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


# MACHINE LEARNING (CO3117)

## Report:

# Assignment 2

**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, APRIL 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 2** 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 MODELS. 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; Participate and Ensure everything stays on schedule and verify all work done by other members. |
| 2 | Pham Huynh Bao Dai | 2252139 | Model training and evaluating; Model implementation (MODELS); Document Model Implementation. |
| 3 | Nguyen Thanh Dat | 2252145 | Model training and evaluating; Model implementation (MODELS); Document Model Implementation. |
| 4 | Nguyen Tien Hung | 2252280 | Model training and evaluating; Model training and evaluating; Model implementation (MODELS); Document Model Implementation. |
| 5 | Nguyen Thien Loc | 2252460 | Visualization; Model evaluation; hyperparameter tuning; Model Comparison; Document Performance analysis. |

## 3.2 Project Organization and Requirements

The project follows structured collaboration and 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 submitted README file is stored in the repository at: https://github.com/pdz1804/ML_LHPD2/blob/main/notebooks/assignment1/ML_LHPD2_Ass1_README.md
- The team's report for Assignment 1 is located in: https://github.com/pdz1804/ML_LHPD2/tree/main/reports/final_project/
- 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:

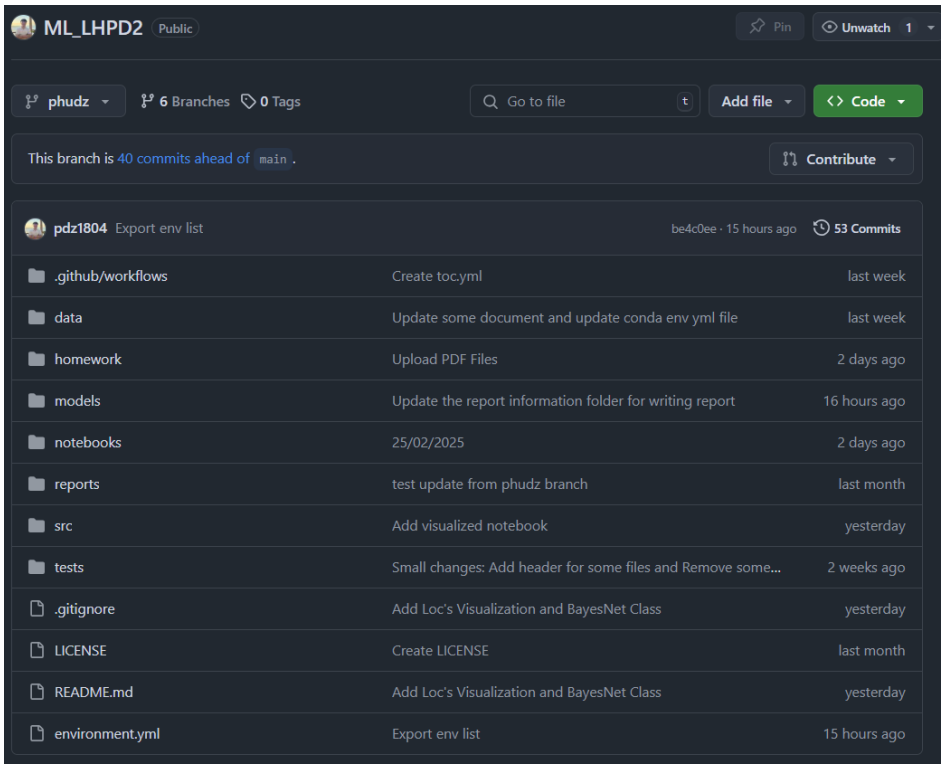



Figure 3.1: Github Repository Structure

### 3.3.1 How to Run This Project

#### 3.3.1.1 Clone the Repository

```
git clone https://github.com/pdz1804/ML_LHPD2
cd ML_LHPD2
```

#### 3.3.1.2 Environment Setup

To train the model locally, install dependencies using the provided `environment.yml` file. Ensure Conda is installed first.

```
conda env create -f environment.yml
conda activate ml_env  # Replace ml_env with your environment name
```

### 3.3.2 Project Exploration

The project structure contains several key components:

- **Data Collection:** To know how we collected all the data, visit the `data` folder.
- **Final Preprocessed Dataset:** The final preprocessed dataset can be found at this link: Tweets Clean PosNeg v1.
- **Preprocessing:** To learn how we preprocess, make all datasets consistent to merge:
  - Notebook: `src/data/process.ipynb`
  - Utility file: `src/data/preprocess.py`
- **Feature Engineering:** To know how we create features, visit:
  - Utility file: `src/features/build_features_utils.py`
- **Model Training and Evaluation:** To understand hyperparameter tuning, k-fold cross-validation, and model training:
  - Hyperparameter tuning utility: `src/models/models_utils.py`
  - Training notebook: `src/models/train_model.ipynb`
- **Visualization Functions:** We have created visualization functions in the folder:
  - Location: `src/visualization/`
- **Dependencies:** The file `environment.yml` contains details about libraries or dependencies required for executing this project.

# Chapter 4

# Code Engineering

## 4.1 Datasets

This project utilizes three key datasets for training and testing the model in order to test and analyze sentiment scores using every model in the syllabus. These datasets have been curated to provide a comprehensive range of content for robust sentiment analysis.Below are the descriptions of the datasets:

### 4.1.1 sentiment140-dataset

The **Sentiment140 dataset** is a large-scale collection of **1,600,000 tweets** obtained using the **Twitter API**. This dataset is designed for **sentiment analysis** and has been preprocessed by removing emoticons to provide a cleaner textual representation of tweets.

#### 4.1.1.1 Dataset Description

The dataset was created using the Twitter API and contains a large collection of tweets labeled for sentiment analysis. Sentiments were automatically assigned using distant supervision, leveraging emoticons as indicators of sentiment polarity. The dataset is widely used for machine learning applications in sentiment classification.

#### 4.1.1.2 Dataset Format

The dataset contains six fields, each representing specific attributes of a tweet:

- **target**: Sentiment label of the tweet (0 = Negative, 2 = Neutral, 4 = Positive).
- **ids**: Unique tweet ID.
- **date**: Timestamp of the tweet (e.g., `Sat May 16 23:58:44 UTC 2009`).

- **flag**: Query keyword used to retrieve the tweet (e.g., `lyx`). If no keyword was used, this field contains `NO_QUERY`.
- **user**: Username of the person who tweeted (e.g., `robotickilldozr`).
- **text**: Actual content of the tweet (e.g., `Lyx is cool`), with emoticons removed for sentiment classification.

#### 4.1.1.3 Dataset Notes

- **Preprocessed Version**: This version of the dataset (`raining.1600000.processed.noemoticon.csv`) has **no emoticons**, making it suitable for text-based sentiment analysis without relying on emoticon cues.
- **Large-scale Dataset**: The dataset contains **1.6 million tweets**, making it one of the largest sentiment analysis datasets available.
- **Automated Labeling**: Since sentiment was assigned based on emoticons, there may be **biases** or **inaccuracies** in certain cases.
- **Historical Data**: The dataset was collected in **2009**, meaning language patterns and sentiment expressions may differ from modern Twitter usage.

The link to this dataset can be found here: https://www.kaggle.com/datasets/kazanova/sentiment140

### 4.1.2 twitter-tweets-sentiment-dataset

The **Twitter Tweets Sentiment Dataset** is a dataset designed for **sentiment analysis in natural language processing (NLP)**. It contains a collection of tweets labeled with sentiment polarity, which can be used to develop models for sentiment classification.

#### 4.1.2.1 Dataset Description

The dataset was sourced from Kaggle competitions and includes labeled tweets aimed at detecting positive, neutral, or negative sentiments. The dataset is useful for training and evaluating machine learning models that classify sentiments and identify key phrases that exemplify the provided sentiment. It is particularly useful for identifying and filtering hateful or negative content on Twitter.

#### 4.1.2.2 Dataset Format

The dataset consists of four fields, each representing specific attributes of a tweet:

- **textID**: Unique identifier for each tweet.
- **text**: The actual content of the tweet, representing the user's post on Twitter.
- **selected_text**: A word or phrase extracted from the tweet that encapsulates the sentiment.
- **sentiment**: Sentiment label of the tweet (e.g., `positive`, `neutral`, `negative`).

#### 4.1.2.3 Dataset Notes

- **Preprocessing Considerations**: When parsing the CSV file, ensure that beginning and ending quotes from the text field are removed to avoid incorrect tokenization.
- **Size and Scope**: The dataset contains **27.5k tweets**, making it suitable for training NLP-based sentiment classifiers.
- **Objective**: The goal is to develop a machine learning model that can accurately predict sentiment and extract the key text that represents it.
- **Classification Models**: Various classification algorithms can be applied and compared based on evaluation metrics to determine the best approach for sentiment classification.
- **License and Updates**: The dataset is under **CC0: Public Domain** and is expected to be updated annually.

The dataset can be accessed here: https://www.kaggle.com/c/tweet-sentiment-extraction/data?select=train.csv

### 4.1.3 twitter-sentiments-dataset

The **Twitter Sentiments Dataset** is a dataset designed for sentiment analysis, containing labeled tweets categorized into three sentiments: negative (-1), neutral (0), and positive (+1). It provides essential data for training models that classify sentiments in social media text.

#### 4.1.3.1 Dataset Description

This dataset contains two fields: the cleaned tweet text and its corresponding sentiment label. The dataset is widely used for text classification and sentiment analysis in NLP applications.

#### 4.1.3.2 Dataset Format

The dataset consists of the following fields:

- **clean_text**: Processed tweet text without unnecessary characters or formatting.
- **category**: Sentiment category of the tweet (`-1 = negative`, `0 = neutral`, `+1 = positive`).

#### 4.1.3.3 Dataset Notes

- **Acknowledgements**: The dataset was provided by **Hussein, Sherif (2021)**, titled *"Twitter Sentiments Dataset"*, available on Mendeley Data (DOI: 10.17632/z9zw7nt5h2.1).
- **Size and Scope**: The dataset is **20.9 MB** and contains a significant number of labeled tweets, making it ideal for sentiment analysis research.
- **Usability Score**: Rated **10.00** in usability, ensuring it is well-structured for machine learning applications.
- **License and Updates**: This dataset is released under the **Attribution 4.0 International (CC BY 4.0)** license and has no expected updates.

The dataset can be accessed here: https://www.mendeley.com/datasets/z9zw7nt5h2.1

## 4.2 Data Preprocessing

In this project, we aimed to thoroughly analyze the sentiment of textual data to gain a deeper understanding of our customers. To achieve this, we utilized three distinct datasets, each containing relevant customer feedback labeled with sentiment information. The data preprocessing steps were crucial in preparing the input for training machine learning models. Our target is to use all the models from the syllabus to accurately determine sentiment scores and uncover valuable insights into customer preferences and needs.

### 4.2.1 Data Collection Process

Using the datasets that we have described in the last sections, we conducted several cleaning methods for preprocessing the text to make them cleaner to some extent.

The datasets were sourced from Kaggle and loaded into separate pandas DataFrames.The data collection process involved downloading the datasets and loading them into our Python environment:

Listing 4.1: Loading Datasets

```
import pandas as pd

# Load a CSV file and initialize the Dataset class
file_path = "../../data/raw/kazanova_sentiment140_training.1600000.
    processed.noemoticon_with_headers.csv"
df = pd.read_csv(file_path, encoding='latin1')
dataset = Dataset(df)
file_path2 = "../../data/raw/yasserh_twitter-tweets-sentiment-
    dataset_Tweets_with_headers.csv"
df2 = pd.read_csv(file_path2, encoding='latin1')
dataset2 = Dataset(df2)
file_path3 = "../../data/raw/saurabhshahane_twitter-sentiment-
    dataset_Twitter_Data_with_headers.csv"
df3 = pd.read_csv(file_path3, encoding='latin1')
dataset3 = Dataset(df3)

# Display basic information about one of the datasets
dataset.show_overview()
```

The loaded datasets contained tweet text and sentiment labels, which were standardized before merging.

### 4.2.2 Data Preprocessing and Merging

Since our project combined multiple datasets, we devised a strategy to merge them into a single cohesive dataset for analysis. The datasets had differing schemas (column names and label formats), so the first step was to standardize column names and label values across

all DataFrames. We extracted the relevant columns from each DataFrame – primarily the tweet text and its sentiment label – and dropped any extraneous fields (such as tweet IDs, timestamps, or user names that were not needed for sentiment analysis). For instance, one dataset's sentiment label was a numeric value (e.g., 0 = negative, 4 = positive), another used textual labels ("positive", "negative", "neutral"), and a third used -1/0/1 to denote sentiment classes. We mapped all these to a consistent labeling scheme. In our case, we unified the sentiment labels to -1, 0, 1 representing negative, neutral, and positive sentiments respectively. For example, a tweet with label 4 (positive in the first dataset) was mapped to 1, and "negative" was mapped to -1. After aligning the schema, we merged the datasets by concatenating them vertically (appending rows) since each dataset contained unique samples. We used `pandas.concat` to combine DataFrames once their columns were made consistent. The code snippet below demonstrates how we merged DataFrames:

```
from preprocess import handle_missing_values, drop_duplicates

# Standardize column names
df1.rename(columns={"target": "sentiment", "text": "text"}, inplace=True)
df2.rename(columns={"category": "sentiment", "clean_text": "text"},
    inplace=True)
df3.rename(columns={"Sentiment": "sentiment", "Tweet": "text"}, inplace=
    True)

# Map sentiment values to a common scheme
df1["sentiment"].replace({4: 1, 0: -1}, inplace=True)
df2["sentiment"].replace({-1: -1, 0: 0, 1: 1}, inplace=True)
df3["sentiment"].replace({"Positive": 1, "Neutral": 0, "Negative": -1},
    inplace=True)

# Merge datasets
combined_df = pd.concat([df1[["text", "sentiment"]], df2[["text", "
    sentiment"]], df3[["text", "sentiment"]]], ignore_index=True)

# Handle missing values and remove duplicates
combined_df = handle_missing_values(combined_df, strategy="mode")
combined_df = drop_duplicates(combined_df)
```

### 4.2.3 Data Cleaning and Preparation

After merging the datasets, we applied a series of data cleaning steps to prepare the text for analysis. We imported necessary libraries and tools for text preprocessing, including Python's `re` module for regular expressions, NLTK for tokenization and stopword lists, and custom preprocessing functions defined in our codebase. The cleaning process aimed to remove noise and standardize the text, enabling machine learning models to focus on the meaningful content of tweets. The main steps in our text cleaning pipeline were as follows:

- **Removing Special Characters and Punctuation:** We filtered out all non-alphanumeric characters, such as punctuation marks and symbols (e.g., "!!??" or "..."), which do

not carry useful sentiment information. Numerical digits (e.g., phone numbers, dates) were also removed, as they are typically uninformative for general sentiment analysis.

- **Removing URLs and HTML Tags:** Tweets often contain URLs (e.g., "http://" or "https://") or HTML markup from scraped content. Using regular expressions, we stripped substrings starting with "http://", "https://", or "www", as well as HTML tags (text within < > brackets). This ensures that only natural language content remains for sentiment analysis.

- **Removing Mentions and Hashtags:** We eliminated Twitter-specific artifacts like user mentions (e.g., "@username") and hashtags (e.g., "#Topic"). Mentions were removed using the regex `@\w+`, while hashtags were handled by removing the non-alphanumeric "#" symbol. This prevents the model from treating usernames or trending tags as features, as they are not generalizable signals of sentiment.

- **Chat Slang Expansion:** Social media text often includes slang or abbreviations (e.g., "LOL" for "laugh out loud", "BRB" for "be right back"). We implemented a dictionary of common chat abbreviations, replacing them with their full meanings (e.g., "OMG" becomes "oh my god"). This normalization aids sentiment analysis by converting informal terms into standard language, often preserving sentiment context (e.g., "LOL" may indicate humor or positivity).

- **Lowercasing Text:** All text was converted to lowercase to normalize words like "Happy" and "happy", reducing redundant distinctions due to capitalization. This is a standard preprocessing step to eliminate case sensitivity issues in text analysis .

- **Tokenization:** Each cleaned tweet was split into individual tokens (words) using NLTK's word tokenizer. For example, "I love this movie!" becomes `["i", "love", "this", "movie"]`. Tokenization is essential for subsequent steps like stopword removal and feature extraction.

- **Stopword Removal:** Common English stopwords (e.g., "the", "is", "on") were removed using NLTK's built-in stopword list. These frequent words carry little sentiment value, and their removal reduces noise and data size, focusing the analysis on meaningful terms .

- **Stemming/Lemmatization (Optional):** Our preprocessing function included options for stemming (e.g., "happiest" to "happi") and lemmatization (e.g., "running" to "run"). We primarily used lemmatization with NLTK's WordNet lemmatizer to normalize words to their dictionary form (e.g., "better" to "good"), reducing inflection variance. This step was configurable, and we analyzed results with and without it to assess its impact.

After these steps, raw tweets were transformed into clean, standardized token sequences. For example, a tweet like:

```
@User OMG I love this movie!!!  Check out https://t.co/xyz #awesome
```

becomes:

```
["oh", "my", "god", "love", "movie", "awesome"]
```

This pipeline was applied to every tweet in the merged dataset using vectorized operations in `pandas` and `tqdm` for efficiency. The result was a new DataFrame column with cleaned text (as strings or token lists), ready for feature extraction.

### 4.2.4 Exploratory Data Analysis (EDA)

To better understand the dataset, we performed Exploratory Data Analysis (EDA) on key features. Below are some visualizations that provide insights into the data distribution.

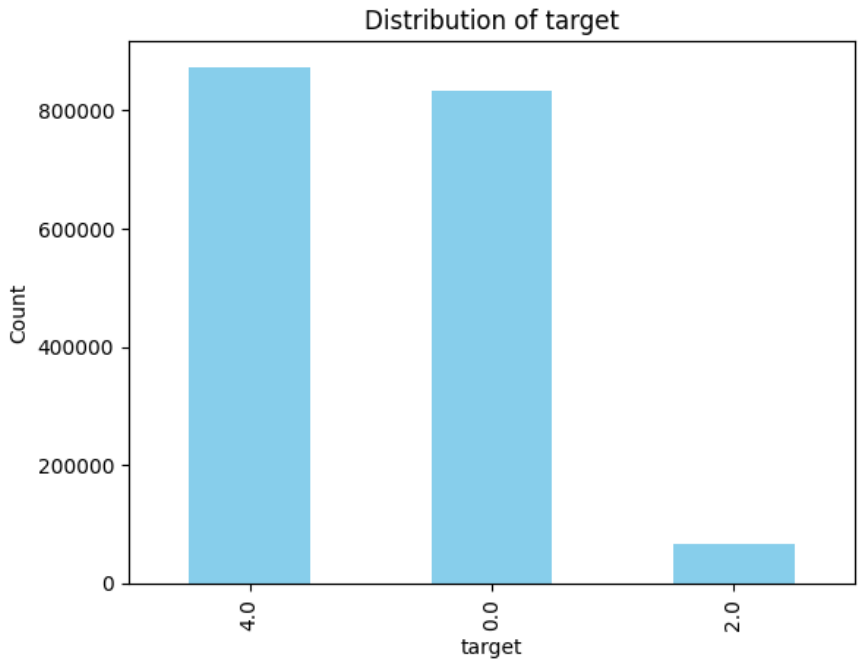



Figure 4.1: Distribution of Target Variable

Figure 4.1 shows the distribution of the target variable in the dataset. The dataset is imbalanced, with a significantly higher number of samples labeled as 4.0 (positive sentiment) and 0.0 (negative sentiment) compared to 2.0 (neutral sentiment). Due to this imbalance, our team decided to ignore the neutral target (2.0) and focus on training and evaluating models for positive and negative sentiment only.

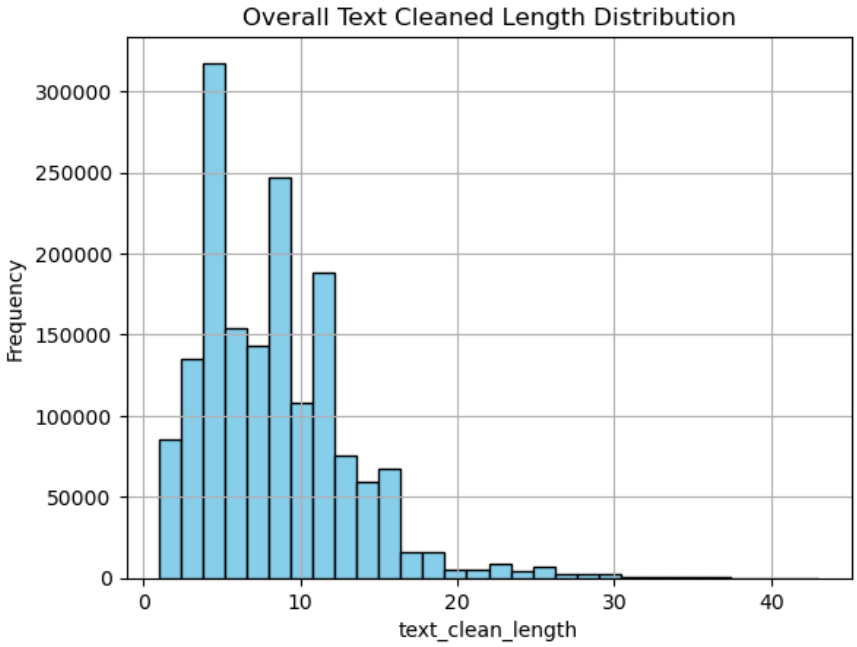



Figure 4.2: Overall Text Cleaned Length Distribution

Figure 4.2 illustrates the distribution of cleaned text lengths across all tweets. Most tweets have a cleaned length between 5 and 15 tokens, with a long tail for shorter or longer tweets.

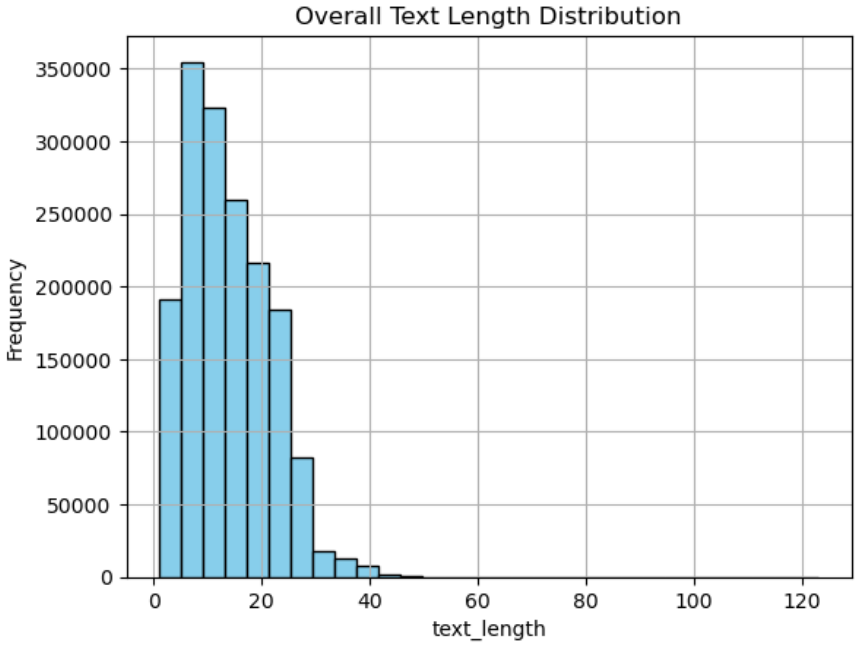



Figure 4.3: Distribution of Raw Text Lengths

Figure 4.3 depicts the distribution of raw text lengths before cleaning. This visualization

highlights the variability in tweet lengths prior to preprocessing.

These visualizations helped us identify key patterns in the data, such as class imbalance and typical text lengths, which informed subsequent preprocessing and modeling decisions.

### 4.2.5 Publishing the Merged Dataset on Kaggle

We published the cleaned, merged dataset on Kaggle to enable further analysis. The process involved:

1. **Saving the Dataset:** The preprocessed data was saved as `merged_cleaned_tweets.csv`, including cleaned text, sentiment labels, and engineered features.
2. **Creating a New Kaggle Dataset:** On Kaggle, we created a new dataset with a descriptive title and an open license aligned with the original datasets' terms.
3. **Uploading the Data:** The CSV was uploaded via Kaggle's interface or API, with column integrity verified in the preview.
4. **Publishing:** After adding metadata (tags, visibility), the dataset was published and shared with our team for use in Kaggle Notebooks or offline analysis.

This step ensured reproducibility and contributed a ready-to-use sentiment analysis dataset to the community. The dataset can be accessed here : https://www.kaggle.com/datasets/zphudzz/tweets-clean-posneg-v1

### 4.2.6 Final Remarks

The preprocessing stage laid a critical foundation for our sentiment analysis project. By merging multiple Kaggle datasets, cleaning noise (e.g., URLs, tags), and normalizing text, we enhanced data quality. Then later on, converting text to embedding vectors and encoding additional features enabled robust model training. Models trained on this preprocessed data outperformed those on raw data, highlighting the importance of these steps for accurate sentiment predictions.

## 4.3 Visualization

**Small note**: We only take a subset of our dataset to visualize these important things.

### 4.3.1 Relationships of the Attributes

Complementing this, the **Pairwise Relationships** Pairplot highlights numerical features such as 'target', 'text_length', and 'text_clean_length'. Histograms show that tweets are generally short, with 'text_length' peaking at 0–100 characters and 'text_clean_length' at 0–40 characters, reflecting the impact of cleaning. Scatter plots reveal no strong correlation between text length and sentiment, while a linear relationship between 'text_length' and 'text_clean_length' confirms the effectiveness of cleaning. These visualizations offer insights into the dataset's structure for further modeling.

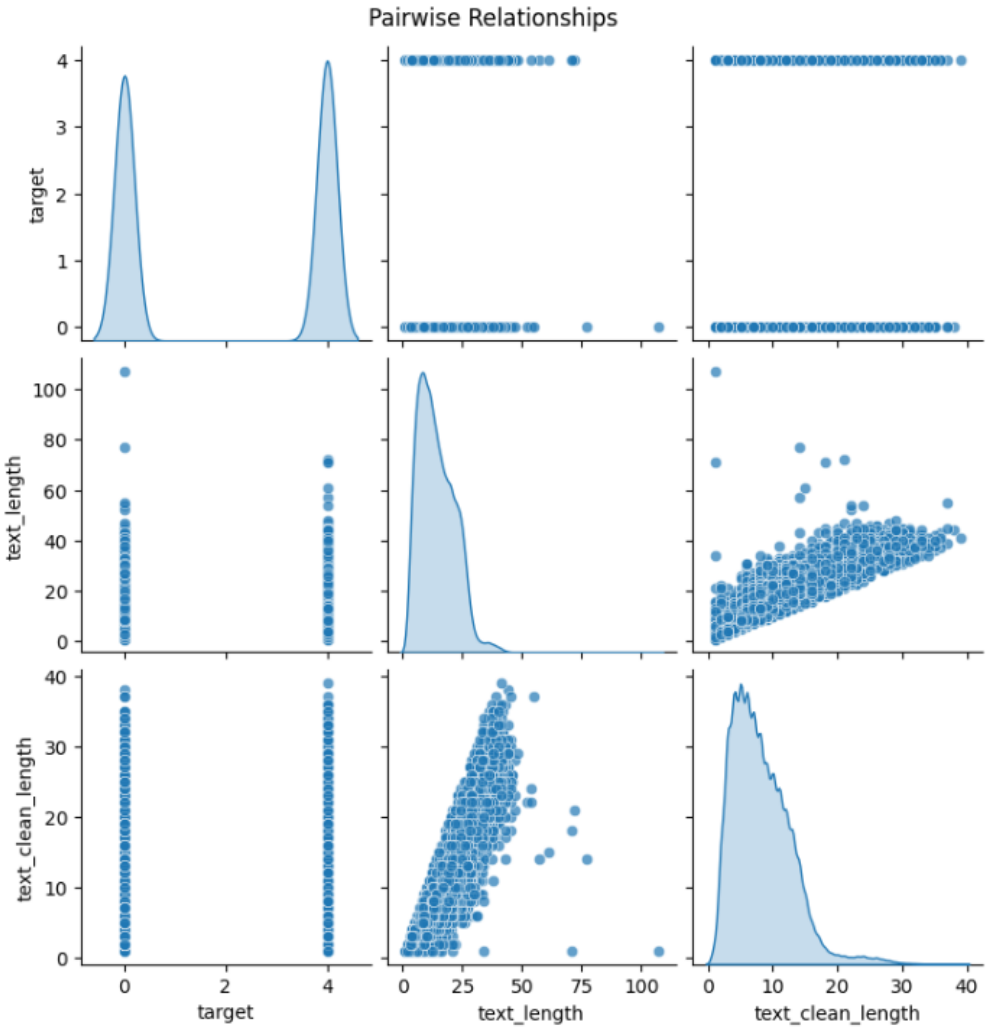



Figure 4.4: Pairwise Relationships

### 4.3.2 Top Words Visualizations

The visualizations highlight the top 20 most frequent words in the 'text_clean' column through a bar chart and a word cloud. The chart shows 'i' (14,000 occurrences), 'm' (12,000), and 'modi' (10,000) as the most common words, alongside verbs like 'get,' 'like,' and 'go,' and positive terms such as 'good,' 'love,' and 'great.' The word cloud emphasizes high-frequency words like 'i,' 'm,' and 'modi' with larger fonts, while less frequent words like 'great' and 'lol' appear smaller. Together, these visualizations reveal the dataset's informal Twitter tone and frequent sentiment-related terms, offering insights into its linguistic patterns.

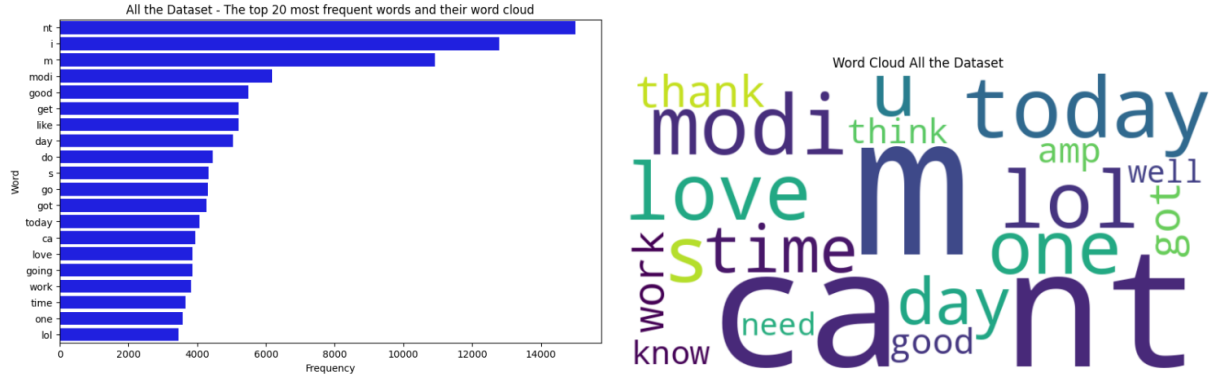



Figure 4.5: Top 20 of entire dataset

The positive class visualization (target = 4.0) uses a bar chart and word cloud to display the top 20 most frequent words. The bar chart highlights 'i' (5,000 occurrences), 'm' (4,500), and 'modi' (4,000) as the most frequent, followed by positive words like 'good' (3,500), 'love' (3,000), 'thanks' (1,800), and casual terms like 'lol' and 'haha.' The word cloud reinforces this, with larger fonts for frequent words like 'i,' 'm,' and 'good,' and smaller fonts for less common ones like 'haha.' These visualizations emphasize optimism, gratitude, and humor in positive tweets.

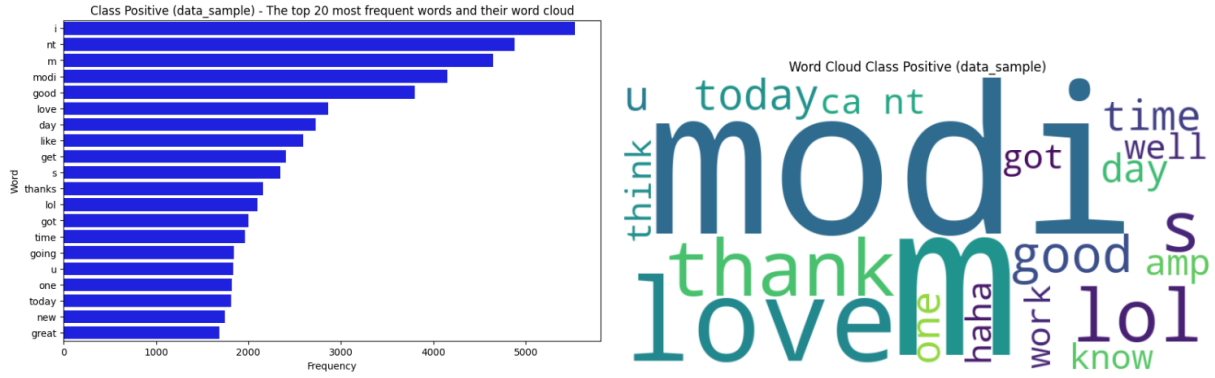



Figure 4.6: Top 20 of class Positive

The negative class visualization (target = 0.0) uses a bar chart and word cloud to display the top 20 most frequent words. The bar chart shows 'i' (10,000 occurrences), 'm' (9,000), and

'don't' (8,000) as the most frequent, followed by words like 'get' (7,000), 'go' (6,000), and negative terms such as 'really,' 'want,' 'still,' and 'miss.' The word cloud reinforces this with larger fonts for frequent words like 'i,' 'm,' and 'don't,' and smaller fonts for less common ones like 'miss.' These visualizations highlight frustration, restriction, and dissatisfaction in negative tweets.

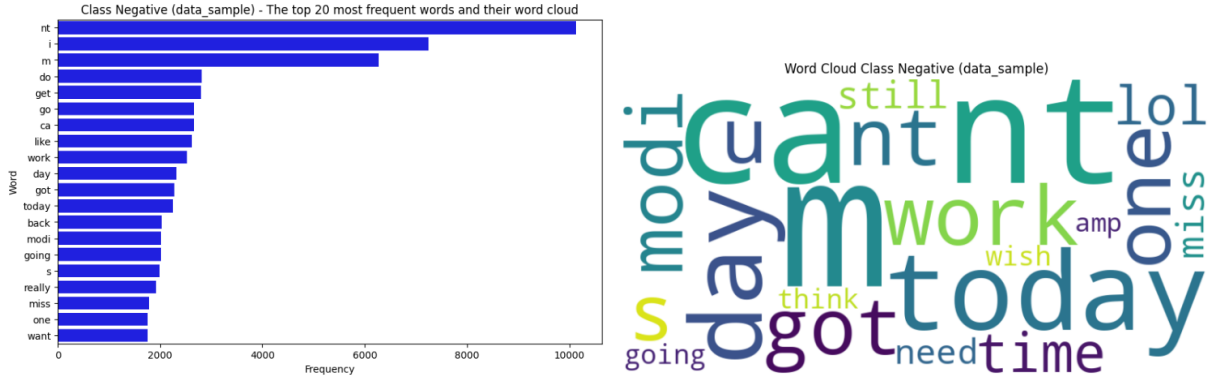
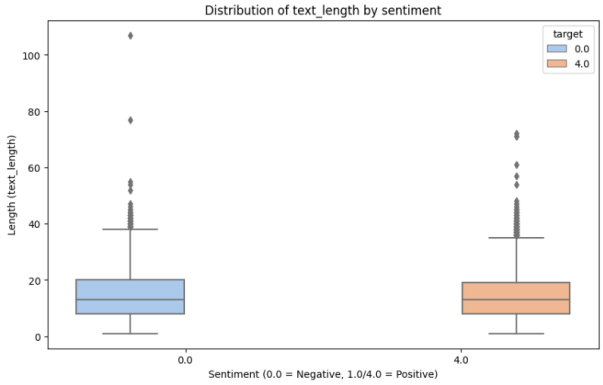


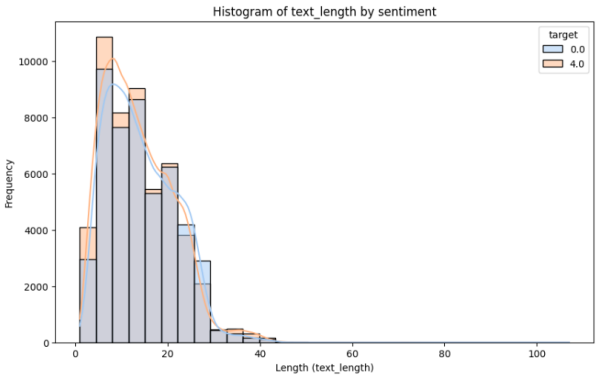

Figure 4.7: Top 20 of class Negative

### 4.3.3 The distribution of Text Length

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.

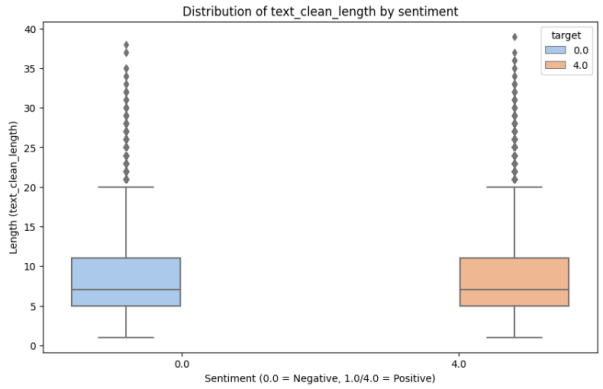



(a) Distribution of text_length by sentiment

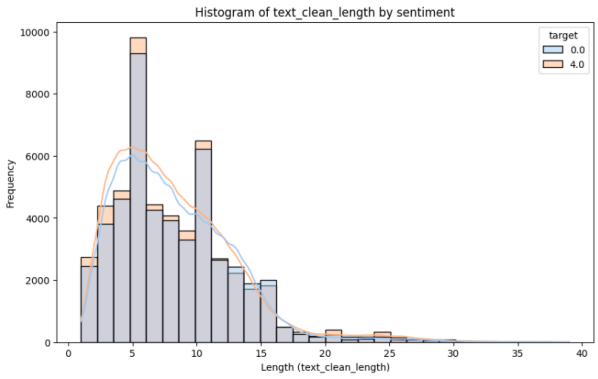



(b) Histogram of text_length by sentiment

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

### 4.3.4 Word Frequencies by Labels

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.

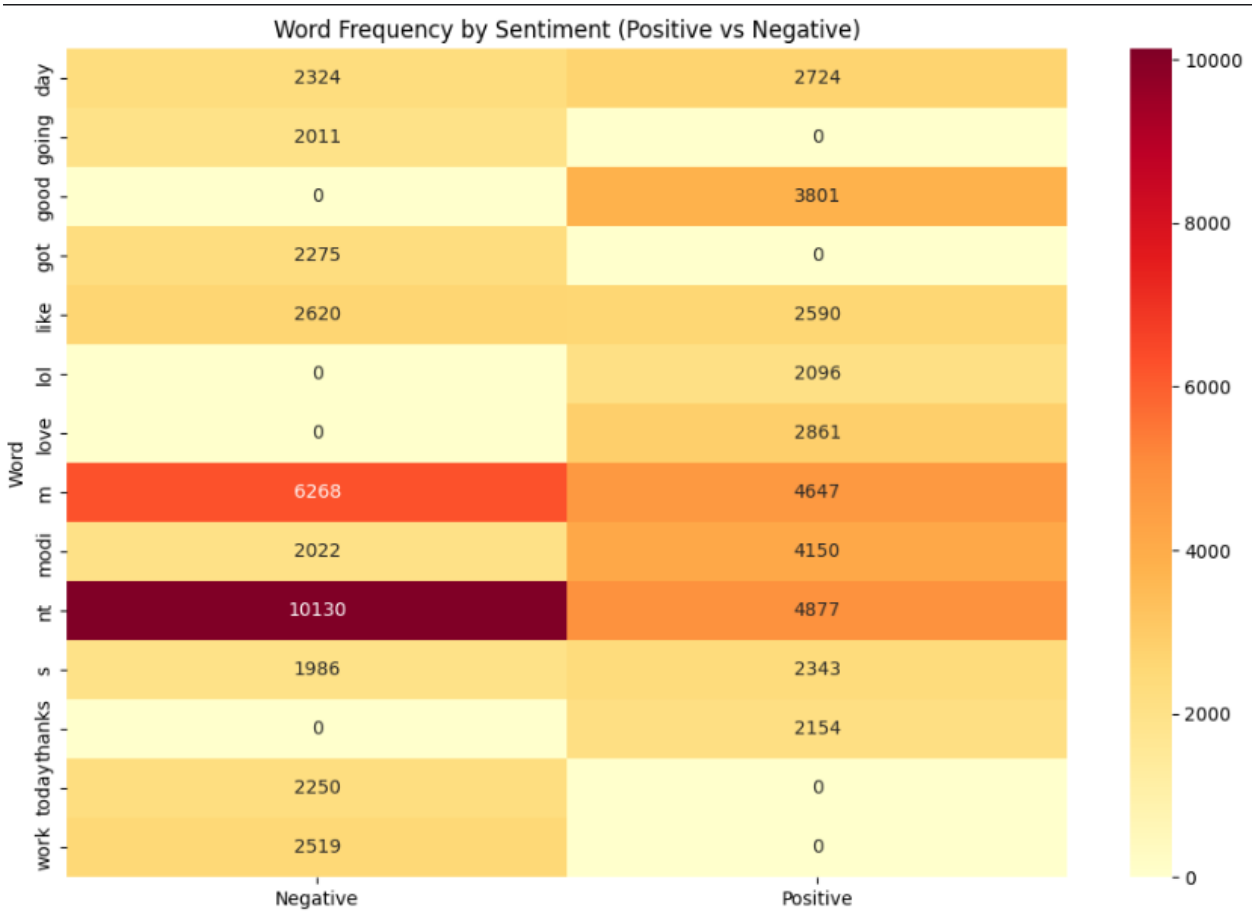



Figure 4.10: Word Frequency by Sentiment

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.

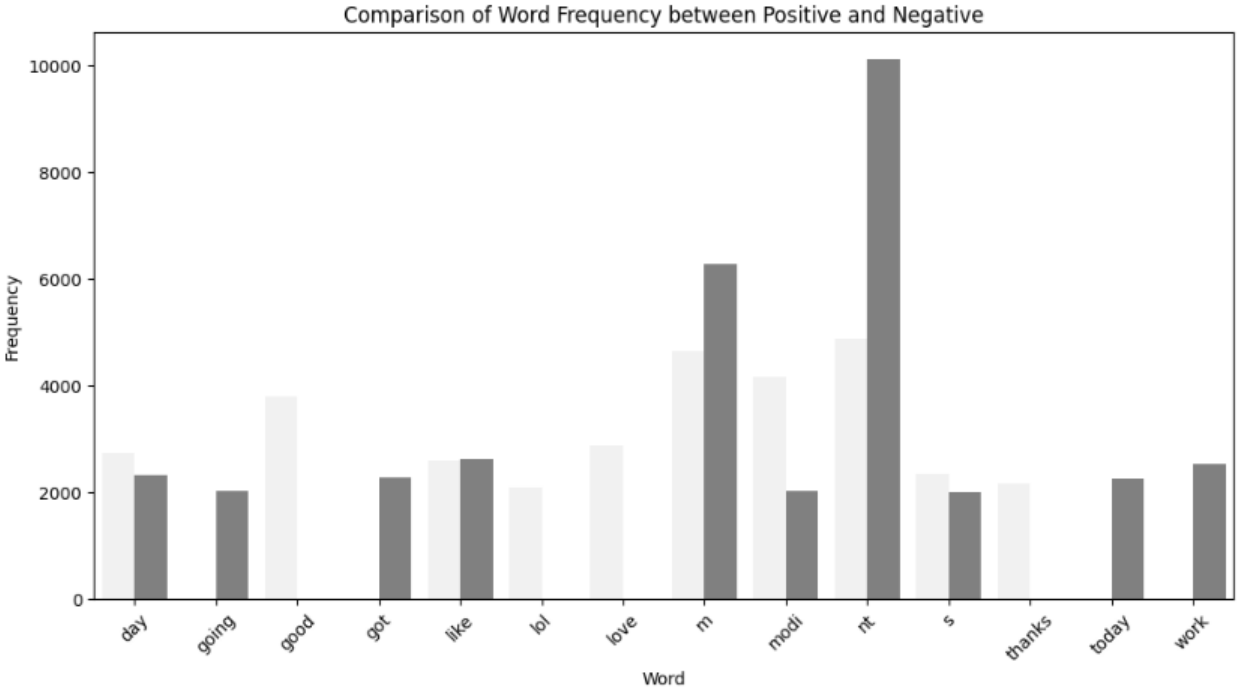



Figure 4.11: Comparision of Word Frequency

## 4.4 Analyze the Training Process of Models

Before training machine learning models on a dataset and making predictions on a test set, a crucial step involves transforming the raw text data into a numerical representation that the models can effectively learn from. Specifically, we need to convert text into vectors of numbers. This section details the approach used for feature building.

### 4.4.1 Building Features

#### 4.4.1.1 Overview

In this project, a dedicated class called `FeatureBuilder` has been designed to handle the feature extraction and transformation process. This class encapsulates various methods for converting text data into numerical features suitable for machine learning models.

**The `FeatureBuilder` Class**

The `FeatureBuilder` class provides functionalities for different feature extraction methods, dimensionality reduction techniques, and model persistence. The code for the class is as follows:

```
class FeatureBuilder:
    def __init__(self, method="tfidf", save_dir="data/processed",
        reduce_dim=None, n_components=100):
        self.method = method
        self.save_dir = save_dir
        self.reduce_dim = reduce_dim
        self.n_components = n_components
        os.makedirs(save_dir, exist_ok=True)

        if method == "tfidf":
            self.vectorizer = TfidfVectorizer(max_features=2000,
                stop_words="english")
        elif method == "count":
            self.vectorizer = CountVectorizer(max_features=2000)
        elif method == "binary_count":
            self.vectorizer = CountVectorizer(binary=True, max_features
                =2000)
        elif method == "word2vec":
            self.word2vec_model = api.load("word2vec-google-news-300")
        elif method == "glove":
            self.glove_model = api.load("glove-wiki-gigaword-100")
        elif method == "bert":
            self.tokenizer = AutoTokenizer.from_pretrained("sentence-
                transformers/all-MiniLM-L6-v2")
            self.bert_model = AutoModel.from_pretrained("sentence-
                transformers/all-MiniLM-L6-v2")

        self.reducer = None
        if self.reduce_dim == "pca":
```

```
            self.reducer = PCA(n_components=self.n_components)
        elif self.reduce_dim == "lda":
            self.reducer = LDA(n_components=self.n_components)

    def fit(self, texts, labels=None):
        if self.method in ["tfidf", "count", "binary_count"]:
            self.vectorizer.fit(texts)
            if self.reduce_dim == "lda":
                assert labels is not None, "LDA requires class labels (y)."
                features = self.vectorizer.transform(texts).toarray()
                self.reducer.fit(features, labels)
            elif self.reduce_dim == "pca":
                features = self.vectorizer.transform(texts).toarray()
                self.reducer.fit(features)
        elif self.method in ["word2vec", "glove", "bert"]:
            if self.reduce_dim == "lda":
                raise ValueError(f"LDA is not supported for method {self.method}")

    def transform(self, texts, labels=None):
        if self.method in ["tfidf", "count", "binary_count"]:
            features = self.vectorizer.transform(texts).toarray()
            return self._apply_reducer(features, labels)

        elif self.method == "word2vec":
            word2vec_embeddings = []
            for doc in tqdm(texts, desc="Processing Word2Vec", unit="document"):
                word2vec_embeddings.append(self._get_word2vec_vector(doc))
            features = np.array(word2vec_embeddings)
            return features

        elif self.method == "glove":
            glove_embeddings = []
            for doc in tqdm(texts, desc="Processing GloVe", unit="document"):
                glove_embeddings.append(self._get_glove_vector(doc))
            features = np.array(glove_embeddings)
            return features

        elif self.method == "bert":
            bert_embeddings = []
            for doc in tqdm(texts, desc="Processing BERT", unit="document"):
                bert_embeddings.append(self._get_bert_embedding(doc))
            features = np.array(bert_embeddings)
            return features

    def fit_transform(self, texts):
        self.fit(texts)  # First fit the model (compute parameters)
        return self.transform(texts)
```

#### 4.4.1.2 Key Components and Functionalities

The `FeatureBuilder` class incorporates the following key components:

- **Feature Extraction Methods:** Implements various feature extraction methods such as TF-IDF, Count Vectorization, Word2Vec, GloVe, and BERT embeddings.
- **Dimensionality Reduction:** Supports dimensionality reduction techniques like PCA and LDA to reduce the complexity of the feature space and improve model performance.
- **Model Persistence:** Provides functionalities to save and load fitted vectorizers, models, and dimensionality reduction objects for later use.

#### 4.4.1.3 Usage

The `FeatureBuilder` class is initialized with a specified feature engineering method, save directory, dimensionality reduction method, and the number of components for dimensionality reduction. It then uses this configuration to fit and transform the text data into numerical feature matrices, which can be used as inputs for training machine learning models.

### 4.4.2 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:

- **MODELS** – Description
- **MODELS** – Description
- **MODELS** – Description
- **MODELS** – Description
- **MODELS** – Description
- **MODELS** – Description
- **MODELS** – Description
- **MODELS** – Description

### 4.4.3 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.

#### 4.4.3.1 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.

#### 4.4.3.2 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.

#### 4.4.3.3 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.

#### 4.4.3.4 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.

#### 4.4.3.5 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.

#### 4.4.3.6 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.

# Chapter 5

# Self-Reflection

## 5.1 Future Developments

Building upon the foundation established in this assignment, our future work will focus on advancing our sentiment analysis system through more sophisticated machine learning techniques, deep learning or modern approaches The key areas of improvement will include:

- **Ensemble Methods:** Develop robust sentiment classifiers using bagging and boosting techniques, incorporating voting and model combination strategies to improve predictive performance.
- **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.