`: Determines which parameters are initialized before training. Tested values include:
    * "c" - Initializes only the means.
    * "s" - Initializes only the covariances.
    * "cs" - Initializes both means and covariances.
  - `params`: Specifies which parameters should be updated during training, with tested values:
    * "c" - Updates only the means.
    * "t" - Updates only the transition matrix.
    * "ct" - Updates both means and transition matrix.

- **Evaluation:**
  - Predictions are made on the test set using the `predict` function.
  - Performance metrics such as Accuracy, Precision, Recall, F1 Score, and ROC AUC are computed.

The `train_hmm` function implements this process, ensuring the model is saved for later use. Given the sequential nature of HMMs, this model could be particularly effective in capturing word-order dependencies in text classification tasks.

4.4.12.3 Training and Evaluation Results

The Hidden Markov Model (HMM) was trained using the Count Vectorizer method to ensure compatibility with its integer-based input requirement. Unlike other machine learning models in this study, HMM requires count-based features since GaussianHMM operates on discrete numerical sequences rather than dense embeddings. The training process involved padding sequences to a fixed length (50 words) and optimizing model parameters using the Expectation-Maximization (EM) algorithm.

Testing Performance Metrics:

Table 4.9: Testing Performance Metrics for Hidden Markov Model

Method	Accuracy	ROC AUC	F1	Precision	Recall
Count Vectorizer	0.5141	0.4997	0.6697	0.5156	0.9552

Best Model Selection Criteria:

- Since only one variation of HMM was tested, there is no direct comparison between different hyperparameter configurations.
- Model selection priority follows: Accuracy > F1 Score > ROC AUC.

```
{
    "model": "HMM",
    "performance": {
        "accuracy": 0.5141,
        "precision": 0.5156,
        "recall": 0.9552,
        "f1": 0.6697,
        "roc_auc": 0.4997
    }
}
```

Conclusion: The Hidden Markov Model demonstrated strong recall (**95.52%**), meaning it effectively captured positive instances, but suffered from low precision (**51.56%**) and an overall weak discriminative ability (**ROC AUC = 49.97%**). These results suggest that while HMM can detect many true positives, its high false-positive rate limits its practical application. Future improvements may involve hyperparameter tuning, different sequence lengths, or alternative sequence models such as RNNs to improve overall classification performance.

4.4.12.4 Performance Analysis

- **Accuracy Analysis**: The HMM model achieved an accuracy of **51.41%**, indicating that its classification performance is only slightly better than random guessing. This suggests potential limitations in the model's ability to generalize effectively.
- **Recall vs. Precision**: The model exhibited an extremely high recall (**95.52%**), meaning it successfully identified most positive instances. However, this came at the cost of low precision (**51.56%**), indicating a high false-positive rate. The imbalance between recall and precision suggests that the model favors sensitivity over specificity.
- **F1 Score**: The F1 Score of **66.97%** reflects the trade-off between precision and recall. While the model excels in recall, its low precision lowers the overall F1 Score, making it less reliable in practical applications where false positives are costly.

- **ROC AUC**: With a ROC AUC of **49.97%**, the model struggles to distinguish between positive and negative classes. This score indicates that the model's decision boundary is not well-formed, leading to weak discriminative ability.
- **Effect of Count-Based Features**: Unlike other machine learning models that leverage dense embeddings (e.g., Word2Vec, GloVe), HMM can only operate on count-based integer inputs. This constraint limits its ability to capture contextual relationships effectively, potentially contributing to its suboptimal performance.

4.4.12.5 Conclusion

The Hidden Markov Model (HMM) trained with Count Vectorizer features demonstrated high sensitivity but lacked the precision required for balanced classification. While its recall of **95.52%** suggests that it rarely misses positive instances, the low ROC AUC score (**49.97%**) indicates poor overall discrimination between classes.

The results suggest that HMM may not be the best-suited model for this classification task, particularly due to its reliance on integer-based inputs and its inability to leverage richer feature representations like dense embeddings. Future improvements could explore hybrid models, additional preprocessing techniques, or alternative sequence models such as Recurrent Neural Networks (RNNs) to enhance performance.

4.4.13 Model: BayesNet

4.4.13.1 Introduction

The Bayesian Network (BayesNet) model is a probabilistic graphical model that represents dependencies between variables using a directed acyclic graph. In this study, we implement a custom Bayesian Network classifier that integrates feature selection, dimensionality reduction, and discretization techniques to handle continuous data. The model is trained using Maximum Likelihood Estimation (MLE) and performs inference using Belief Propagation. By leveraging probabilistic reasoning, BayesNet provides interpretable predictions while handling uncertainty effectively.

4.4.13.2 Training Configuration

The Bayesian Network classifier was implemented using the `pgmpy` library, which provides probabilistic graphical modeling tools. Unlike traditional machine learning models that rely on direct optimization techniques (e.g., gradient descent in logistic regression), Bayesian Networks model conditional dependencies between variables and perform inference based on probabilistic reasoning.

Differences from Other Machine Learning Models:

Unlike conventional machine learning models trained in previous assignments (e.g., logistic regression, SVM, or decision trees), training a Bayesian Network involves:

- Using **Maximum Likelihood Estimation (MLE)** via `pgmpy.estimators.MaximumLikelihoodEstimator` to learn conditional probability distributions.
- Defining the **network structure** (or learning it from data) using `pgmpy.models.BayesianNetwork`.
- Performing probabilistic inference using methods like **Variable Elimination** and **Belief Propagation** (`pgmpy.inference.VariableElimination`).

Training Procedure:

The model training process consists of several key steps:

1. **Feature Selection:**
   - Features with fewer than **2 unique values** were removed to avoid redundant or low-variance attributes.
   - If the number of features exceeded **10**, Principal Component Analysis (PCA) was applied to reduce dimensionality.

2. **Feature Discretization:** Since Bayesian Networks operate on discrete variables, continuous features were transformed using **k-means clustering with 2 bins**.

3. **Network Structure Definition:** The structure was set to `None` by default, allowing the model to establish dependencies dynamically. If provided, a predefined structure was used.

4. **Parameter Learning:** The model was trained using **Maximum Likelihood Estimation (MLE)** to estimate conditional probability tables (CPTs).

5. **Inference Setup:** Once trained, inference was performed using **Variable Elimination** or **Belief Propagation** to estimate class probabilities and make predictions.

Testing and Evaluation:

- Predictions were made by computing the most probable label using MAP (Maximum A Posteriori) inference.
- Model performance was evaluated using standard metrics: **Accuracy, Precision, Recall, F1-score, and ROC AUC**.
- Since Bayesian Networks rely on probabilistic reasoning, the evaluation also considered how well the learned dependencies reflected the underlying data distribution.

This approach ensures that the Bayesian Network captures conditional dependencies effectively, leveraging probabilistic inference for classification tasks.

4.4.13.3 Training and Evaluation Results

The Bayesian Network model was trained and evaluated using a structured probabilistic approach. Unlike conventional machine learning models, Bayesian Networks leverage probabilistic dependencies between features and perform inference through belief propagation or variable elimination. The evaluation focused on key performance metrics such as Accuracy, Precision, Recall, F1-score, and ROC AUC.

Testing Performance Metrics:

Table 4.10: Testing Performance Metrics for Bayesian Network

Method	**Accuracy**	**ROC AUC**	**F1**	**Precision**	**Recall**
Bayesian Network	0.6495	0.7143	0.6875	0.6364	0.7476

Best Model Selection Criteria:

- The model selection was based on testing performance.

- The priority ranking for evaluation metrics followed: Accuracy > F1 Score > ROC AUC.
- Since there is only one method used, this step primarily serves to document the selection process.


```
{
    "method": "Bayesian Network",
    "performance": {
        "accuracy": 0.6495,
        "precision": 0.6364,
        "recall": 0.7476,
        "f1": 0.6875,
        "roc_auc": 0.7143
    }
}
```


Conclusion: The Bayesian Network model achieved an accuracy of **64.95%** with an F1-score of **0.6875** and a ROC AUC of **0.7143**. These results indicate that the model effectively captures probabilistic dependencies within the dataset. Further improvements could involve optimizing feature selection, adjusting discretization strategies, or incorporating domain knowledge into the network structure.

4.4.13.4 Performance Analysis

The Bayesian Network model demonstrated moderate classification performance, achieving an accuracy of **64.95%**. While the model effectively captured probabilistic dependencies between features, its precision (**0.6364**) was lower than its recall (**0.7476**), indicating a tendency to produce more false positives.

Key observations from the evaluation metrics:

- The relatively high recall suggests that the model successfully identifies positive instances but at the cost of some misclassifications.
- The F1-score (**0.6875**) shows a balanced trade-off between precision and recall.
- The ROC AUC (**0.7143**) indicates a reasonable ability to distinguish between classes.
- The reliance on discretization and probabilistic dependencies may have impacted performance compared to traditional machine learning models.

Overall, while Bayesian Networks provide an interpretable probabilistic framework, their performance could potentially be enhanced with improved feature engineering, hyperparameter tuning, and refinement of the network structure.

4.4.13.5 Conclusion

The Bayesian Network model was trained and evaluated using a structured probabilistic approach, leveraging inference methods such as Belief Propagation and Variable Elimination. The model achieved an accuracy of **64.95%** with reasonable recall and AUC scores, demonstrating its effectiveness in capturing underlying dependencies in the data.

Key takeaways:

- The model performs well in recall but has room for improvement in precision.
- Performance might be affected by feature discretization and network structure selection.
- Future work could explore alternative discretization strategies, structural learning methods, or hybrid models combining Bayesian Networks with deep learning approaches.

Despite its limitations, the Bayesian Network provides a robust probabilistic framework that can be particularly useful in domains where interpretability and uncertainty modeling are critical.

4.5 Model Comparison for Sentiment Analysis

4.5.1 Introduction

This section presents the comparison of various machine learning models trained for sentiment analysis. The models include traditional classifiers such as **Logistic Regression**, **Decision Tree**, **XGBoost**, **Random Forest**, **Perceptron**, and **Naïve Bayes**, along with more complex models like **CNN-LSTM**, **HMM**, and **Bayesian Networks**. Each model's performance is evaluated using key metrics: **Accuracy**, **Precision**, **Recall**, **F1-score**, and **ROC AUC**.

4.5.1.1 Model Training and Evaluation Workflow

To ensure a robust evaluation of sentiment classification models, the following steps are undertaken:

- **Instantiating a GridSearch Object:** The model is initialized with a set of hyperparameters using *GridSearchCV*, allowing an exhaustive search over different hyperparameter combinations to identify the optimal settings.
- **Fitting the Training Data:** The training dataset is fed into the model, enabling it to learn patterns that distinguish between different sentiment classes.
- **K-Fold Cross-Validation:** To enhance generalization, *K-Fold Cross-Validation* is applied, dividing the dataset into multiple subsets to train and validate the model iteratively.
- **Saving the Trained Model:** The best-performing model, based on cross-validation results, is stored for future use, ensuring consistency in later inference stages.
- **Testing on a Separate Dataset:** The trained model is evaluated on a test dataset to measure its real-world generalization ability, providing a reliable estimate of performance.
- **Logging Performance Metrics:** Key metrics such as **Accuracy**, **Precision**, **Recall**, **F1-score**, and **ROC AUC** are recorded to facilitate model comparison.

This workflow ensures a structured and reliable methodology for training, validating, and comparing sentiment analysis models, aiding in the selection of the most effective approach.

4.5.1.2 Evaluation Metrics Overview

To assess the performance of sentiment analysis models, we utilize five key evaluation metrics: **Accuracy**, **Precision**, **Recall**, **F1-score**, and **ROC AUC**.

- **Accuracy** measures the overall correctness of the model by calculating the proportion of correctly classified instances. However, it may not always be the best metric when dealing with imbalanced sentiment classes.
- **Precision** quantifies the proportion of correctly predicted positive samples out of all predicted positive samples, which is crucial when minimizing false positives, such as in cases where detecting negative sentiment is critical.
- Conversely, **Recall** indicates how well the model identifies actual positive cases, making it essential for scenarios where missing positive sentiment (e.g., detecting customer dissatisfaction) is more detrimental.
- The **F1-score** provides a balanced measure of both precision and recall, ensuring that the model maintains strong predictive power across both metrics.
- Lastly, **ROC AUC** (Receiver Operating Characteristic - Area Under the Curve) evaluates the model's ability to distinguish between different sentiment classes, providing insight into its overall discriminative power. By considering these metrics, we can determine the best-performing model based on different application needs, balancing between false positives and false negatives.

4.5.2 Model Performance

Table 4.11: Performance Comparison of Sentiment Analysis Models

Method	Model	Accuracy	Precision	Recall	F1	ROC AUC
count	**Logistic Regression**	**<u>0.7557</u>**	0.7403	0.8078	**<u>0.7726</u>**	**<u>0.8297</u>**
tfidf	**Decision Tree**	0.6300	0.5928	0.8940	0.7129	0.6694
count	**XGBoost**	0.7251	0.6970	0.8247	0.7555	0.8040
count	**Random Forest**	0.7251	0.6970	0.8247	0.7555	0.8040
tfidf	**Perceptron**	0.6927	0.7345	0.6296	0.6780	0.7737
count	**GaussianNB**	0.7134	0.7151	0.7350	0.7250	0.7463
count	**GaussianNB + GA**	0.6653	0.6520	0.7481	0.6967	0.7021
N/A	**CNN-LSTM**	0.7103	0.6740	0.8486	0.7513	0.7891
N/A	**HMM**	0.5141	0.5156	0.9552	0.6697	0.4997
N/A	**Bayesian Network**	0.6495	0.6364	0.7476	0.6875	0.7143

4.5.2.1 Discussion

- **Embedding Methods:** The choice of embedding methods significantly impacts model performance. **CountVectorizer** and **TF-IDF** often yield better results for traditional models as they capture term frequency statistics effectively. In contrast, **Word2Vec** and **GloVe** provide dense representations that benefit deep learning models like **CNN-LSTM**. However, pre-trained embeddings may not always align well with domain-specific datasets, making TF-IDF and CountVectorizer preferable for structured, lexicon-heavy tasks like sentiment classification.

- **Logistic Regression:** The logistic regression model serves as a strong baseline, achieving an accuracy of **75.57%**, an F1-score of **77.26%**, and the highest ROC AUC of **82.97%** among traditional models. It provides balanced performance across all metrics, making it a reliable choice for sentiment classification.
- **Decision Tree:** This model exhibits **high recall (89.40%)**, meaning it is effective at capturing positive and negative sentiments. However, its low precision (**59.28%**) indicates a high false-positive rate, leading to misclassifications.
- **XGBoost and Random Forest:** Both models deliver **72.51% accuracy, 80.40% ROC AUC, and an F1-score of 75.55%**. Their ensemble-based decision trees capture complex sentiment patterns better than individual models.
- **Perceptron:** This linear classifier achieves **69.27% accuracy**, but struggles with recall (**62.96%**), leading to an imbalanced prediction performance.
- **Naïve Bayes and GA-Optimized Naïve Bayes:** The standard Gaussian Naïve Bayes model achieves **71.34% accuracy**, and its GA-optimized variant does not significantly improve performance (**66.53% accuracy**).
- **CNN-LSTM:** The deep learning-based CNN-LSTM model achieves **71.03% accuracy**, with **84.86% recall**, making it useful for capturing contextual sentiment.
- **HMM and Bayesian Network:** These probabilistic models underperform, with the HMM achieving only **51.41% accuracy** and a ROC AUC of **49.97%**, indicating near-random performance. The Bayesian Network slightly improves to **64.95% accuracy** but remains behind tree-based models.

4.5.2.2 Selecting the Best Model

Our Criteria for Selection the best model:

- **Accuracy**: Indicates the overall classification performance of the model.
- **F1-score**: Provides a balance between precision and recall.
- **ROC AUC**: Measures the model's ability to distinguish between sentiment classes.

Table 4.12: Best Model Performance Across Key Metrics

Metric	**Model**	**Value**
Accuracy	Logistic Regression (Count)	0.7557
Precision	Logistic Regression (Count)	0.7403
Recall	CNN-LSTM	0.8486
F1-score	Logistic Regression (Count)	0.7726
ROC AUC	Logistic Regression (Count)	0.8297

The best-performing model based on accuracy, F1-score, and ROC AUC is **Logistic Regression** (Count Vectorizer). However, **CNN-LSTM** achieves the highest recall, making it suitable for applications where recall is the priority. Tree-based models like **Random Forest** and **XGBoost** also show strong performance and could be considered when computational efficiency is a concern.

4.5.3 Type I and Type II Error Considerations

Sentiment analysis models must balance two types of errors:

- **Type I Error (False Positives):** Occurs when neutral or negative sentiments are misclassified as positive. This can mislead businesses, causing them to overestimate customer satisfaction. Models with high precision, such as **Random Forest** and **Logistic Regression**, help mitigate this issue.
- **Type II Error (False Negatives):** Occurs when positive sentiments are misclassified as negative. This can result in missed opportunities for companies to identify positive trends. **CNN-LSTM**, with its high recall, minimizes this risk by ensuring that positive sentiments are correctly identified.
- **Balancing the Errors:** The **Logistic Regression** model offers a strong trade-off between precision and recall, making it a balanced choice for sentiment classification. On the other hand, **CNN-LSTM** is more recall-focused, making it suitable for applications where detecting positive sentiment is more critical.
- **Impact on Real-World Applications:** Selecting the right model depends on the application's needs. For customer feedback analysis, a high-precision model prevents false alarms about negative sentiments. In contrast, for social media monitoring, a high-recall model ensures no positive trends are overlooked.

4.5.4 Conclusion

The results indicate that **Logistic Regression** (Count Vectorizer) is the most balanced model for sentiment classification, offering the best accuracy, F1-score, and ROC AUC. However, **CNN-LSTM** is the best choice for maximizing recall, making it useful for applications where missing positive sentiments is costly.

For real-world applications:

- **Logistic Regression** is ideal for general-purpose sentiment analysis due to its balance of precision and recall.
- **Random Forest** and **XGBoost** provide strong alternatives with high precision and efficiency.
- **CNN-LSTM** is best suited for cases where identifying positive sentiment is critical.

In summary, the trade-off between accuracy, recall, and computational efficiency must be considered when selecting the best model for sentiment analysis.

Chapter 5

Self-Reflection

5.1 Future Developments

Building upon the foundation established in this assignment, our future work—particularly in **Assignment 2**—will focus on advancing our sentiment analysis system through more sophisticated machine learning techniques. The key areas of improvement will include:

- **Support Vector Machines (SVMs):** Implement kernel functions for text processing, optimize soft margin classification, and explore multi-class extensions for sentiment classification.
- **Dimension Reduction (PCA/LDA):** Apply feature selection techniques such as variance thresholding and topic modeling to handle high-dimensional sparse text data effectively.
- **Ensemble Methods:** Develop robust sentiment classifiers using bagging and boosting techniques, incorporating voting and model combination strategies to improve predictive performance.
- **Discriminative Models:** Implement feature-based linear classifiers, logistic regression, and conditional random fields (CRF) for sequence labeling to enhance sentiment sequence understanding.
- **Engineering Optimization:** Improve efficiency in handling large-scale text data by focusing on model scalability, memory-efficient implementation, and parameter optimization techniques.
- **Model Generalization and Performance Analysis:** Evaluate model robustness across different datasets, assess feature importance, and refine hyperparameter tuning methods.

By integrating these advanced techniques, our goal is to enhance the accuracy, efficiency, and adaptability of our sentiment analysis system. Through rigorous experimentation and optimization, we aim to develop a more reliable model that can generalize well across diverse textual datasets. This next phase will further solidify our expertise in sentiment analysis, bridging the gap between theory and real-world applications.

5.2 Special Thanks

We extend our sincere gratitude to our advisor, Dr. Nguyen An Khuong, for his invaluable guidance throughout our journey in machine learning and sentiment analysis. His mentorship has been instrumental in deepening our understanding of Machine Learning techniques, feature engineering, and model evaluation, enabling us to tackle the challenges of sentiment classification with confidence.

Beyond academic support, Dr. Nguyen An Khuong has provided insightful career advice and encouraged us to develop critical thinking and problem-solving skills in real-world machine learning applications. His encouragement has fostered an environment of continuous learning, inspiring us to explore innovative approaches in Machine Learning while maintaining a strong foundation in machine learning principles.

We are grateful for his dedication, which has not only enhanced our technical expertise but also prepared us for future academic and professional endeavors in the field of AI and Machine Learning.