TWITTER SENTIMENT ANALYSIS
A MINI PROJECT REPORT

Submitted for the partial fulfillment for the award of the degree of

BACHELOR OF COMPUTER SCIENCE WITH DATA SCIENCE

OF

UNIVERSITY OF MADRAS

BY DHEEKSHITHA K

REGISTER NO: 222206971

Under the guidance of

Dr. P.Pakutharivu, M.Sc., M.Phil.,Ph.D.

DEPARTMENT OF COMPUTER SCIENCE WITH DATA SCIENCE

ANNA ADARSH COLLEGE FOR WOMEN (AUTONOMOUS)

ANNA NAGAR, CHENNAI-600 040.

APRIL 2025

CERTIFICATE

This is to certify the report entitled

TWITTER SENTIMENT ANALYSIS

Being submitted to the University of Madras

BY DHEEKSHITHA K

REGISTER NO: 222206971

for the partial fulfillment for the

Award of the degree of

BACHELOR OF COMPUTER SCIENCE WITH DATA SCIENCE

Is a Bonafide record of a work carried out by her, under my guidance and supervision

HEAD OF THE DEPARTMENT **GUIDE**

Submitted for the practical examination held in April 2025

at Anna Adarsh College for Women (Autonomous)

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DATE:

ACKNOWLEDGEMENT

I would like to take this opportunity to express my sincere gratitude to everyone who supported me throughout this mini-project. Their guidance, invaluable constructive criticism, and encouragement have been instrumental in the successful completion of this work.

I extend my heartfelt thanks to our **Principal, Dr. R. Shanthi, M.Com., M.Phil., Ph.D.,** Anna Adarsh College for Women (Autonomous) , Chennai, for providing me with the opportunity to undertake this project.

I am profoundly grateful to **Dr. A. Lakshmi, M.C.A., M.Phil., SET, Ph.D.,** Associate Professor & Head, Department of Computer Science with Data Science, Anna Adarsh College for Women (Autonomous), for her unwavering support and valuable guidance throughout the project.

I also extend my sincere appreciation to **Dr. P. Pakutharivu, M.Sc., M.Phil., Ph.D.,** Assistant Professor, Post Graduate Department of Computer Science, Anna Adarsh College for Women (Autonomous), for her continuous encouragement, insightful suggestions, and technical expertise, which greatly contributed to the successful completion of this project. Her valuable inputs and constant motivation have been pivotal in shaping this work.

Finally, I would like to express my deepest gratitude to **God,** whose blessings have guided me through this journey. I am also profoundly thankful to my **parents** for their unwavering support, encouragement, and for giving me the opportunity to pursue my undergraduate degree in Computer Science with Data Science.

ABSTRACT

ABSTRACT

Sentiment analysis helps classify textual data into different sentiment categories, providing valuable insights into public opinion. With the rise of social media, platforms like Twitter serve as real-time sources for sentiment analysis. This project applies machine learning techniques to analyze Twitter sentiments, comparing **Logistic Regression (LR), Bernoulli Naïve Bayes (BNB), and Support Vector Machine (SVM)** models.

The study follows a structured approach, including **data preprocessing, feature extraction using Term Frequency-Inverse Document Frequency (TF-IDF), and model evaluation** based on accuracy, precision, recall, F1-score, and execution time. The **Sentiment140 dataset** is used, containing tweets labelled as positive, negative, or neutral.

A major challenge in sentiment analysis is handling noisy, unstructured text. To improve dataset quality, **preprocessing steps** such as stopword removal, tokenization, lemmatization, and punctuation removal are applied. The effectiveness of each model is assessed based on classification performance and computational efficiency.

This project aims to identify the most suitable sentiment classification model, providing insights into machine learning applications for sentiment-driven decision-making, benefiting businesses, policymakers, and researchers.

TABLE OF CONTENTS

INTRODUCTION

INTRODUCTION

1.1 Project Overview

Introduction to Machine Learning and Sentiment Analysis

Machine Learning (ML) has significantly impacted various domains by enabling computers to analyze large datasets and make decisions without explicit human intervention. ML models learn patterns from data and generalize them to make predictions, automating complex tasks that were previously performed manually. One of the most important applications of ML is **Natural Language Processing (NLP),** which focuses on the interaction between computers and human language. NLP techniques allow machines to read, understand, and process textual data, making them useful for various tasks such as language translation, text summarization, speech recognition, and sentiment analysis.

Sentiment analysis, also known as **opinion mining,** is a subfield of NLP that deals with identifying and categorizing opinions expressed in text. It helps determine whether a given piece of text conveys a **positive, negative, or neutral** sentiment. Sentiment analysis has gained immense importance with the rise of social media and online reviews, where people express their opinions on different topics, products, services, and events. Organizations and businesses use sentiment analysis to monitor customer feedback, track brand reputation, and make data-driven decisions.

The Role of Twitter in Sentiment Analysis

Twitter is one of the largest social media platforms, with millions of tweets posted daily on diverse topics, including politics, entertainment, business, and technology. Unlike traditional sources of public opinion, such as surveys or news

articles, Twitter provides **real-time, large-scale, and diverse** opinions directly from users. This makes it a valuable resource for sentiment analysis, allowing businesses, researchers, and policymakers to understand public sentiment on a wide range of topics.

However, analyzing sentiment from Twitter data presents several challenges.

Challenges in Twitter Sentiment Analysis

1. **Short Text Format** – Tweets are limited to 280 characters, often making it difficult to understand the full context of an opinion. Unlike long-form content, where sentiment is more explicitly expressed, tweets require advanced NLP techniques to infer meaning from limited text.
2. **Informal and Noisy Language** – Twitter users frequently use slang, abbreviations, emojis, hashtags, and non-standard grammar, making it challenging for traditional NLP models to interpret the text accurately.
3. **High Volume and Velocity** – Twitter generates a massive number of tweets every second. Processing such a large volume of data efficiently requires robust data handling and computational techniques.
4. **Sentiment Ambiguity** – Many words and phrases can have different meanings depending on the context. Additionally, sarcasm, irony, and humour make sentiment classification difficult. A tweet like "Oh great, another power outage! Just what I needed!" expresses frustration but contains the word "great," which might be misclassified as positive.
5. **Data Imbalance** – Some topics may have more positive sentiments, while others may be predominantly negative. This imbalance can lead to biased sentiment classification models.

To tackle these challenges, machine learning models are used to automate sentiment classification, providing a scalable and efficient way to analyze vast amounts of Twitter data.

Need for Machine Learning in Sentiment Analysis

Traditional sentiment analysis methods relied on lexicon-based approaches, where predefined word lists (sentiment lexicons) were used to determine the sentiment of a text. However, these approaches have significant limitations.

- **Context Insensitivity** – Lexicon-based methods cannot distinguish between different meanings of the same word (for example, the word "hot" can refer to temperature, popularity, or attractiveness).
- **Difficulty Handling Sarcasm and Slang** – Many words carry sentiment differently based on context, and lexicon-based methods struggle with sarcasm and informal text.
- **Static Nature** – Sentiment lexicons require constant updates to keep up with evolving language trends, making them less adaptable.

Machine Learning addresses these challenges by learning directly from labelled datasets. Instead of relying on predefined word lists, ML models:

- Identify patterns in sentiment expression, improving accuracy.
- Adapt to new language trends without needing manual updates.
- Handle contextual variations more effectively than rule-based methods.

Objectives of the Project

The primary focus of this project is to implement machine learning techniques for sentiment analysis on Twitter data. The key objectives include:

- **Data Preprocessing** – Cleaning raw Twitter data by removing stopwords, punctuation, URLs, special characters, and non-useful text elements to enhance model performance.
- **Feature Extraction Using TF-IDF** – Converting text into numerical format using Term Frequency-Inverse Document Frequency (TF-IDF) to make it suitable for machine learning models.
- **Comparing Three Classification Models** – Implementing and evaluating:
    - **Logistic Regression (LR)** – A linear model used for binary and multi-class classification.
    - **Bernoulli Naïve Bayes (BNB)** – A probabilistic classifier based on Bayes' theorem, commonly used for text classification.
    - **Support Vector Machine (SVM)** – A powerful supervised learning model that finds the best separation boundary between classes.
- **Evaluating Model Performance** – Assessing the models based on:
    - **Accuracy** – Measures the percentage of correctly classified tweets.
    - **Precision, Recall, and F1-score** – Evaluates how well the model distinguishes between positive and negative sentiments.
    - **ROC-AUC Score** – Measures how effectively the model differentiates between different sentiment classes.
    - **Execution Time** – Compares how efficiently each model processes and classifies tweets.

- **Determining the Best Classifier** – Identifying which of the three models performs best for sentiment analysis on Twitter data, considering both accuracy and computational efficiency.

By achieving these objectives, the project aims to contribute to the field of sentiment analysis by determining the most effective model for analyzing sentiments on social media.

1.2 Literature Review

Introduction to Sentiment Analysis

Sentiment analysis, or **opinion mining**, is a technique in **Natural Language Processing (NLP)** used to determine the sentiment expressed in textual data. It has applications in various fields, including **business intelligence, social media monitoring, political analysis, and customer feedback analysis**. With the increasing use of **social media platforms like Twitter**, sentiment analysis has become an essential tool for understanding public opinion in real time.

Traditional sentiment analysis techniques relied on **lexicon-based approaches**, where predefined sentiment dictionaries were used to classify text. However, the rise of **machine learning (ML) and deep learning (DL) techniques** has significantly improved the accuracy and scalability of sentiment classification.

This literature review discusses the **evolution of sentiment analysis techniques, feature extraction methods, and classification models** while identifying key research gaps that justify the approach taken in this project.

Evolution of Sentiment Analysis Techniques

Lexicon-Based Approaches

Early sentiment analysis methods primarily used **lexicon-based techniques**, where each word in a text was assigned a sentiment score based on predefined dictionaries. Some of the most widely used sentiment lexicons include:

- **SentiWordNet** – A lexical database that assigns positive, negative, and neutral scores to words.
- **AFINN** – A wordlist with manually assigned sentiment scores for short texts.

- **VADER (Valence Aware Dictionary and Sentiment Reasoner)** – A rule-based model designed specifically for social media sentiment analysis.

Limitations of Lexicon-Based Methods:

- **Contextual misunderstanding:** Words with multiple meanings may be misclassified (e.g., "cool" can mean temperature or approval).
- **Difficulty handling negation and sarcasm:** Sentences like *"I am not happy"* can be incorrectly classified as positive.
- **Static nature:** Lexicons must be manually updated to include new slang, abbreviations, and domain-specific terms.

Due to these limitations, researchers shifted toward **machine learning-based sentiment analysis** for more accurate classification.

Machine Learning-Based Approaches

With advancements in **ML and NLP**, researchers began using **supervised learning models** to classify sentiment based on labelled datasets. Some of the most commonly used ML models for sentiment classification include:

- **Naive Bayes (NB):** A probabilistic classifier based on Bayes' theorem, widely used in text classification.
- **Support Vector Machine (SVM):** A model that finds the optimal hyperplane to separate sentiment classes.
- **Logistic Regression (LR):** A linear model that predicts sentiment probabilities based on input features.

Several studies have shown that **machine learning models outperform lexicon-based approaches** in accuracy and scalability. However, **feature extraction** plays a crucial role in determining the performance of ML-based sentiment classifiers.

Feature Extraction Techniques in Sentiment Analysis

To apply machine learning to textual data, words must be converted into numerical representations. Various **feature extraction techniques** have been explored in sentiment analysis research:

1. Bag of Words (BoW)

- Represents text as a matrix of word occurrences.
- Ignores word order and context, treating all words as independent.
- Works well for **simple classification tasks** but lacks semantic understanding.

2. TF-IDF (Term Frequency-Inverse Document Frequency)

- Weighs words based on their importance in a document relative to their occurrence in the dataset.
- More effective than BoW for **removing the influence of frequently used words** while highlighting key sentiment-heavy words.
- Works well with **traditional ML models like Naïve Bayes, SVM, and Logistic Regression**.

3. Word Embeddings (Word2Vec, GloVe, FastText)

- Captures **semantic relationships** between words by representing them as dense vectors.
- Used in **deep learning models (LSTMs, Transformers)** for context-aware sentiment classification.
- **Not used in this project** since the focus is on **comparing traditional ML models using TF-IDF**.

Several studies have demonstrated that **TF-IDF outperforms BoW in traditional ML-based sentiment analysis**, making it the preferred method for this project.

Comparison of Sentiment Classification Models in Research

Many studies have compared different **machine learning models** for sentiment classification. Key findings include:

- **Pang et al. (2002)** conducted a comparative study using **Naïve Bayes, SVM, and Decision Trees** for sentiment classification. They found that **SVM performed best**, followed by **Naïve Bayes**.
- **Go et al. (2009)** introduced the **Sentiment140 dataset** and trained **Naïve Bayes, Maximum Entropy, and SVM** classifiers. Their results showed that **SVM achieved the highest accuracy** when combined with **TF-IDF features**.
- **Pak & Paroubek (2010)** studied sentiment classification on Twitter using **Naïve Bayes** and found that while it was computationally efficient, **it struggled with highly imbalanced datasets**.
- **Medhat et al. (2014)** reviewed sentiment analysis techniques and concluded that **Logistic Regression and SVM are among the best classifiers** for text-based sentiment classification.

Based on these findings, this project selects **Logistic Regression, Naïve Bayes, and SVM** for sentiment analysis and compares their performance using **TF-IDF feature extraction**.

Research Gaps and Justification for Our Approach

While previous research has made significant progress in **machine learning-based sentiment analysis**, several challenges remain:

- **Handling Sarcasm and Irony:** Most ML models struggle to detect sarcasm, as sentiment is often opposite to the literal meaning.
- **Short Text Classification:** Twitter sentiment analysis is challenging due to the **limited context in 280-character tweets**.
- **Feature Engineering:** While **TF-IDF is widely used**, newer techniques like **word embeddings** are being explored for deep learning models.

Why This Project's Approach?

1. **Compares three widely used ML classifiers (LR, NB, SVM)** to identify the most efficient model for Twitter sentiment analysis.
2. **Uses TF-IDF for feature extraction**, as research has shown it to be highly effective for traditional ML models.
3. **Focuses on the Sentiment140 dataset**, which has been widely used in previous studies, ensuring comparability with past research.

1.3 Model Description

1. Data Collection

The dataset used in this project is the **Sentiment140 dataset**, which is widely used for Twitter sentiment analysis. It contains **1.6 million tweets**, each labelled as **positive (1) or negative (0)**. The dataset is structured as follows:

- **target** – Sentiment label (0 = negative, 1 = positive).
- **ids** – Unique identifier for each tweet.
- **date** – Timestamp indicating when the tweet was posted.
- **flag** – Query information (not used in this project).
- **user** – Twitter username of the person who posted the tweet.
- **text** – The actual tweet content.

For sentiment analysis, only the **text** and **target** columns were used, as they contain the core information required for classification. The dataset was loaded using **pandas** and pre-processed before training the models.

2. Data Preprocessing

Raw Twitter data contains various elements that do not contribute to sentiment classification, such as URLs, special characters, and punctuation. To ensure a clean dataset, the following preprocessing steps were applied:

Text Cleaning

- **Lowercasing:** Converts all text to lowercase to maintain uniformity.
- **Removing URLs and Mentions:** Eliminates hyperlinks (https://...) and user mentions (@username).

- **Removing Punctuation and Special Characters:** Filters out unnecessary symbols that do not carry sentiment-related information.

Tokenization and Stopword Removal

- **Tokenization:** Breaks tweets into individual words for better analysis. This was done using **nltk's word_tokenize()** function.
- **Stopword Removal:** Removes common words like "the," "is," and "and," which do not provide meaningful sentiment context. The **nltk stopwords list** was used for this process.

Lemmatization

- **Lemmatization:** Converts words to their base form (e.g., "running" → "run") to standardize variations. This was implemented using **WordNetLemmatizer()** from the nltk library.

These preprocessing steps helped in cleaning the dataset and ensuring that only relevant textual data was passed to the machine learning models.

3. Feature Selection

Machine learning models cannot process raw text directly, so it was necessary to convert text into a numerical representation. This was achieved using **Term Frequency-Inverse Document Frequency (TF-IDF) vectorization**.

Why TF-IDF?

TF-IDF is a feature extraction method that assigns numerical weights to words based on their importance in a document. It ensures that commonly used words (e.g., "the," "is") receive lower importance, while sentiment-heavy words (e.g., "love," "hate") receive higher importance.

The formula for **TF-IDF** is:

- **Term Frequency (TF):** Measures how often a word appears in a document.
- **Inverse Document Frequency (IDF):** Assigns lower importance to frequently occurring words across multiple documents.
- **TF-IDF Score = TF × IDF**

In this project, the **TfidfVectorizer** from sklearn.feature_extraction.text was used with the following parameters:

- **ngram_range=(1,2):** Considers both single words (unigrams) and word pairs (bigrams) to improve context understanding.
- **max_features=500000:** Limits the vocabulary to the most significant 500,000 words to reduce memory consumption. This transformation allowed machine learning models to process textual data effectively.

4. Model Selection

Three **supervised learning models** were selected based on their effectiveness in text classification tasks:

A. Logistic Regression (LR)

- A **linear model** that predicts sentiment based on a weighted combination of features.
- Uses a **sigmoid activation function** to output probability scores for each sentiment class.
- **Advantages:** Fast, efficient, and well-suited for large text datasets.

B. Bernoulli Naïve Bayes (BNB)

- A **probabilistic classifier** based on Bayes' theorem, assuming feature independence.
- Works best for **binary features**, making it effective for TF-IDF data.
- **Advantages:** Fast and efficient for text classification, even with large datasets.

C. Support Vector Machine (SVM)

- A **non-linear model** that identifies the best decision boundary for classification.
- Uses a **kernel function** to transform data into a higher-dimensional space for better separation.
- **Advantages:** High accuracy and resistance to overfitting, but computationally expensive.

These models were selected to compare their performance and determine the best approach for Twitter sentiment classification.

5. Model Training

The dataset was **split into training and testing sets** using train_test_split() from sklearn.model_selection:

- **60% of the data** was used for training.
- **40% of the data** was used for testing.

Each model was trained on the **pre-processed TF-IDF feature vectors**. The training process involved:

- **Fitting the model to the training data.**

- **Optimizing hyperparameters** (e.g., regularization strength in Logistic Regression, kernel type in SVM).
- **Ensuring model generalization** to avoid overfitting.

After training, the models were evaluated on unseen test data.

6. Model Evaluation

To compare model performance, several evaluation metrics were used:

A. Accuracy

- Measures the proportion of correctly classified tweets.
- A higher accuracy indicates better overall performance.

B. Precision, Recall, and F1-Score

- **Precision:** Measures how many predicted positive tweets are actually positive.
- **Recall:** Measures how many actual positive tweets were correctly identified.
- **F1-Score:** The harmonic mean of Precision and Recall, useful for imbalanced datasets.

C. ROC-AUC Score

- Measures the model's ability to distinguish between positive and negative tweets.
- A higher AUC score indicates better classification performance.

D. Execution Time

- Measures how quickly each model processes and classifies tweets.

Each model was evaluated using these metrics, and results were compared to determine the most efficient classifier.

7. Model Deployment

Since the primary goal of this project was to compare different machine learning models for sentiment classification, no deployment phase was included. The project focused on identifying the **best-performing model** rather than integrating it into a real-world application.

Summary of the Model Description

Step	Description
Data Collection	Used Sentiment140 dataset with labeled tweets.
Data Preprocessing	Cleaned text by removing stopwords, special characters, and performing tokenization and lemmatization.
Feature Selection	Applied TF-IDF vectorization to convert text into numerical format.
Model Selection	Compared three classifiers: Logistic Regression, Bernoulli Naïve Bayes, and Support Vector Machine.
Model Training	Split data into training and testing sets (60-40 split) and trained models using the TF-IDF features.
Model Evaluation	Assessed models based on accuracy, F1-score, ROC-AUC, and execution time.
Model Deployment	Not included, as the project focused on model comparison rather than real-world deployment.

SYSTEM ANALYSIS

2. SYSTEM ANALYSIS

2.1 Feasibility Study

A feasibility study evaluates the viability of implementing the Twitter Sentiment Analysis (TSA) system. This assessment considers various aspects to determine if the project is practical, efficient, and beneficial.

Technical Feasibility:

The TSA system leverages **Natural Language Processing (NLP)** and **Machine Learning (ML)** techniques for sentiment classification. The project makes use of **NLTK, Scikit-learn, and TF-IDF vectorization** for preprocessing and feature extraction. These libraries offer optimized methods for handling text-based data, ensuring an optimized and efficient model-building process.

- **NLTK**: Used for text preprocessing techniques such as tokenization, stop-word removal, lemmatization, and stemming.
- **TF-IDF Vectorization**: Converts textual data into numerical representations for better model performance.
- **Scikit-learn**: Provides efficient machine learning algorithms, including **Naïve Bayes, Logistic Regression, and Support Vector Machines (SVM)** for sentiment classification.

Operational Feasibility:

The primary goal of this project is to provide **accurate sentiment analysis** of tweets, helping businesses, researchers, and policymakers understand public opinions and trends. The system ensures high interpretability and effective handling of large-scale social media data

Target Audience: Market analysts, researchers, businesses, and social media strategists.

- **Key Benefits:** Helps in trend analysis, opinion mining, and decision-making.

Economic Feasibility:

The project is developed using **open-source tools**, reducing the financial burden. The system is designed to run on platforms like Google Colab and Jupyter Notebook, ensuring cost-effective model training and evaluation

- **No licensing costs**: Uses publicly available datasets and free libraries.
- **Minimal computational cost**: Can run on standard hardware configurations with cloud support.

Legal and Ethical Feasibility:

The TSA system ensures compliance with ethical AI standards by strictly using **publicly available datasets** like **Sentiment140**. This eliminates privacy concerns and avoids direct user data collection. Ethical considerations include:

- **No real-time tweet scraping**, ensuring no privacy breaches.
- **Fair and unbiased model training**, avoiding discrimination in sentiment classification.

2.2 Existing System

Many existing sentiment analysis models have the following limitations:

- **High Computational Demand:** Most modern NLP models require **significant processing power**, making them inefficient for real-time applications.
- **Limited Contextual Understanding:** Challenges in detecting sarcasm, slang, and informal language lead to misinterpretations.

- **Dependency on Pre-trained Models:** Systems that rely on pre-trained deep learning models may not generalize well to new datasets or different domains.
- **Lack of Flexibility:** Many rule-based sentiment analyzers have **fixed constraints**, limiting their adaptability to diverse datasets.

2.3 Proposed System

To overcome these issues, the proposed Twitter Sentiment Analysis system integrates NLP and ML techniques to enhance sentiment prediction. The core functionalities of the system include:

- **Text Preprocessing:**
    - Tokenization: Splitting text into words.
    - Stop-word removal: Eliminating common words that do not contribute to sentiment analysis.
    - Lemmatization & Stemming: Reducing words to their base or root form.
- **Feature Extraction:**
    - **TF-IDF Vectorization** transforms text into numerical values for model input.
- **Machine Learning Algorithms:**
    - **Naive Bayes:** A probabilistic classifier based on Bayes' theorem.
    - **Logistic Regression:** A statistical method for binary classification.
    - **Support Vector Machines (SVM):** An effective classification algorithm for text data.

- **Static Dataset Analysis:**
    - Uses **Sentiment140**, a dataset containing labelled tweets with sentiment annotations.
- **Performance Evaluation:**
    - Models are assessed based on **accuracy, F1-score, and computational efficiency**.

This structured approach ensures a scalable, interpretable, and efficient sentiment analysis model.

SYSTEM CONFIGURATION

3. SYSTEM CONFIGURATION

In the software development life cycle (SDLC), system configuration is the first step of major importance. The entire concentration is on gathering the functional and the non-functional requirements for the product to be developed and estimating the feasibility of those attributes. Complete understanding of the requirements leads to the successful development of the software.

3.1 Hardware Requirements

Execution Environment

- **Google Colab** (Primary execution platform, handles all model processing).
- **Stable internet connection** (Required for accessing Google Colab, datasets, and running models).

Optional Local Requirements (For Offline Preprocessing, If Needed)

- **Processor (CPU):** Intel Core i5 (8th Gen or higher) / AMD Ryzen 5.
- **RAM:** 8GB (Sufficient for small-scale data processing).
- **Storage:** 256GB SSD (For storing datasets and project files).
- **Operating System:** Windows 10/11, Linux (Ubuntu preferred), macOS.

Recommended Local Requirements (If Running Models on a Personal PC Instead of Colab)

- **Processor (CPU):** Intel Core i7 (10th Gen or higher) / AMD Ryzen 7.
- **RAM:** 16GB (For handling larger datasets efficiently).
- **Storage:** 512GB SSD (For faster data processing and storage).
- **Internet:** Required for dataset access, Colab usage, and cloud-based execution.

3.2 SOFTWARE SPECIFICATION

3.2.1 GOOGLE COLAB

Google Colab (Colaboratory) is a cloud-based Jupyter notebook environment that allows users to write and execute Python code through a web browser. It is particularly beneficial for machine learning and data science projects, as it provides a ready-to-use development environment with pre-installed ML libraries. Colab is integrated with Google Drive, enabling seamless file management and collaboration. It supports popular libraries such as Scikit-learn, NumPy, Pandas, and NLTK, making it an excellent tool for sentiment analysis.

Note: Some additional libraries may require manual installation, such as downloading NLTK stopwords (`nltk.download('stopwords')`).

Key Features:

✓ Cloud-based execution, eliminating the need for local setup.

✓ Automatic saving and easy integration with Google Drive.

✓ Supports Jupyter notebooks and multiple Python libraries.

✓ Easy collaboration with team members.

3.2.2 PYTHON

Python is an open-source, high-level programming language known for its simplicity, readability, and versatility. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. In this project, Python is used for natural language processing (NLP), sentiment analysis, and machine learning using libraries like Scikit-learn, NLTK, and Pandas.

This project also utilizes the **TF-IDF vectorization technique** for text feature extraction, converting text data into numerical representations for sentiment analysis.

Key Features:

✓ High-level, easy-to-read syntax.

✓ Extensive libraries for machine learning (Scikit-learn), NLP (NLTK), and data handling (Pandas).

✓ Platform-independent and open-source.

✓ Supports integration with Jupyter notebooks for interactive data analysis.

3.2.3 NUMPY

NumPy (Numerical Python) is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these data structures efficiently. In this project, **NumPy is indirectly used in ML evaluations through Scikit-learn for handling numerical computations** such as accuracy calculation and confusion matrix generation.

Key Features:

✓ Supports multi-dimensional arrays and matrices.

✓ Optimized mathematical functions for numerical operations.

✓ Efficient memory management and performance optimization.

✓ Used in ML evaluations through Scikit-learn for numerical computations.

3.2.4 PANDAS

Pandas is a powerful open-source library for data manipulation and analysis. It provides data structures such as Series (1D array) and DataFrame (2D table) that enable efficient handling of structured data. In this project, **Pandas is primarily used for reading, processing, and handling the Sentiment140 dataset before applying sentiment analysis techniques**. Additionally, **Pandas is used for dataset filtering and exploratory data analysis (EDA) before model training** to ensure clean and structured input data.

Key Features:

✓ Provides easy-to-use data structures: Series and DataFrame.
✓ Functions for data cleaning, transformation, and analysis.
✓ Supports reading and handling CSV datasets (e.g., Sentiment140).
✓ Enables dataset filtering and exploratory data analysis before training models.
✓ Works well with other libraries like NumPy and Scikit-learn.

SYSTEM DESIGN

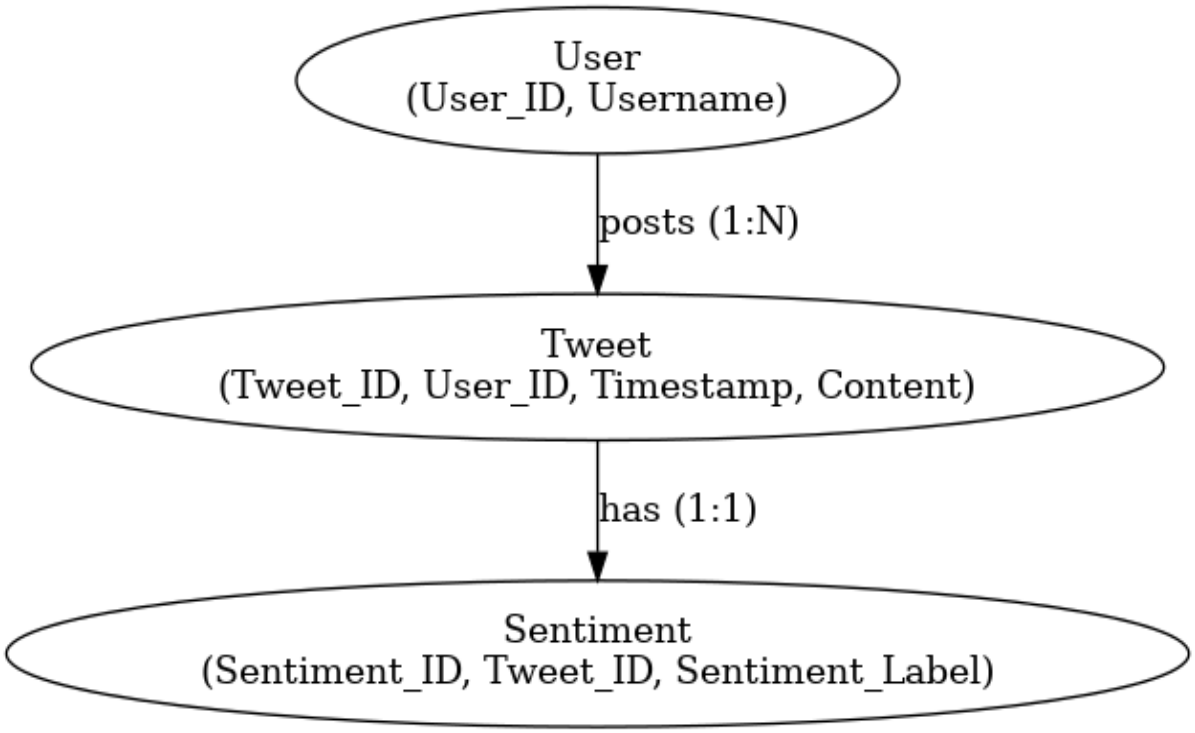
4.1 SYSTEM ARCHITECTURE:

A system architecture is the conceptual model that defines the structure, behaviour, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviour of the system.

A system architecture can consist of system component and the sub-systems developed, that will work together to implement the overall system. There have been efforts to formalize language to describe system architecture, collectively these are called architecture description languages (ADLs).

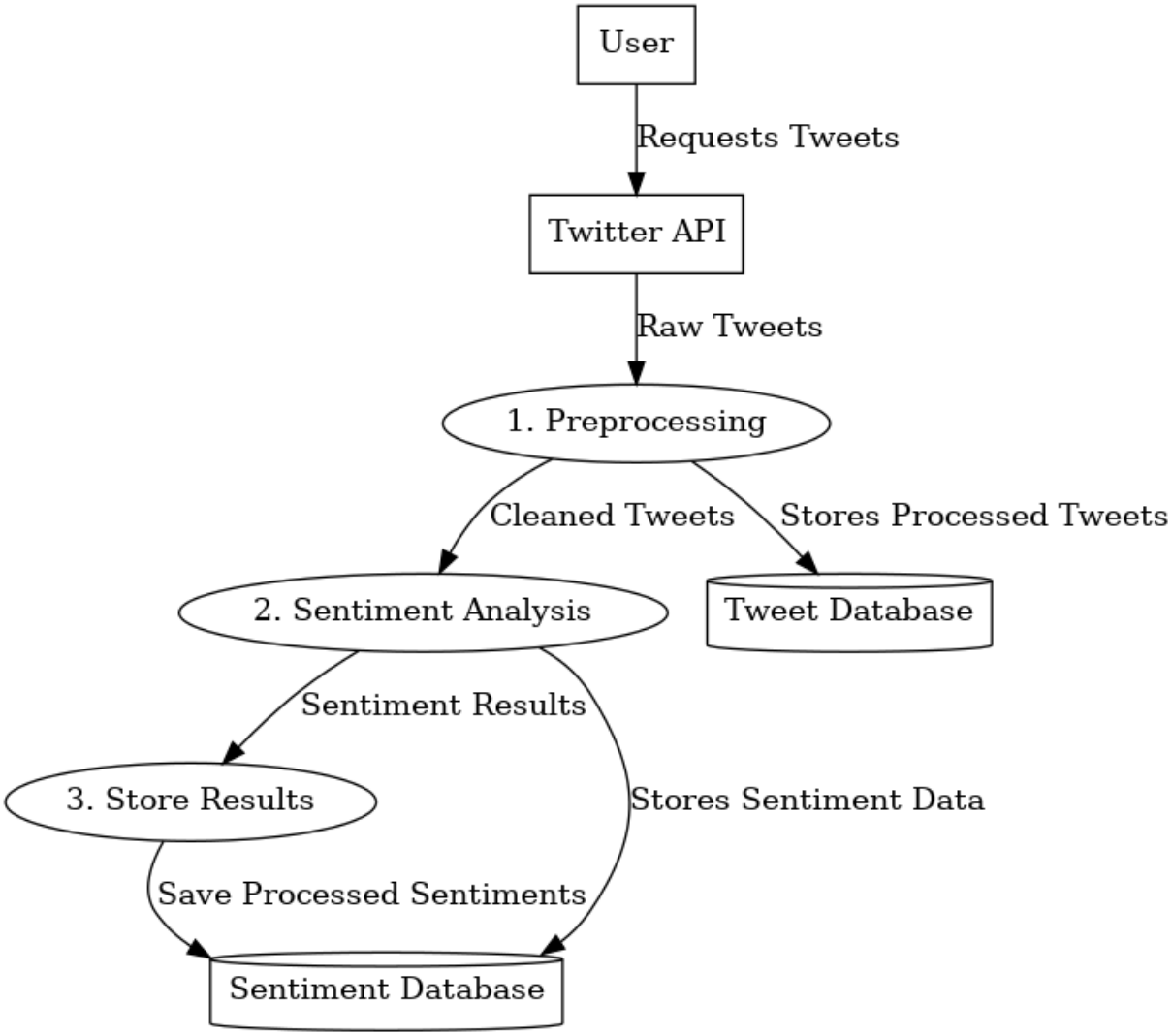
4.2 ER DIAGRAM

ER models stand for Entity Relationship model. It is a high-level data model. This model is used to define the data elements and relationships for a specified system. It develops a conceptual design for the database. It also develops a very simple and easy design view of the data. In ER modelling, the database structure is portrayed as a diagram called an entity relationship diagram.

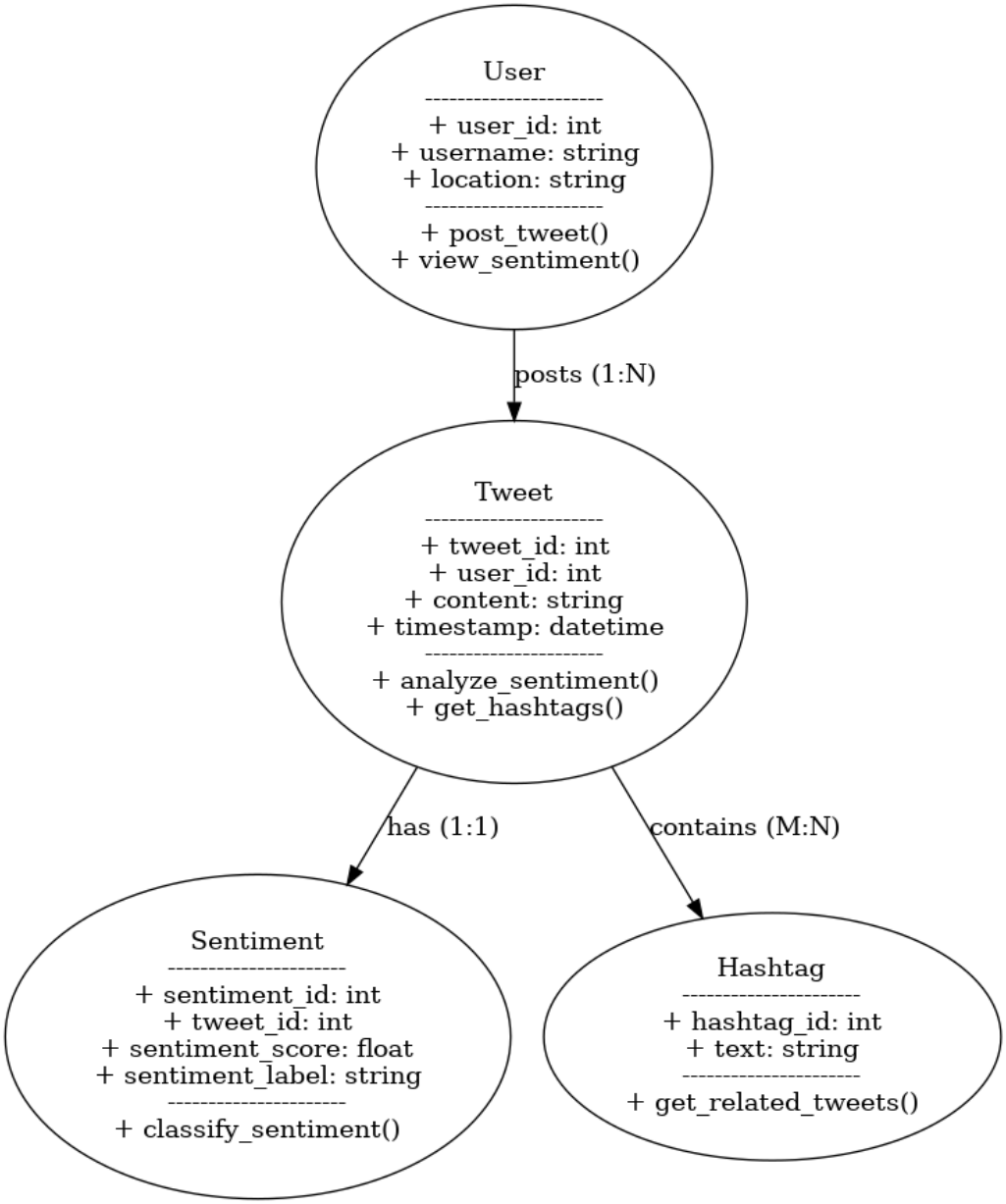


4.3 Data Flow Diagram

A Data Flow Diagram (DFD) is the starting point of the design phase that functionally decomposes the requirements specifications. A DFD consists of a series of bubbles joined by lines. The bubbles represent data transformation, and the lines represent data flows in the system. A DFD describes what data flows rather than how they are processed, so it does not use hardware, software, and data structure. A data flow diagram is a graphical representation of the flow of data through and information system.



4.5 UML DIAGRAM



SYSTEM TESTING

5. SYSTEM TESTING

5.1 Unit Testing

Unit testing is conducted to validate individual components of the system:

- **Preprocessing Functions:** Ensures tokenization, stop-word removal, lemmatization, and stemming operate correctly.
- **Feature Extraction:** Checks proper application of **TF-IDF vectorization**.
- **Model Training:** Verifies correct training of classification algorithms without errors.
- **Data Handling:** Confirms **Sentiment140 dataset** loads properly and is transformed into required formats.

5.2 Integration Testing

Integration testing assesses the collaboration between different system modules:

- **Preprocessing & Model Training:** Ensures processed text is correctly inputted into ML models.
- **Feature Extraction & Classification:** Confirms that numerical representations from **TF-IDF** are effectively used for sentiment classification.
- **Model Evaluation:** Ensures fair model comparisons using the same dataset and parameters.

5.3 Acceptance Testing

This phase ensures that the system meets expectations:

- **Accuracy Testing:** Comparing predicted sentiments with actual labels.
- **Performance Evaluation:** Measuring **execution time, precision, recall, and F1-score**.

- **Result Analysis:** Ensuring classification results align with dataset sentiment distribution.

5.4 System Testing

Comprehensive validation of the TSA system's end-to-end workflow:

- **Processing Large Datasets:** Verifies if the system can efficiently process bulk tweets.
- **Feature Extraction & Classification:** Validates seamless integration of TF-IDF and classification models.
- **Robustness Against Noisy Data:** Checks preprocessing techniques against irrelevant or ambiguous tweets.

5.5 White Box Testing

White box testing evaluates the internal logic and flow of the system:

- **Code Review:** Ensures preprocessing scripts work as intended.
- **Algorithm Debugging:** Tests if ML model implementations function correctly.
- **Feature Transformation Analysis:** Confirms TF-IDF applies proper weightage to terms.
- **Edge Case Handling:** Ensures the model can process a wide variety of text inputs, including abbreviations and symbols.
- **Performance Optimization:** Evaluates the efficiency of different ML algorithms under varying computational loads.

5.6 Black Box Testing

Black box testing focuses on external system behavior:

- **Input Validation:** Tests how the system handles unexpected inputs (e.g., emojis, hashtags, misspelled words).
- **Output Verification:** Ensures sentiment predictions align with expected results.
- **Model Performance Comparison:** Ensures fairness in evaluation metrics across models.
- **Scalability Testing:** Evaluates how the system performs under increasing dataset sizes.
- **Stress Testing:** Analyzes the system's behavior under high computational load.

By performing **rigorous testing** at multiple levels, the TSA system ensures robustness, accuracy, and efficiency, making it a reliable solution for sentiment analysis on static datasets.

CONCLUSION

After evaluating the performance of different machine learning models for Twitter sentiment analysis, we can draw the following conclusions:

1. **Execution Time:** When comparing the execution time of the models, Bernoulli Naive Bayes (BNB) is the fastest, taking only 0.56 seconds. Logistic Regression (LR) follows with a runtime of 21.23 seconds, whereas Support Vector Machine (SVM) is the slowest, taking 97.08 seconds. This indicates that BNB is the most efficient model in terms of computation time, making it suitable for real-time applications.
2. **Accuracy:** Among the three models tested, Logistic Regression achieves the highest accuracy at 82%, followed by SVM at 81%, and Bernoulli Naive Bayes at 80%. This suggests that Logistic Regression is the most reliable model in correctly predicting sentiments in the dataset.
3. **F1-Score:** The F1-score measures the balance between precision and recall. The results show that Logistic Regression performs the best in terms of F1-score, achieving 82% for both class 0 (negative tweets) and class 1 (positive tweets). SVM follows with an F1-score of 81%, while Bernoulli Naive Bayes has the lowest F1-score at 80%.
4. **AUC Score:** The AUC (Area Under the Curve) score, which measures the overall performance of the model, is also highest for Logistic Regression at 82%, followed by SVM at 81%, and Bernoulli Naive Bayes at 80%. This further confirms the superior performance of Logistic Regression in sentiment classification.

Final Recommendation

Considering all the evaluation metrics, **Logistic Regression** emerges as the best-performing model in terms of accuracy, F1-score, and AUC score, though it is slower than Bernoulli Naive Bayes. On the other hand, if speed is a primary concern, Bernoulli Naive Bayes is the best option. SVM, while relatively accurate, is not the most efficient due to its high execution time. The choice of the best model ultimately depends on the specific requirements of the application, balancing performance and computational efficiency.

APPENDIX

7.1 CODING

```
import re
import numpy as np
import pandas as pd
import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('wordnet')
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix, classification_report
import time
from google.colab import drive
drive.mount('/content/drive')
```

```
DATASET_COLUMNS=['target','ids','date','flag','user','text']

DATASET_ENCODING = "ISO-8859-1"

df =
pd.read_csv('/content/drive/MyDrive/Project/training.1600000.processed.noemo
ticon.csv', encoding=DATASET_ENCODING,
names=DATASET_COLUMNS)

df.sample(5)

df.head()

df.columns

print('length of data is', len(df))

df.shape

df.info()

df.dtypes

np.sum(df.isnull().any(axis=1))

print('Count of columns in the data is:  ', len(df.columns))

print('Count of rows in the data is:  ', len(df))

df['target'].unique()

df['target'].nunique()

df.groupby('target').count()

ax = df.groupby('target').count().plot(kind='bar', title='Distribution of
data',legend=False)

ax.set_xticklabels(['Negative','Positive'], rotation=0)

text, sentiment = list(df['text']), list(df['target'])
```

```
import seaborn as sns
sns.countplot(x='target', data=df)
data = df[['text','target']]
data['target'] = data['target'].replace(4,1)
data['target'].unique()
data_pos = data[data['target'] == 1]
data_neg = data[data['target'] == 0]
dataset = pd.concat([data_pos, data_neg])
dataset['text'].tail()
dataset['text'] = dataset['text'].str.lower()
dataset['text'].tail()
stopwordlist = ['a', 'about', 'above', 'after', 'again', 'ain', 'all', 'am', 'an',
            'and', 'any', 'are', 'as', 'at', 'be', 'because', 'been', 'before',
            'being', 'below', 'between', 'both', 'by', 'can', 'd', 'did', 'do',
            'does', 'doing', 'down', 'during', 'each', 'few', 'for', 'from',
            'further', 'had', 'has', 'have', 'having', 'he', 'her', 'here',
            'hers', 'herself', 'him', 'himself', 'his', 'how', 'i', 'if', 'in',
            'into', 'is', 'it', 'its', 'itself', 'just', 'll', 'm', 'ma',
            'me', 'more', 'most', 'my', 'myself', 'now', 'o', 'of', 'on', 'once',
            'only', 'or', 'other', 'our', 'ours', 'ourselves', 'out', 'own', 're',
            's', 'same', 'she', "shes", 'should', "shouldve", 'so', 'some', 'such',
            't', 'than', 'that', "thatll", 'the', 'their', 'theirs', 'them',
```

```
            'themselves', 'then', 'there', 'these', 'they', 'this', 'those',
            'through', 'to', 'too', 'under', 'until', 'up', 've', 'very', 'was',
            'we', 'were', 'what', 'when', 'where', 'which', 'while', 'who', 'whom',
            'why', 'will', 'with', 'won', 'y', 'you', "youd", "youll", "youre",
            "youve", 'your', 'yours', 'yourself', 'yourselves']
STOPWORDS = set(stopwordlist)
def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in
STOPWORDS])
dataset['text'] = dataset['text'].apply(lambda text: cleaning_stopwords(text))
dataset['text'].head()
import string
english_punctuations = string.punctuation
punctuations_list = english_punctuations
def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_punctuations(x))
dataset['text'].tail()
def cleaning_repeating_char(text):
    return re.sub(r'(.)1+', r'1', text)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_repeating_char(x))
```

```
dataset['text'].tail()
def cleaning_URLs(data):
    return re.sub('((www.[^s]+)|(https?://[^s]+))', ' ', data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_URLs(x))
dataset['text'].tail()
def cleaning_numbers(data):
    return re.sub('[0-9]+', '', data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_numbers(x))
dataset['text'].tail()
!pip install nltk
import nltk
nltk.download('punkt_tab')
dataset['text'] = dataset['text'].apply(word_tokenize)
dataset['text'].head()
st = nltk.PorterStemmer()
def stemming_on_text(data):
    text = [st.stem(word) for word in data]
    return data
dataset['text'] = dataset['text'].apply(lambda x: stemming_on_text(x))
dataset['text'].head()
lm = nltk.WordNetLemmatizer()
def lemmatizer_on_text(data):
```

```
    text = [lm.lemmatize(word) for word in data]
    return data
dataset['text'] = dataset['text'].apply(lambda x: lemmatizer_on_text(x))
dataset['text'].head()
X = data.text
y = data.target
data_neg = data['text'][:800000] # selecting the negative tweets.
plt.figure(figsize=(20, 20))
wc = WordCloud(max_words=1000, width=1600, height=800,
              collocations=False).generate(" ".join(data_neg))
plt.imshow(wc)
data_pos = data['text'][800000:]  # selecting the positive tweets.
wc = WordCloud(max_words=1000, width=1600, height=800,
              collocations=False).generate(" ".join(data_pos))
plt.figure(figsize=(20, 20))
plt.imshow(wc)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40,
random_state=26105111)
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
```

```
X_test
def model_Evaluate(model):
    y_pred = model.predict(X_test)
    print(classification_report(y_test, y_pred))
    cf_matrix = confusion_matrix(y_test, y_pred)
    categories = ['Negative','Positive']
    group_names = ['True Neg','False Pos', 'False Neg','True Pos']
    group_percentages = ['{0:.2%}'.format(value) for value in cf_matrix.flatten()
/ np.sum(cf_matrix)]
    labels = [f'{v1}n{v2}' for v1, v2 in zip(group_names,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    sns.heatmap(cf_matrix, annot = labels, cmap = 'Blues',fmt = '',
    xticklabels = categories, yticklabels = categories)
    plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)
    plt.ylabel("Actual values" , fontdict = {'size':14}, labelpad = 10)
    plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)
from sklearn.metrics import accuracy_score
BNBmodel = BernoulliNB()
start = time.time()
BNBmodel.fit(X_train, y_train)
end = time.time()
print("The execution time of this model is {:.2f} seconds\n".format(end-start))
```

```
y_pred1 = BNBmodel.predict(X_test) # Now, y_pred1 is defined

print(f"Accuracy of Bernoulli Naive Bayes: {accuracy_score(y_test, y_pred1) * 100:.2f}%")

model_Evaluate(BNBmodel)

from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_test, y_pred1)

roc_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC CURVE')

plt.legend(loc="lower right")

plt.show()

SVCmodel = LinearSVC()

start = time.time()

SVCmodel.fit(X_train, y_train)

end = time.time()

print("The execution time of this model is {:.2f} seconds\n".format(end-start))
```

```
y_pred2 = SVCmodel.predict(X_test)  # Now, y_pred2 is defined
print(f"Accuracy of SVM: {accuracy_score(y_test, y_pred2) * 100:.2f}%")


model_Evaluate(SVCmodel)
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred2)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
LRmodel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
start = time.time()
LRmodel.fit(X_train, y_train)
end = time.time()
print("The execution time of this model is {:.2f} seconds\n".format(end-start))
```

```
y_pred3 = LRmodel.predict(X_test)

print(f"Accuracy of Logistic Regression: {accuracy_score(y_test, y_pred3) * 100:.2f}%")

model_Evaluate(LRmodel)

from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_test, y_pred3)

roc_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC CURVE')

plt.legend(loc="lower right")

plt.show()
```

7.2 SCREENSHOTS

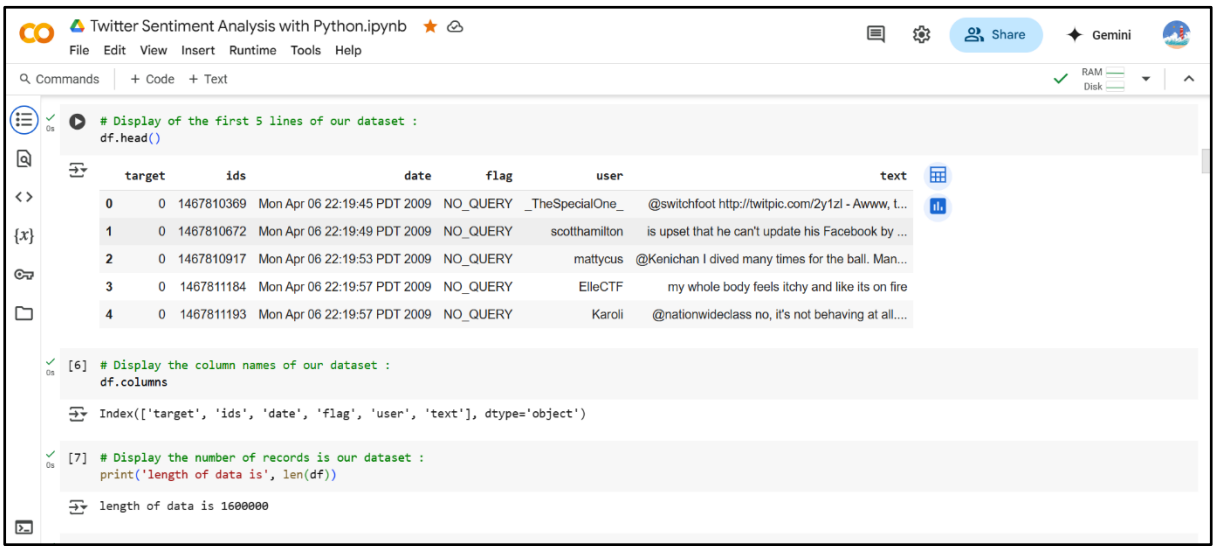


Figure 1: Sample of Sentiment140 Dataset Before Preprocessing

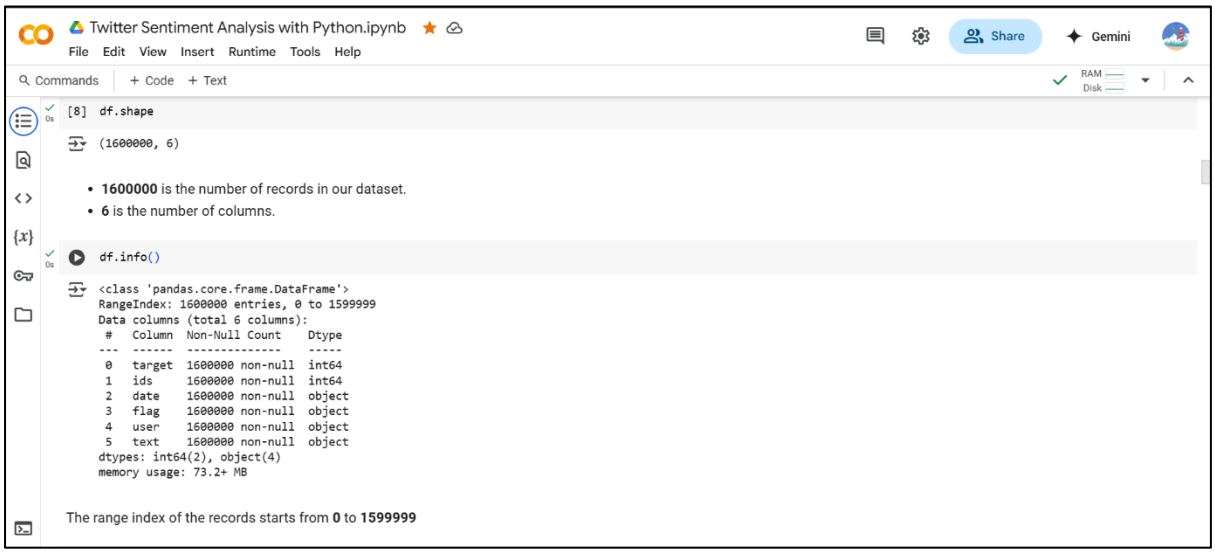


Figure 2 : Dataset Overview – Number of Records & Features

Figure 3: Sentiment Distribution in the Dataset

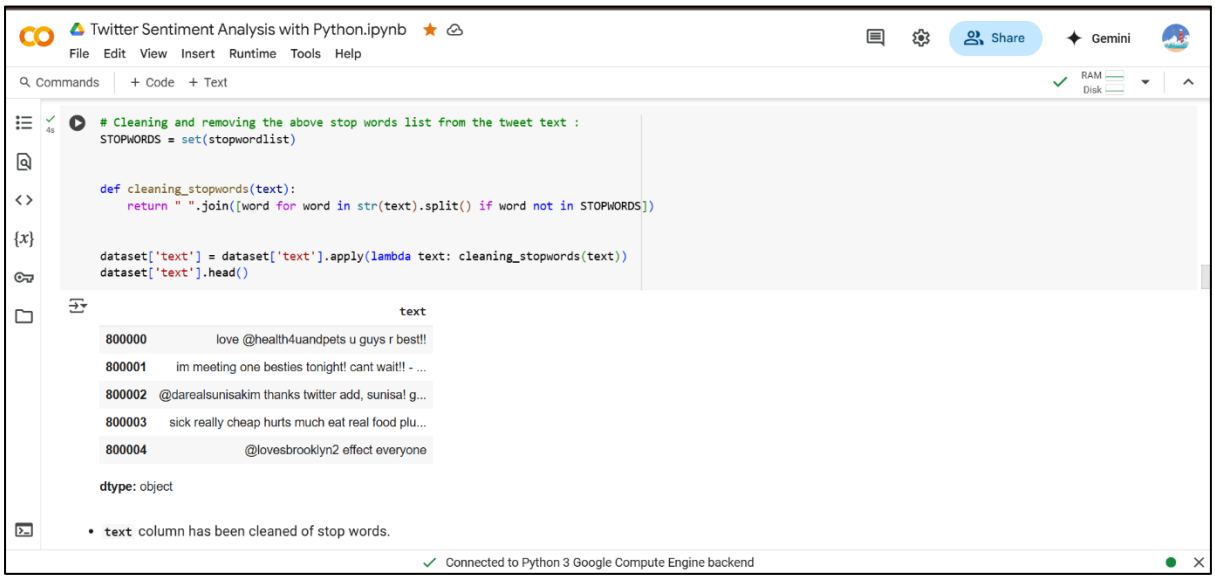


Figure 4: Text Preprocessing - Stopword Removal

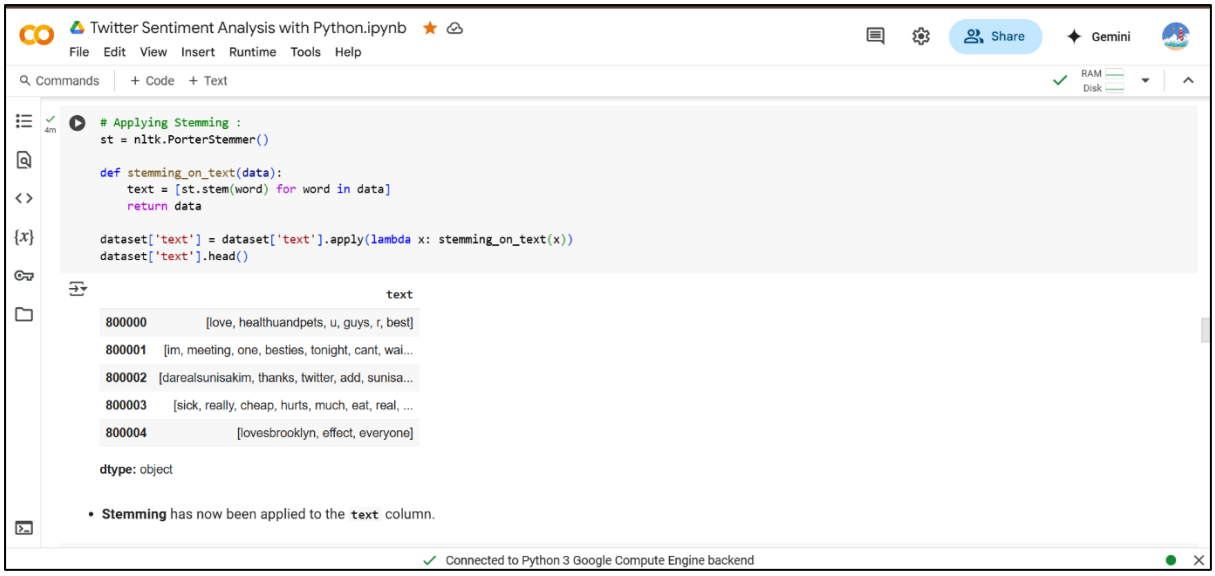


Figure 5: Text Preprocessing - Stemming

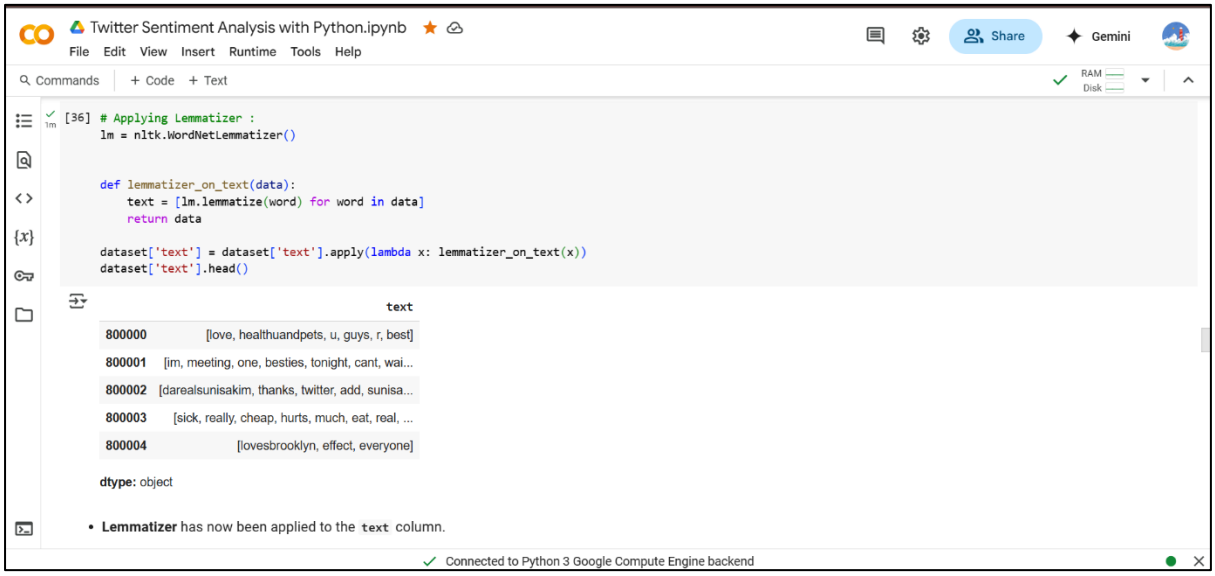


Figure 6: Text Preprocessing – Lemmatization

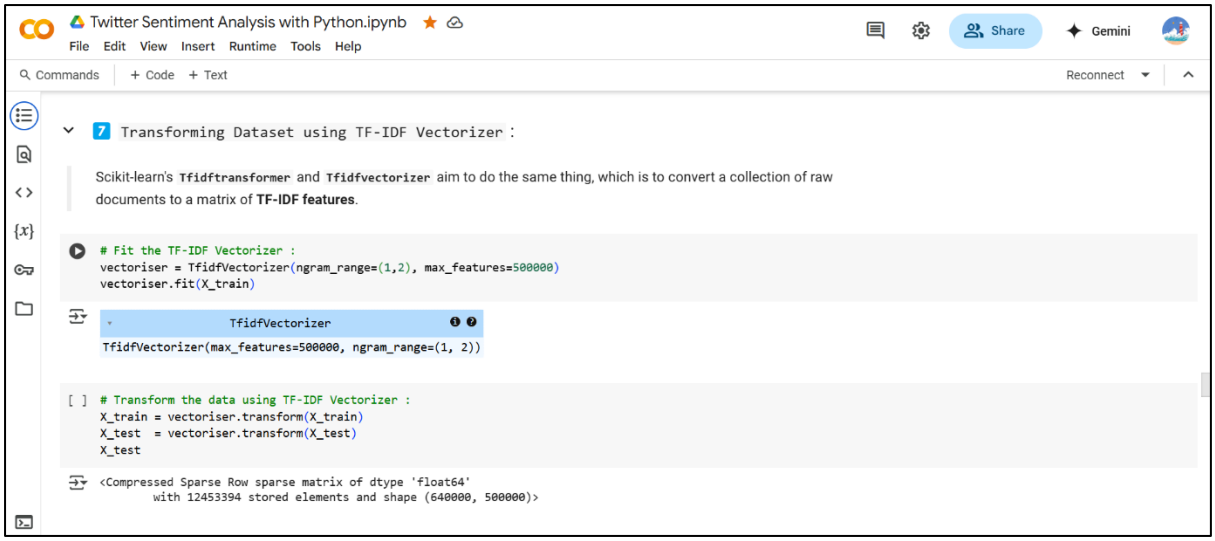


Figure 7: TF-IDF Vectorized Text Data

Figure 8 : Model Training & Evaluation metrics for BNB

Figure 9 : Model Training & Evaluation metrics for SVM

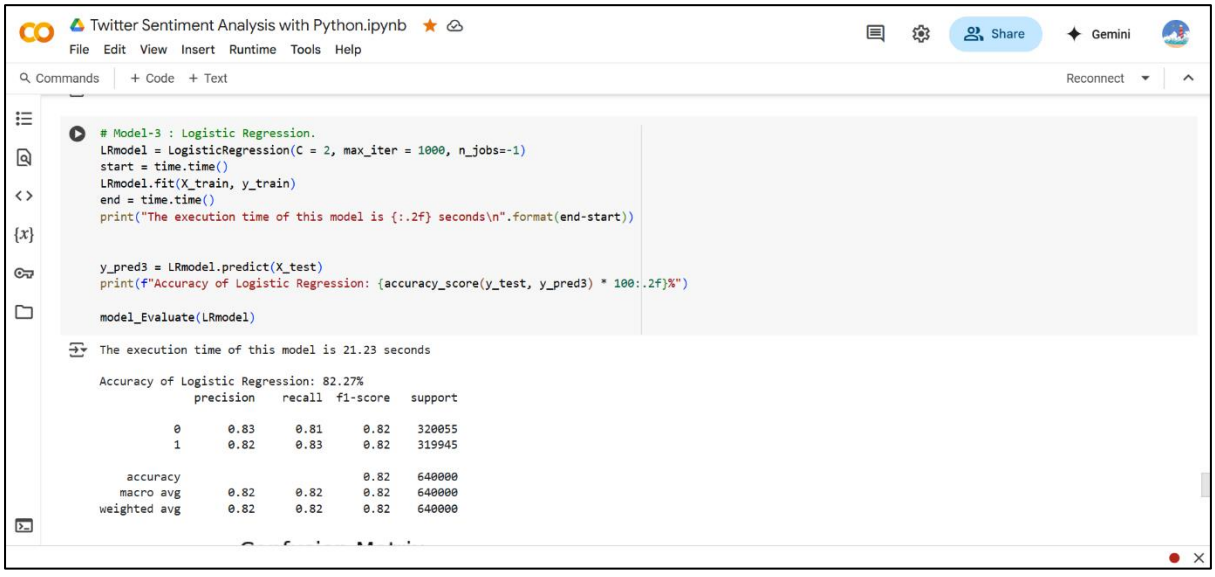
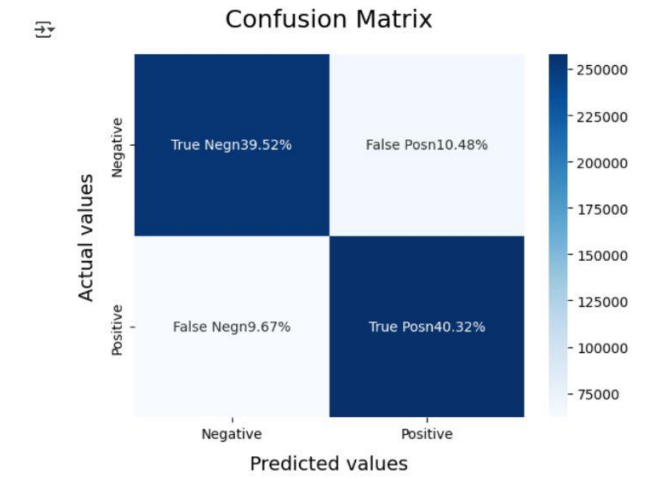
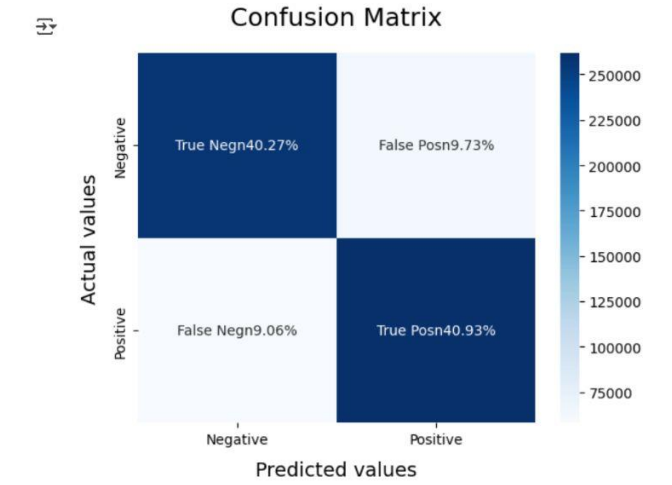


Figure 10 : Model Training & Evaluation metrics for LR

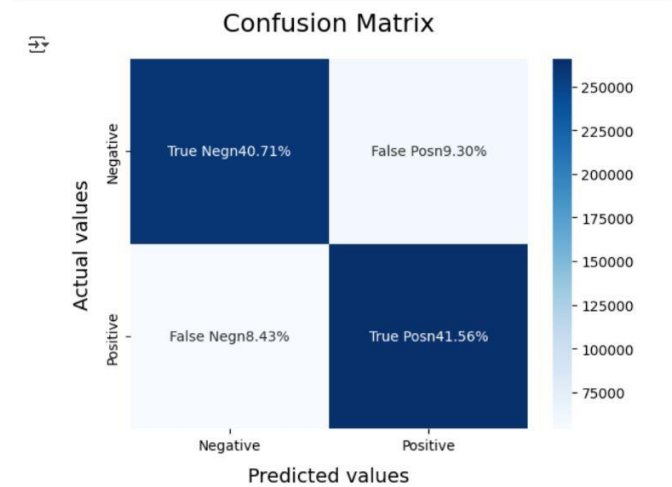
Figure 11: Confusion Matrix for Sentiment Classification



BNB

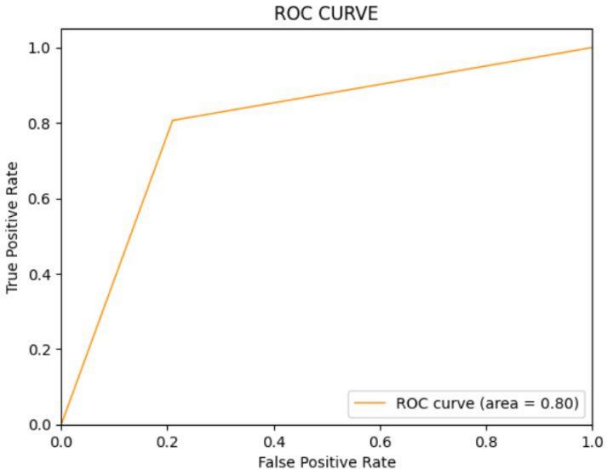


SVM

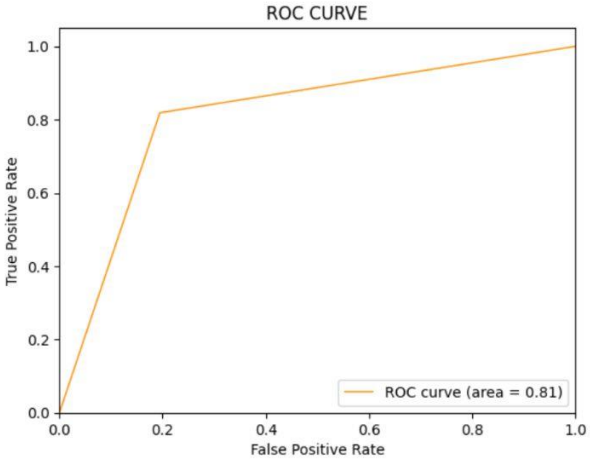


LR

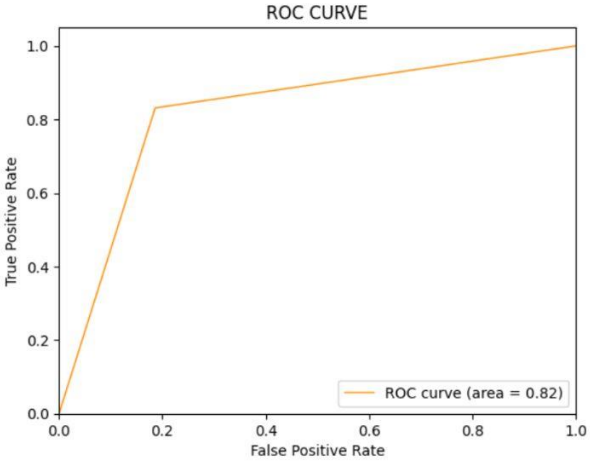
Figure 12 : ROC Curve Comparison for Models



BNB



SVM



LR

After evaluating all models, we can conclude the following details :

Model	Accuracy	F1-score (class 0)	F1-score (class 1)	AUC Score	Execution time
Bernoulli Naive Bayes (BNB)	80%	80%	80%	80%	0.56 seconds
Support Vector Machine (SVM)	81%	81%	81%	81%	97.08 seconds
Logistic Regression (LR)	82%	82%	82%	82%	21.23 seconds

- `Execution time` : When it comes to comparing the running time of models, `Bernoulli Naive Bayes` performs faster than `Logistic Regression`, which in turn runs faster than `SVM`.
- `Accuracy` : When it comes to model accuracy, `logistic regression` performs better than `SVM`, which in turn performs better than `Bernoulli Naive Bayes`.
- `F1-score` : The F1 Scores for **class 0** and **class 1** are :
    - For **class 0** (negative tweets) :

      ```
      accuracy : BNB (= 0.80) < SVM (=0.81) < LR (= 0.82)
      ```

    - For **class 1** (positive tweets) :

      ```
      accuracy : BNB (= 0.80) < SVM (=0.81) < LR (= 0.82)
      ```

- `AUC Score` : All three models have the same ROC-AUC score.

  ```
  AUC score : BNB (= 0.80) < SVM (=0.81) < LR (= 0.82)
  ```

- We therefore conclude that `logistic regression` is the **best model** for the above dataset as it has higher accuracy than other models.

Figure 13: Performance Comparison of Sentiment Classification Models

BIBLIOGRAPHY

REFERENCES

1. Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media.

2. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

3. Rajaraman, A., & Ullman, J. D. (2011). Mining of Massive Datasets. Cambridge University Press.

4. Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th ed.). Pearson.

5. Go, A., Bhayani, R., & Huang, L. (2009). Twitter Sentiment Classification Using Distant Supervision. Stanford University Technical Report.

6. Python Software Foundation. (2024). Python 3 Documentation. Retrieved from https://docs.python.org/3/

7. Kaggle. (n.d.). Sentiment140 Dataset for Twitter Sentiment Analysis. Retrieved from https://www.kaggle.com/datasets/kazanova/sentiment140

8. Salton, G., & Buckley, C. (1988). Term-Weighting Approaches in Automatic Text Retrieval. Information Processing & Management, 24(5), 513-523.

9. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer.

10. Labriji, S. (n.d.). Twitter Sentiment Analysis with Python. GitHub Repository. Retrieved from https://github.com/labrijisaad/Twitter-Sentiment-Analysis-with-Python/blob/main/notebooks/Twitter%20Sentiment%20Analysis%20with%20Python.ipynb