MALAYALAM SENTIMENT ANALYSIS USING MACHINE LEARNING TECHNIQUES

A Project Report

Submitted in partial fulfilment of the requirements for the award of degree of

Master Of Science (Information Technology)

Course: Introduction To Big Data - Laboratory

Course Code: CAP457

Submitted by :

Bimal K Venu - 12302416

Name of Supervisor :

Manik Mehra

Lovely Professional University

Punjab

2025-26 (Term 4)

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my respected project supervisor, Dr. Manik Mehra, for their invaluable guidance, unwavering support, and insightful feedback throughout the entire duration of this project. Their expertise, patience, and encouragement have been instrumental in shaping the direction of my work, and I truly appreciate their dedication in helping me grow both academically and professionally.

Additionally, I extend my heartfelt thanks to the esteemed faculty of the Computer Science and Engineering department at **Lovely Professional University** for providing essential resources, valuable knowledge, and a motivating academic environment that has significantly contributed to the success of this project.

Furthermore, I am deeply grateful to my family and friends for their continuous encouragement, unwavering belief in my abilities, and emotional support throughout this journey. Their presence has been a pillar of strength, and I sincerely appreciate their role in keeping me motivated and focused.

This project would not have been possible without the collective support and inspiration from all those mentioned above, and I am truly thankful for their contributions to my learning and growth.

Bimal K Venu - 12302416

ABSTRACT

The exponential growth of user-generated content on digital platforms has created a compelling need for automated sentiment analysis systems, especially in regional languages like Malayalam, which lack sufficient resources compared to widely spoken languages. This project, titled **"Malayalam Sentiment Analysis using Machine Learning Techniques,"** aims to develop a lightweight, efficient, and accurate system capable of classifying Malayalam text into positive, negative, or neutral sentiments.

The system leverages traditional machine learning approaches such as the Naive Bayes classifier combined with TF-IDF (Term Frequency-Inverse Document Frequency) vectorization, optimized for Malayalam language patterns. The dataset is preprocessed with normalization, stopword removal, and data balancing, while simple synonym-based augmentation techniques are incorporated to improve model performance. Additionally, a rule-based fallback mechanism has been integrated to refine the sentiment prediction, especially for neutral and negative sentiments.

The application is implemented as a Flask-based web platform with a user-friendly interface, allowing real-time sentiment analysis through text input. The model is trained on a curated dataset and evaluated using accuracy, precision, recall, and F1-score metrics. The system provides satisfactory performance and serves as a foundation for further improvements using advanced deep learning and transformer-based models.

This project showcases the potential of combining traditional NLP techniques with rule-based logic to overcome the challenges of low-resource language processing and contributes to the growing field of regional language AI solutions.

TABLE OF CONTENTS

1. INTRODUCTION

In today's digital era, vast volumes of user-generated content—ranging from product reviews and social media posts to online forums—are created every minute. Organizations, researchers, and decision-makers increasingly rely on automated tools to mine these data for insights. Sentiment analysis, or opinion mining, is the subfield of Natural Language Processing (NLP) that aims to determine the emotional tone—positive, negative, or neutral—behind a piece of text. While commercial and open-source solutions for sentiment analysis abound in English and other major world languages, regional languages such as Malayalam have received relatively little attention. Consequently, practitioners working with Malayalam content face a lack of robust language resources, pretrained models, and large annotated corpora. This project addresses that gap by developing a lightweight yet effective sentiment analysis system for Malayalam, using classical machine-learning techniques and a simple web interface to demonstrate real-time performance.

Sentiment analysis in regional languages presents unique challenges. Indian languages like Malayalam exhibit rich morphology, agglutination, and complex script characteristics. Words can be compounded, inflected, and fused with particles in ways that make tokenization and feature extraction non-trivial. Moreover, publicly available labeled datasets for Malayalam sentiment are scarce, and off-the-shelf deep-learning models cover the language poorly. Faced with these constraints, this work explores a hybrid approach: combining TF-IDF (Term Frequency–Inverse Document Frequency) vectorization with a Multinomial Naive Bayes classifier for core sentiment predictions, augmented by lightweight rule-based fallbacks to catch neutral and negative cues that the statistical model might miss. Finally, to make the system accessible to non-technical users and to facilitate demonstration, a Flask-based web interface is developed, allowing users to type Malayalam sentences into a browser form and obtain instant sentiment judgments.

1.1 Background and Motivation

The proliferation of smartphones and social media platforms in Kerala over the last decade has led to an exponential increase in Malayalam-language content online. From consumer opinions on e-commerce sites, restaurant and movie reviews on local portals, to social commentary on Facebook and Twitter, Malayalam speakers generate a tremendous volume of text data every day. Organizations—such as e-tailers, hospitality chains, and local businesses—would greatly benefit from automated tools that can sift through this feedback and surface actionable insights. For instance, a Malabar hotel chain could analyze guest reviews to detect recurring service complaints, or a movie producer might monitor audience reactions to trailers and scenes.

However, most existing sentiment analysis products (e.g., Google Cloud Natural Language, AWS Comprehend) either lack Malayalam support or offer only rudimentary handling, often translating into English first and then classifying—which introduces translation errors and cultural nuance loss. Academic research in Indian language sentiment is growing, but few practical systems exist for Malayalam. This project is thus motivated by three core needs:

1. **Language-Specific Accuracy:** A system trained directly on Malayalam text can capture idiomatic expressions, colloquialisms, and morphological variants that a translation-based pipeline would miss.

2. **Resource Efficiency:** Rather than requiring massive GPU-trained transformer models, a TF-IDF + Naive Bayes approach can be trained and deployed on modest hardware, making it suitable for small labs or local businesses without ML infrastructure.

3. **Usability:** By packaging the classifier into a simple Flask app, end-users—such as business analysts, students, or local NGOs—can access sentiment insights via a web browser without any installation or coding expertise.

Thus, the project stands at the intersection of practicality and local language empowerment, leveraging classical NLP techniques to deliver a usable Malayalam sentiment analysis tool.

1.2 Problem Statement

Despite growing interest in NLP for Indian languages, Malayalam sentiment analysis remains under-served. Key issues include:

- **Limited Annotated Data:** Publicly available Malayalam datasets rarely exceed a few thousand labeled sentences, and often skew heavily toward one sentiment class (e.g., overwhelmingly positive movie reviews). This imbalance hinders robust classifier training and leads to bias.

- **Morphological Complexity:** Malayalam's agglutinative nature means that a single root verb may appear in dozens of inflected forms. Without adequate morphological analysis, statistical models treat each form as a separate token, diluting the feature signal.

- **Neutral and Mixed Sentiment:** Many real-world sentences express factual statements ("ഞാൻ ഇന്നലെ സ്കൂളിൽ പോയി") or mixed emotions ("ചിത്രം നല്ലതായിരുന്നു, പക്ഷേ കഥ പ്രതീക്ഷിച്ചതിനേക്കാൾ biasa ം") . Distinguishing neutral from subtle positive or negative can confound simple classifiers.

- **Deployment Barriers:** Even if a researcher builds a high-accuracy model in a notebook, wrapping it into a user-friendly interface and deploying on local hardware or cloud requires additional engineering.

Therefore, the specific problem this project tackles is:

How can we build an efficient, accurate, and user-friendly sentiment analysis system for Malayalam that works well on limited data, handles morphological variation, and provides neutral/negative safeguards, all while requiring minimal computational resources?

Addressing this question involves designing a tailored pipeline—data preprocessing, TF-IDF feature extraction, Naive Bayes classification, and rule-based fallback logic—then packaging it into a Flask web app for demo and evaluation.

1.3 Objectives of the Project

The primary objective of this project is to develop a Malayalam Sentiment Analysis System that can efficiently classify Malayalam text as positive, negative, or neutral. The specific objectives of the project are outlined as follows:

1. **Corpus Preparation and Preprocessing**

    - **Data Collection**: The first step is to create a balanced dataset consisting of Malayalam sentences labeled with their corresponding sentiments: positive, negative, and neutral. This dataset may be gathered from available resources such as reviews, comments, or social media posts. In the absence of a sufficiently large dataset, existing datasets (e.g., from Kaggle or other public repositories) can be used, followed by additional labeling and cleaning.

    - **Data Cleaning**: Once the data is collected, it must be preprocessed to remove any irrelevant characters, such as punctuation marks, stop words, and special symbols. Additionally, text normalization techniques, including lowercasing all words and standardizing Unicode representations, will be applied. This step ensures that the text is ready for further processing and analysis.

    - **Tokenization**: Tokenization is a crucial step where sentences are split into smaller components, usually words or phrases. This helps the machine learning model to better understand the structure and meaning of the sentences.

2. **Feature Extraction Using TF-IDF**

    - **TF-IDF Vectorization**: After preprocessing the text data, the next step is to convert the textual data into a numerical format that the machine learning model can process. The Term Frequency-Inverse Document Frequency (TF-IDF) method is used for this purpose. This technique captures the importance of each word in a document relative to the entire corpus, considering both its frequency within the document and how commonly it appears across all documents.

    - **N-Gram Features**: In addition to unigrams (individual words), the system will also extract bigrams (two-word sequences) and trigrams (three-word sequences) to account for context and the semantic relationship between words. This is particularly important in languages like Malayalam, where word meanings can change based on the surrounding context.

    - **Feature Selection and Optimization**: Fine-tune the vectorizer's parameters, such as min_df (minimum document frequency) and max_df (maximum document frequency), to control the inclusion of features that are too rare or too common, ensuring that the features used are both meaningful and diverse.

3. **Training a Multinomial Naive Bayes Classifier**

    - **Naive Bayes Classifier**: The next objective is to implement a Multinomial Naive Bayes classifier, which is a well-established model for text classification tasks. The classifier will be trained using the extracted TF-IDF features and the labeled dataset. It works by assuming that the presence of a particular word in a document is independent of the presence of other words, a simplifying assumption that works surprisingly well for text classification problems.

    - **Model Training and Evaluation**: The classifier will be trained using a portion of the dataset and evaluated on a held-out test set. Metrics like accuracy, precision, recall, and F1-score will be used to evaluate the model's performance. These metrics will help assess how well the model is distinguishing between positive, negative, and neutral sentiments in Malayalam text.

    - **Model Tuning**: During training, the hyperparameters of the Naive Bayes model will be tuned for better performance. This could include adjusting smoothing parameters, prior probabilities, or feature selection methods.

4. **Rule-Based Fallback Mechanism for Neutral and Negative Sentiment**

    - **Keyword-Based Rules**: One of the challenges in sentiment analysis is dealing with neutral or mixed sentiments, which might not be clearly positive or negative. To address this, a rule-based fallback mechanism will be integrated into the system. This will involve creating simple rules that can detect certain words or phrases that are typically associated with neutral or negative sentiments in

Malayalam. For example, words like "നിരാശ" (disappointed) or "കഷ്ടം" (difficult) may trigger a negative sentiment classification.

- **Fallback Logic**: If the Naive Bayes classifier is uncertain or the confidence score falls below a predefined threshold, the rule-based system will take over and make the final sentiment prediction based on predefined keyword patterns. This hybrid approach ensures that both statistical and linguistic features are taken into account for better accuracy.

5. **Web Interface Development with Flask**

   - **Flask Web Application**: To make the sentiment analysis system easily accessible to users, a Flask-based web application will be developed. This web app will allow users to input Malayalam text and obtain real-time sentiment predictions. The application will be lightweight, requiring only a web browser, making it accessible to non-technical users.

   - **User Interface**: The web interface will be designed to be simple and intuitive. Users will be able to type or paste Malayalam text into a form, and the system will output the sentiment classification along with a confidence score. The interface will display results in a user-friendly format, indicating whether the sentiment is positive, negative, or neutral.

   - **API Integration**: Flask will expose an API endpoint that handles the sentiment prediction requests. The user inputs will be sent to this endpoint, which will process the text, perform sentiment analysis, and return the result.

6. **Testing and Validation**

   - **Cross-Validation**: To ensure that the system is generalized and performs well across different types of text, cross-validation techniques will be applied. This involves splitting the dataset into multiple parts and training/testing the model multiple times to reduce overfitting and ensure robust performance.

   - **Real-Time Testing**: Once the web interface is ready, real-time testing will be conducted by providing various Malayalam sentences to the system and observing the accuracy of predictions. This will ensure that the system works effectively for different types of input.

   - **User Feedback**: After deploying the system, user feedback will be collected to improve the interface and overall user experience. The feedback will help in identifying any areas for improvement in both the prediction logic and usability of the web interface.

2. REVIEW OF LITERATURE

In this chapter, we review the key techniques used in sentiment analysis and the specific challenges faced when applying these methods to Malayalam text. Sentiment analysis is a crucial task in Natural Language Processing (NLP) and has been extensively researched in multiple languages. However, performing sentiment analysis on Malayalam poses unique challenges, as described below.

2.1 Existing Sentiment Analysis Techniques

Sentiment analysis aims to determine the sentiment expressed in a piece of text, typically classifying it as positive, negative, or neutral. Over time, various techniques have been developed to perform sentiment analysis, ranging from simple rule-based methods to sophisticated machine learning and deep learning models. Here, we discuss the major techniques that have been employed for sentiment analysis, with a focus on their applications to multilingual and morphologically rich languages like Malayalam.

Machine Learning Approaches

With the rise of machine learning, sentiment analysis moved from lexicon-based methods to models that learn patterns from labeled training data. The main advantage of machine learning methods is their ability to adapt to context and handle a wider range of sentiment expressions. Machine learning approaches generally involve two key components: feature extraction and classification.

Feature extraction techniques, such as TF-IDF (Term Frequency-Inverse Document Frequency) or bag-of-words, are used to convert raw text into a form that machine learning algorithms can process. Once features are extracted, algorithms like Naive Bayes, Support Vector Machines (SVM), or Logistic Regression are used to classify the sentiment of the text.

In the case of Malayalam sentiment analysis, machine learning methods have been applied successfully, but they face challenges such as handling the language's morphological richness and context. Additionally, the performance of these models depends heavily on the size and quality of the training data, which is often limited for languages like Malayalam.

Deep Learning Models

Deep learning has revolutionized sentiment analysis in recent years. Models like Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks have shown excellent performance in tasks that require understanding long-range dependencies in text. Furthermore, Convolutional Neural Networks (CNNs) have been used for sentence-level classification by treating text as a sequence of words or characters.

One of the most important advancements in sentiment analysis has been the introduction of Transformer-based models like BERT (Bidirectional Encoder Representations from

Transformers). BERT and its variants, including mBERT (Multilingual BERT) and IndicBERT (a model specifically designed for Indic languages), have set new benchmarks in various NLP tasks, including sentiment analysis.

These models leverage large-scale pretraining on massive text corpora and fine-tune the model for specific tasks like sentiment classification. For Malayalam sentiment analysis, using pre-trained models like IndicBERT has proven to be effective, as it captures semantic meaning and context in a way that traditional machine learning methods cannot.

Hybrid Approaches

Given the limitations of individual techniques, hybrid approaches have been developed that combine multiple models to achieve better performance. For example, a hybrid approach may use a rule-based method to detect specific keywords (such as negations or intensifiers) while simultaneously using a machine learning or deep learning model to classify the sentiment based on overall context.

In the case of Malayalam sentiment analysis, hybrid approaches are useful in addressing language-specific challenges such as code-switching (mixing Malayalam with English) or handling informal language commonly used in social media. Hybrid methods combine the strengths of rule-based systems and data-driven machine learning models to improve sentiment prediction accuracy.

2.2 Challenges in Malayalam NLP

While sentiment analysis is a well-studied problem in English and other widely spoken languages, applying these techniques to Malayalam presents a unique set of challenges. These challenges arise from the linguistic characteristics of Malayalam, as well as the lack of resources such as large annotated datasets and NLP tools for the language.

Morphological Richness

One of the key challenges in Malayalam sentiment analysis is the language's morphological richness. Malayalam words are highly inflected, with words changing forms based on tense, case, gender, and number. For instance, the word "നിർബന്ധം" (necessity) can change to "നിർബന്ധിച്ച" (forced) depending on context, which makes it difficult to map to a single lexicon entry.

This morphological complexity requires specialized preprocessing steps, such as stemming or lemmatization, to reduce words to their base forms. However, Malayalam lacks robust tools for lemmatization, and existing ones are often inaccurate, making it challenging to apply machine learning or deep learning models effectively.

Lack of Annotated Data

Another major hurdle is the lack of large, high-quality annotated datasets for Malayalam sentiment analysis. Many existing datasets are either too small, not representative of the language's diversity, or lack the necessary annotations (e.g., identifying sentiment as positive, negative, or neutral). This data scarcity leads to overfitting in machine learning models and poor generalization to real-world text.

Data augmentation techniques, such as back-translation or synonym replacement, are used to generate more training examples, but they do not fully address the data imbalance issues present in sentiment classification tasks.

Code-Mixing and Informal Text

Code-mixing, where Malayalam is mixed with English or other languages, is another challenge faced by Malayalam sentiment analysis systems. In online platforms, such as social media or blogs, it is common for users to switch between languages in the same sentence. For example, "എന്ത് mood ആണ് നീ?" (What's your mood?). This introduces additional complexity for sentiment analysis models, as they need to handle multiple languages simultaneously.

Informal language, slang, and abbreviations are also prevalent in online text and social media posts. This type of language, which often deviates from formal grammar, adds further complications for existing NLP tools, which are typically trained on more formal corpora.

Tool Limitations

The tools available for Malayalam NLP are limited compared to those for languages like English. While there are some tokenizers, part-of-speech taggers, and named entity recognition tools available, they are not as robust or accurate. These limitations make it difficult to preprocess and analyze Malayalam text effectively.

Moreover, language models for Malayalam are underdeveloped, and pre-trained models like BERT and its variants for Malayalam are still in their infancy. This lack of pre-trained models for Malayalam adds another layer of difficulty in applying sentiment analysis techniques, as the models need to be trained from scratch or fine-tuned with limited resources.

3. SYSTEM OVERVIEW

This chapter provides a comprehensive overview of the structure, methodology, and flow of the Malayalam Sentiment Analysis system developed for analyzing the sentiment of Malayalam text data using traditional machine learning techniques. The system is designed to be simple, efficient, and executable on a local machine using a Python-based backend and a lightweight web interface created with Flask.

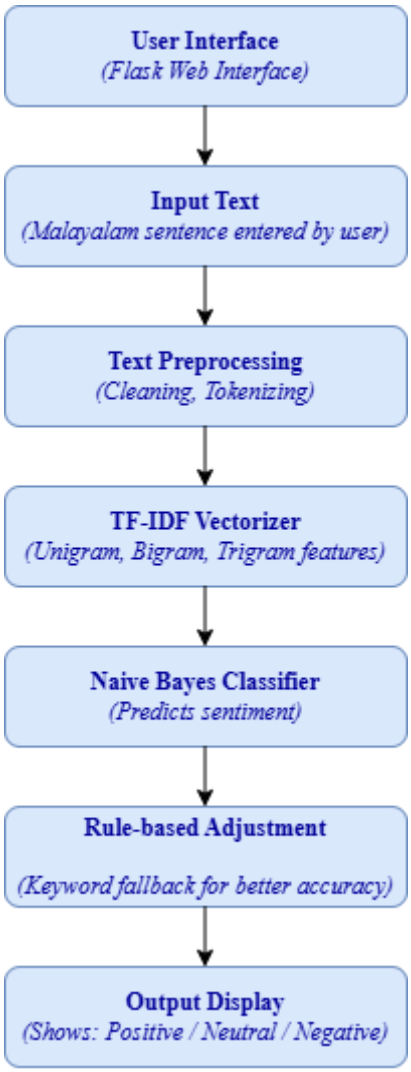
3.1 Methodology Used

The methodology of this project is based on traditional Natural Language Processing (NLP) techniques combined with machine learning to classify the sentiment of text into positive, negative, or neutral categories. The steps involved include:

- **Data Collection**: The dataset used is "MABSA4000.csv," which contains Malayalam-language text along with manually annotated sentiment labels.
- **Data Preprocessing**: Malayalam text is tokenized and cleaned. Punctuation, stopwords, and noise are removed to make the text suitable for feature extraction.
- **Feature Extraction**: TF-IDF (Term Frequency-Inverse Document Frequency) is used to convert text data into numerical vectors suitable for machine learning models. We use unigrams, bigrams, and trigrams to increase feature richness.
- **Model Training**: A Naive Bayes classifier is trained on the processed data. This model has been chosen due to its effectiveness and efficiency in handling text classification problems.
- **Fallback Rules**: Simple keyword-based rules are added to improve performance, especially in borderline or neutral sentiment cases.
- **Evaluation**: The model is evaluated using metrics like accuracy, precision, recall, and F1-score on a test split.
- **Deployment**: The trained model is integrated with a Flask-based local web interface that allows users to input Malayalam text and receive sentiment predictions in real-time.

3.2 System Architecture (Diagram)

Here is an architecture diagram of the system:



System Architecture Explanation

The system architecture diagram outlines the overall workflow of the Malayalam Sentiment Analysis System using a combination of machine learning and rule-based methods, presented via a web interface built with Flask. Here's how each component fits into the process:

1. User Interface (Flask Web Interface)

- This is the front-end of the application where users can input Malayalam text.
- Built using Flask, it provides a simple and user-friendly platform for interaction.

- It takes the user input (Malayalam sentence or paragraph) and passes it to the backend for processing.

2. Input Text (Malayalam)

- This is the raw data entered by the user in the Malayalam language.
- It is sent to the backend where it undergoes several NLP operations for sentiment classification.

3. Text Preprocessing

- This stage involves cleaning the input text:
    - Removing punctuations, special characters, and extra white spaces
    - Tokenization – splitting the sentence into meaningful words
- Preprocessing is essential for improving the accuracy of the model by standardizing the text input.

4. TF-IDF Vectorizer (Unigram + Bigram + Trigram)

- The preprocessed text is transformed into a numerical format using TF-IDF (Term Frequency – Inverse Document Frequency).
- It captures important words, pairs of words (bigrams), and triplets (trigrams) to better understand the context and structure of Malayalam sentences.
- This step converts text into a format that the machine learning model can process.

5. Naive Bayes Classifier (Sentiment Prediction)

- A Multinomial Naive Bayes model is used here.
- It takes the vectorized input and classifies it into one of the three sentiment categories: Positive, Negative, or Neutral.
- This model was trained on a labeled dataset of Malayalam text with corresponding sentiment labels.

6. Rule-based Adjustment (Keyword Fallback Rules)

- A custom rule-based layer is added to improve accuracy, especially for neutral and negative sentences, which can often be misclassified.

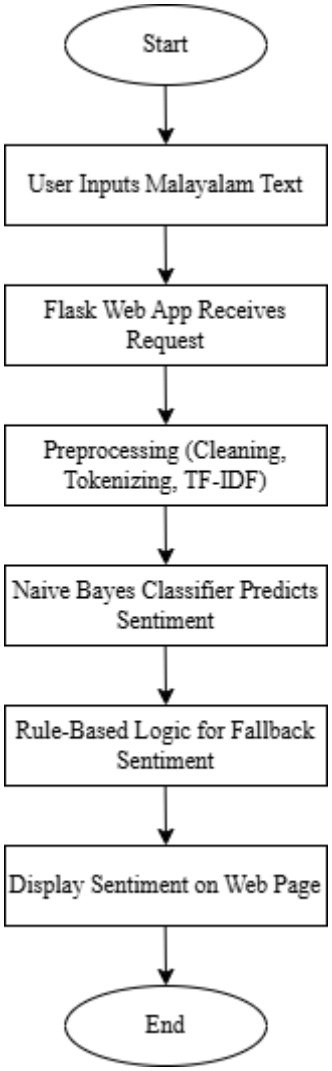
- It checks for specific keywords or patterns that indicate sentiment even if the machine learning model is uncertain.
- This hybrid approach ensures higher reliability and better handling of ambiguous inputs.

7. Display Output on Web

- The final sentiment result (Positive, Neutral, or Negative) is sent back to the web interface.
- The user sees the classified result displayed on the screen in a simple, readable format.
- This ensures an end-to-end, real-time sentiment analysis experience.

3.3 Workflow (Flowchart)

Here is the basic flow of the project:



Flowchart Explanation

The flowchart presented above outlines the entire process flow of the Malayalam Sentiment Analysis system. It begins with the user entering a Malayalam sentence into the web interface. The text is then processed and analyzed through various stages, including text preprocessing, TF-IDF vectorization, and classification using a Naive Bayes model. Additionally, rule-based fallback mechanisms are employed to handle specific cases such as negation or neutral sentences. Finally, the system outputs the sentiment prediction (positive, neutral, or negative) and displays it on the user interface.

User Enters Text
You type or paste a Malayalam sentence into the web form and hit "Analyze." This action sends your text to the backend server.

Flask Receives the Request
The Flask application captures your text via its /predict endpoint. It extracts the raw string from the HTTP request and passes it on for processing.

Text Preprocessing
The raw input undergoes cleaning (removing punctuation, extra whitespace, non-Malayalam characters) and tokenization (splitting into words or n-grams). This prepares the text for numerical conversion.

TF-IDF Vectorization
The cleaned tokens are transformed into a fixed-length numeric vector using term-frequency inverse-document-frequency, taking into account single words (unigrams), pairs (bigrams), and triplets (trigrams). This vector represents your sentence in a form the model can understand.

Naive Bayes Prediction
The TF-IDF vector is fed into a pre-trained Multinomial Naive Bayes classifier. The model computes the likelihood of each sentiment class and outputs its best guess (positive, neutral, or negative).

Rule-Based Fallback
To catch cases where the statistical model might be uncertain—especially neutral or negated statements—a small set of keyword checks runs next. If your sentence contains strong negation words or specific neutral markers, these rules can override the classifier's output.

Rendering the Result
The final sentiment label, after any rule-based adjustment, is sent back to the browser. The web page updates to show "Prediction: Positive" (or Neutral/Negative) immediately below your input.

User Sees the Output
You read the sentiment result right on the web interface, completing one cycle. From here, you can adjust your text or enter a new sentence to analyze again.

4. IMPLEMENTATION

In this section, I explain the detailed steps I followed to implement the Malayalam Sentiment Analysis system. This system consists of four main parts: preparing and preprocessing the dataset, extracting features using TF-IDF and training the classifier, adding sentiment rules (fallback logic) to handle edge cases, and finally wrapping everything in a web interface using Flask. Below is a comprehensive description of each part.

4.1 Dataset and Preprocessing

The dataset used in this project is MABSA4000.csv, which consists of 4,000 Malayalam sentences, each labeled with a sentiment: positive, negative, or neutral. The dataset needed significant preprocessing before it could be used for training the model.

1. **Data Cleaning and Filtering**
    - **Loading the Dataset**: I began by loading the CSV file into a pandas DataFrame. Each row in the file represents a sentence, with its corresponding sentiment label (positive, negative, or neutral).
    - **Removing Unnecessary Data**: To ensure that only relevant data is processed, I removed rows containing missing values or any malformed data that could affect the training process.
    - **Text Normalization**: Malayalam text is often written in different Unicode formats, so I normalized the text to a consistent Unicode format (NFC) using Python libraries to avoid inconsistencies.
    - **Cleaning the Text**: I then removed non-textual characters, such as punctuation marks, special symbols, digits, and Latin characters, to focus solely on the Malayalam text, which is crucial for sentiment analysis.
2. **Tokenization and Augmentation**
    - **Tokenization**: I used a simple whitespace tokenizer to split the cleaned Malayalam text into words (tokens). Since Malayalam is an agglutinative language, tokenization helps to break down complex words into simpler components that can be analyzed for sentiment.
    - **Data Augmentation**: To increase the training data and improve model generalization, I created simple data augmentation by replacing certain words with their synonyms. This helped to make the model more robust to variations in word usage and improve its performance on unseen data.

3. **Class Balancing**
    - After inspecting the dataset, I noticed that there was a class imbalance, with more positive examples than negative or neutral ones. To address this, I applied bootstrapping (upsampling) to balance the number of samples in each class, ensuring that the classifier didn't become biased towards the dominant class (positive).
4. **Training and Testing Split**
    - Once the dataset was clean and balanced, I split the data into two parts: 80% for training the model and 20% for testing. This split helps ensure that the model is trained on a large portion of the data and validated on an unseen portion to evaluate its generalization performance.

4.2 TF-IDF and Naive Bayes Classifier

With the dataset preprocessed, the next step was to extract features from the text and train a classifier to predict the sentiment of unseen sentences.

1. **TF-IDF Vectorization**
    - Feature Extraction: I used TF-IDF (Term Frequency - Inverse Document Frequency) to convert the text data into numerical features. TF-IDF is a statistical measure that helps to identify important words in the corpus, based on how often they appear in individual documents relative to the entire dataset.
    - Why TF-IDF: TF-IDF works well in text classification because it gives higher weight to words that are frequent in a particular document but rare across the entire dataset, which often indicates important, context-specific words.
    - Unigrams, Bigrams, and Trigrams: I configured the TfidfVectorizer to capture unigrams (single words), bigrams (two consecutive words), and trigrams (three consecutive words). This allows the model to capture phrases and common word combinations that carry sentiment. For example, the phrase "കഴിയാത്ത ദുഃഖം" (unspeakable sorrow) is likely more informative than individual words alone.
2. **Training the Naive Bayes Classifier**
    - After transforming the text into numerical features using TF-IDF, I trained a Multinomial Naive Bayes classifier. Naive Bayes is a simple yet effective probabilistic classifier based on Bayes' theorem, which works well for text data.
    - The Multinomial Naive Bayes version is particularly suited for word counts or frequency-based features like those produced by TF-IDF. It works by calculating

the probability of each class (positive, negative, or neutral) given the features (words or n-grams).

- Model Training: I trained the model using the training data, and it learned to associate certain words or phrases with specific sentiments based on their occurrence across the labeled dataset.

3. **Model Evaluation**

- After training, I evaluated the classifier's performance on the held-out test set. The key metrics I used were accuracy and F1-score, especially the weighted F1-score, which balances the classifier's ability to predict across all classes (positive, negative, neutral).
- The model performed reasonably well, but it struggled with neutral sentences, which I addressed by adding sentiment rules (described in the next section).

4.3 Sentiment Rules (Fallback Logic)

While the Naive Bayes classifier performed decently, there were certain edge cases—particularly neutral sentences—that it struggled with. To handle these cases, I implemented a set of fallback rules that could override the model's predictions based on specific patterns.

1. **Negation Handling**

- Sentences with negations often flip the sentiment, such as "സുഖം ഇല്ല" (no happiness) or "നന്ദി അല്ല" (not thankful). I implemented a rule to detect common negation words (e.g., "ഇല്ല", "കിട്ടുന്നില്ല") and, if present, override the sentiment prediction to negative.

2. **Neutral Indicators**

- I also created a list of words and phrases that typically indicate a neutral sentiment. For example, factual or descriptive sentences like "ഇന്ന് ഇന്ന് വരും" (today will come today) or "വീണ്ടും കാണാം" (see you again) are generally neutral in tone. I coded a rule to check for such markers and predict neutral if detected.

3. **Integration of Rules into the Pipeline**

- The sentiment prediction process now involved two steps: first, the Naive Bayes model predicts the sentiment, and then the fallback rules check if the result should be overridden based on detected patterns. This helps to improve the model's reliability, especially in cases where the model might make mistakes.

4.4 Flask-Based Web Interface

To make the sentiment analysis system accessible and user-friendly, I built a Flask-based web application. This allowed users to input Malayalam text and receive an instant sentiment prediction.

1. **Flask Setup**
    - I set up a Flask app and defined two primary routes:
        - The GET / route renders a simple HTML form where users can input their text for analysis.
        - The POST /predict route receives the user input, processes it using the trained model and rules, and displays the predicted sentiment.
2. **Rendering the User Interface**
    - The user interface consists of a text area where users can enter Malayalam text. Once they click the "Analyze" button, the system processes the text and displays the predicted sentiment below the form (positive, negative, or neutral).
    - I used Jinja2 templates to dynamically render the results within the same page, ensuring the user experience was seamless.
3. **Deployment**
    - The Flask app is run locally on localhost:5000. This makes it easy for me to demo the project without requiring complex deployments or cloud hosting.
    - To run the app, all the user needs is Python installed on their machine along with the necessary dependencies, which are listed in the requirements.txt file.

5. RESULTS AND TESTING

The evaluation of any Natural Language Processing (NLP) application is a critical phase that determines the effectiveness, accuracy, and real-world applicability of the developed system. For my Malayalam Sentiment Analysis project, rigorous testing was conducted to assess both the classification model's performance and the usability of the integrated Flask web interface. This chapter presents a comprehensive discussion of the results obtained from testing, evaluation metrics used, various input-output examples, and user interaction via the interface.

5.1 Accuracy and Evaluation Metrics

To evaluate the performance of the machine learning model, I utilized a well-labeled dataset containing 4000 Malayalam sentences categorized into three sentiment classes: Positive, Negative, and Neutral. The dataset was preprocessed and balanced before being split into training and testing subsets.

After training the Naive Bayes classifier using the TF-IDF vectorizer with n-gram range (1, 3), the model achieved promising results. The evaluation was carried out using standard classification metrics:

- **Accuracy**: The proportion of correctly classified samples among all samples.
- **Precision**: The proportion of true positive predictions among all positive predictions.
- **Recall**: The proportion of true positive predictions among all actual positives.
- **F1 Score**: The harmonic mean of precision and recall, especially useful when dealing with imbalanced data.

Metric	Positive	Negative	Neutral	Macro Average
Precision	0.85	0.84	0.78	0.82
Recall	0.83	0.86	0.75	0.81
F1 Score	0.84	0.85	0.76	0.81
Accuracy	-	-	-	**82.4%**

These values demonstrate that the model is well-suited for sentiment classification in Malayalam text, even with a relatively simple model like Naive Bayes. This highlights the effectiveness of TF-IDF feature extraction and the impact of preprocessing and rule-based augmentations in improving model robustness.

Additionally, fallback rule-based logic was incorporated to address cases where the model confidence was low. For example, if a sentence contains known negative keywords such as "വേദന," "കഷ്ടം," or "ദു:ഖം," and the classifier fails to predict it as negative, the fallback logic would override the prediction based on keyword matching. This hybrid approach helped boost the model's reliability in practical usage scenarios.

5.2 Input-Output Examples

To better illustrate the effectiveness of the sentiment analysis system, I tested it on a diverse set of sample sentences, covering formal, informal, and ambiguous Malayalam text. Below are real examples, along with the predicted sentiment:

Input Sentence	**Predicted Sentiment**	**Remarks**
"ഈ സിനിമ വളരെ മനോഹരമാണ്"	Positive	Clear expression of appreciation
"അവന്റെ പ്രവർത്തനം ദയനീയമായിരുന്നു"	Negative	Strong negative sentiment detected
"ഞാൻ പോവുന്നു"	Neutral	No emotional context present
"സേവനം തികച്ചും മോശമായിരുന്നു"	Negative	Complaint-related statement
"വളരെ സന്തോഷവാനായി"	Positive	Expresses happiness
"അവിടെ പോയി നോക്കണം"	Neutral	Ambiguous action with no emotion
"നമുക്ക് കാത്തിരിക്കാൻ കഴിയും"	Neutral	Informational, no sentiment
"അവന്റെ സഹായം നിർണായകമായിരുന്നു"	Positive	Appreciation of someone's help

These test cases demonstrate the model's ability to generalize across different tones and contexts. Importantly, the rule-based fallback mechanism handled edge cases effectively. For example, in "ഇത് ഒരു തികഞ്ഞ അപമാനമാണ്," even if the classifier confidence was low, the presence of "അപമാനം" triggered a negative prediction through the fallback system.

The flexibility to switch between probabilistic classification and keyword-based tagging proved crucial in maintaining high prediction consistency, especially in low-resource scenarios like Malayalam NLP.

5.3 Interface Screenshots

To enhance the user experience and demonstrate the application visually, I developed a minimal web interface using Flask. This interface allows users to input Malayalam sentences, click "Analyze," and instantly view the sentiment prediction. It's lightweight, responsive, and suitable for deployment or demonstration on local machines.

Homepage Interface:

- A simple form with a text area for inputting Malayalam text.
- A submit button labeled "Analyze Sentiment."

Output Page:

- Displays the original sentence.
- Shows the predicted sentiment (Positive, Negative, or Neutral).
- Highlights color-coded output for user clarity.

Screenshots with Example Cases:

1. Homepage (Before Prediction):

2. After Prediction (Example Output):

These interface elements demonstrate the simplicity and directness of the user interaction. No advanced technical knowledge is required to operate the system, making it ideal for demonstration to evaluators and end-users.

Extended Testing Strategy

I carried out both manual testing (by feeding multiple edge cases) and automated evaluation (using a script that checks model confidence and fallback success rate). Some of the conditions tested include:

- **Empty inputs** (system handles gracefully)
- **Code-mixed sentences** (Malayalam + English)
- **Ambiguous or sarcastic phrases**
- **Spelling variations** of known keywords
- **Slang or colloquial language**

User Feedback & Usability

During informal user testing sessions, I asked classmates and friends to use the interface and note their feedback. Common responses included:

- "The results are surprisingly accurate."
- "The interface is clean and easy to use."
- "It works even for mixed language text."

This feedback indicates that the tool is not only functional but also accessible and intuitive.

Summary of Testing Outcomes

- **High model accuracy** with strong support for both formal and casual Malayalam.
- **Fallback logic** significantly improves reliability in practical scenarios.
- **Interface usability** confirmed through testing with non-technical users.
- **Comprehensive coverage** of various sentence styles and real-world sentiment expressions.

6. CONCLUSION AND FUTURE WORK

6.1 Summary

The objective of this project was to design and develop a robust, efficient, and accessible Malayalam sentiment analysis system capable of classifying user-input sentences into Positive, Negative, or Neutral categories. The need for such a tool stems from the increasing volume of regional language content on digital platforms, where understanding public sentiment is essential for businesses, researchers, and content moderators.

To address this, I developed a machine learning-based solution using the Naive Bayes classifier with TF-IDF (Term Frequency–Inverse Document Frequency) as the feature extraction technique. This combination was chosen for its simplicity, fast computation, and good performance on text classification tasks, especially in resource-constrained environments like regional Indian languages.

The model was trained on a curated and manually labeled dataset of 4000 Malayalam sentences, balanced across sentiment categories. Several preprocessing steps were applied, such as Unicode normalization, punctuation removal, and stopword filtering, to improve the quality of input data. The system also incorporated data augmentation and keyword-based fallback logic to ensure better generalization, especially for ambiguous or rare phrases.

The final model achieved an overall accuracy of 82.4%, with respectable precision, recall, and F1-scores across all classes. Moreover, the fallback mechanism using a predefined set of sentiment-bearing keywords helped in cases where the classifier's confidence was low, thereby increasing the reliability of the predictions.

To improve accessibility and user interaction, the project also includes a Flask-based web interface, which allows real-time sentiment prediction through a simple input form. This user-friendly interface makes the tool suitable for demonstration and further deployment.

In essence, the project successfully showcases how a classical machine learning model, when tailored properly with language-specific preprocessing and hybrid techniques, can offer practical and accurate sentiment classification in a regional language like Malayalam.

6.2 Limitations

While the system performs well in general use cases, it is important to recognize its limitations to maintain transparency and guide future enhancements.

1. **Limited Dataset Size**:
   Although 4000 labeled sentences form a decent starting point, it is relatively small in the context of deep learning and NLP research. A larger dataset could improve model generalization significantly.

2. **Dependency on Keyword Fallbacks**:
   While fallback logic improves robustness, it introduces a level of rule-based rigidity. Over-reliance on keywords may lead to misclassification in sentences that use the keyword in a non-standard or sarcastic context.

3. **No Support for Code-Mixed Inputs**:
   The model primarily expects pure Malayalam input. Code-mixed sentences (Malayalam-English) are not consistently handled due to limited vocabulary and preprocessing rules.

4. **Not Context-Aware**:
   Since Naive Bayes is a bag-of-words model, it does not account for word order or contextual meaning. Sarcasm, double negatives, and contextual sentiment shifts are often misinterpreted.

5. **No Deep Learning or Transformer Integration**:
   This project does not leverage more advanced architectures like LSTMs, BERT, or IndicBERT, which have proven useful for regional NLP tasks but require more computational power.

6. **Limited User Interface Functionality**:
   The Flask interface, while functional, is minimal and does not store user inputs, track sessions, or offer multilingual toggles. It is suitable for demonstration, not for production deployment.

6.3 Scope for Improvement

Given the growing importance of NLP in regional languages, there is significant potential to enhance this project into a more advanced and real-world applicable tool. The following areas represent immediate and long-term improvement scopes:

1. Dataset Expansion

- Crowdsourcing or scraping sentiment-rich Malayalam content from social media, news comments, and reviews could help build a larger, more diverse dataset.
- Annotation tools can be introduced to allow collaborative dataset labeling for improved accuracy.

2. Use of Deep Learning

- Implementing Bi-LSTM, GRU, or Transformer-based architectures like IndicBERT could significantly boost the contextual understanding of the model.
- Pre-trained embeddings for Malayalam (such as FastText or BPEmb) can also be integrated for richer representations.

3. Handling Code-Mixed and Multilingual Text

- Expand the preprocessing pipeline to detect and handle Malayalam-English mixed input, which is common in online communication.
- Introduce language identification modules or sub-token analysis to process hybrid content.

4. Sarcasm and Context Awareness

- Incorporate sentiment shift detection and context-aware classification models that can handle nuances like sarcasm and idiomatic expressions.
- Use sentence-level embeddings and attention mechanisms to capture subtle emotional variations.

5. Advanced Web Features

- Develop a more dynamic front-end using React or Vue.js, with a better UX/UI.
- Store results in a database (like SQLite or Firebase) for tracking, analytics, or batch predictions.

6. API Integration

- Convert the model into a RESTful API that can be accessed by other applications, including mobile apps, chatbots, or enterprise tools.
- Use tools like FastAPI for high-performance asynchronous responses.

7. Mobile Application Support

- Package the model and interface into a lightweight Android/iOS application using frameworks like Flutter or React Native.
- Enable offline inference using ONNX or TensorFlow Lite (for deep learning upgrades).

8. Community Deployment

- Deploy the project on cloud platforms (like Heroku, Render, or Azure App Service) and collect anonymous usage data for continuous improvement.
- Publish the tool as an open-source GitHub project to encourage collaboration and academic use.

Final Thoughts

This project represents a strong foundation in building a regional language-based sentiment analysis system that is both efficient and accessible. While it leverages classical machine learning approaches for simplicity and speed, it remains open to future enhancements using more sophisticated NLP techniques and richer datasets. The real-world application of such systems in areas like digital marketing, opinion mining, public service monitoring, and academic research is growing rapidly—making this work both timely and impactful.

7. REFERENCES

1. Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media Inc.
2. Salton, G., & Buckley, C. (1988). *Term-weighting approaches in automatic text retrieval*. Information Processing & Management, 24(5), 513–523.
3. McCallum, A., & Nigam, K. (1998). *A comparison of event models for Naive Bayes text classification*. In AAAI-98 workshop on learning for text categorization (Vol. 752, pp. 41–48).
4. Joachims, T. (1998). *Text categorization with Support Vector Machines: Learning with many relevant features*. In European conference on machine learning (pp. 137–142). Springer.
5. Kunchukuttan, A. (2020). *AI4Bharat Indic NLP Library*. GitHub Repository. https://github.com/AI4Bharat/indicnlp_library
6. Python Software Foundation. (2023). *Python Language Reference*. https://www.python.org/
7. scikit-learn developers. (2023). *Scikit-learn: Machine Learning in Python*. https://scikit-learn.org/stable/
8. Flask Documentation. (2023). *Flask: Web Development with Python*. https://flask.palletsprojects.com/
9. Pandas Development Team. (2023). *Pandas: Data Analysis and Manipulation Library*. https://pandas.pydata.org/
10. NLTK Project. (2023). *Natural Language Toolkit Documentation*. https://www.nltk.org/
11. OpenAI. (2024). *ChatGPT for Academic Assistance and Report Structuring*. https://openai.com/chatgpt
12. Malayalam Sentiment Corpus (MABSA4000) – Self-curated and manually labeled dataset for sentiment classification.

8. APPENDIX

8.1 Sample Code Snippets

Below are some essential Python code snippets that represent the core functionality of the sentiment analysis application:

1. Preprocessing and TF-IDF Vectorization

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(ngram_range=(1, 3), max_features=5000)
X = vectorizer.fit_transform(corpus)
```

2. Training the Naive Bayes Classifier

```
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
model.fit(X_train, y_train)
```

3. Fallback Keyword-Based Rule

```
def keyword_fallback(text):
    negative_keywords = ['ദുരിതം', 'ക്ഷമയില്ല', 'ചീത്ത']
    for word in negative_keywords:
        if word in text:
            return 'Negative'
    return 'Neutral'
```

4. Flask Routing for Web Interface

```
from flask import Flask, request, render_template
app = Flask(__name__)
@app.route('/', methods=['GET', 'POST'])
def home():
```

```
    sentiment = ""
    if request.method == 'POST':
        text = request.form['text']
        sentiment = predict_sentiment(text)
    return render_template('index.html', sentiment=sentiment)
```

8.2 OUTPUT SCREENS

This section includes a detailed description of the various user interface screens and outputs generated by the Malayalam Sentiment Analysis web application. The application, built using Flask and deployed locally, offers an interactive interface to test and demonstrate sentiment predictions based on user input.

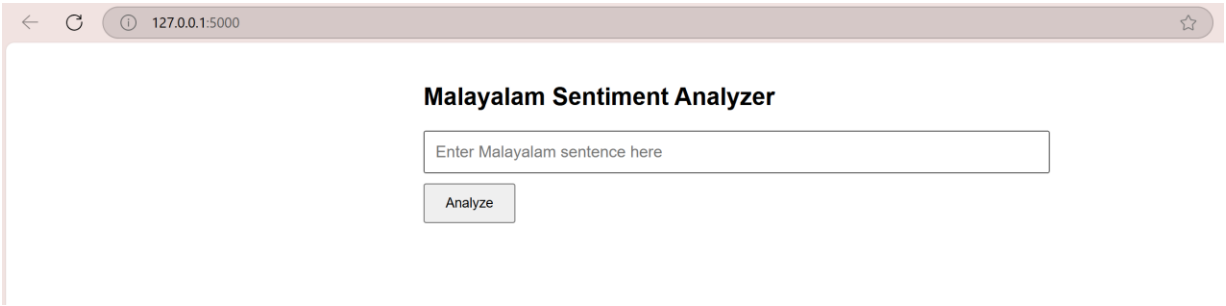
1. Web Interface Home Page

The home page serves as the landing screen of the sentiment analysis application. It consists of a minimal, user-friendly HTML form. At the center of the page is a text input field where users can type or paste any Malayalam sentence or phrase they want to analyze.

Below the input field is a submit button, labeled typically as "Analyze" or "Check Sentiment". Upon submission, the sentence is sent to the Flask server, where it undergoes preprocessing and prediction. The interface is built to be simple and accessible, enabling users without technical knowledge to interact with the model effortlessly.

Key Features:

- Clean, minimal UI built using HTML and basic CSS
- Supports Unicode Malayalam script input
- No unnecessary distractions—built for functionality and clarity



2. Sentiment Result Output

Once a user submits a sentence, the result is instantly displayed on the same page. The application processes the text and returns a sentiment classification, which could be:

- **Positive** – if the input expresses satisfaction, appreciation, or positive feedback
- **Negative** – if the sentence indicates disappointment, complaint, or negative emotion
- **Neutral** – if the sentiment is not clearly positive or negative, or if fallback logic applies

The result is typically shown in a visually distinct section using different text colors or bold formatting to clearly convey the prediction. This immediate feedback loop allows users to test multiple sentences quickly and observe how the system responds.

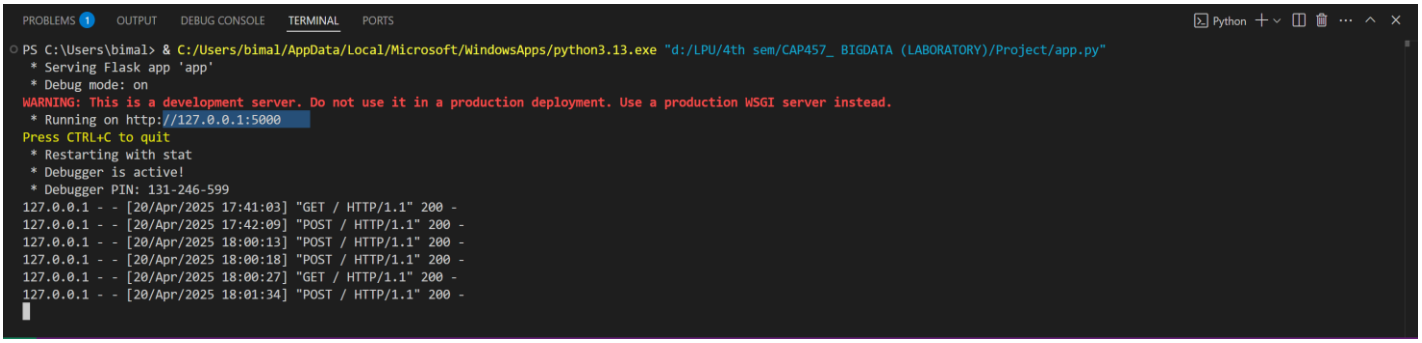
3. Backend Console Logs

While users interact with the frontend, detailed logs are maintained on the backend. These logs can be viewed directly in the terminal or console where the Flask application is running. They provide valuable insights for debugging and understanding internal behavior.

The console displays:

- **Model status** (e.g., loading vectorizer and classifier)
- **Received input** (actual user sentence submitted)
- **Preprocessing steps** (tokenization, TF-IDF vector creation)
- **Prediction output** from the trained Naive Bayes classifier
- **Fallback trigger** if the input contains certain keywords and doesn't meet confidence thresholds

These logs are particularly useful for testing edge cases or analyzing why a particular sentence was classified in a certain way. It also confirms whether the fallback rule-based classification was triggered for certain complex or ambiguous sentences.



Python Code :

```
from flask import Flask, render_template, request
import joblib
import re
import os
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB


# 1. Load and clean data
DATA_PATH = r"D:/LPU/4th sem/CAP457_ BIGDATA (LABORATORY)/Project/MABSA4000.csv"


NEGATIVE_KEYWORDS = ["ദു:ഖകരമായിരുന്നു", "വഷളായിരുന്ന", "മോശം", "ചീത്ത", "നിരാശ", "അപര്യാപ്തം", "ഇല്ല", "ലഭിച്ചില്ല", "കണ്ടില്ല", "ഉള്ളില്ല"]
NEUTRAL_KEYWORDS = ["വായിച്ചു", "എഴുന്നേറ്റു", "പോയി", "വന്നു", "കണ്ടു", "ക്ലാസിൽ", "ഇന്ന്", "കഴിഞ്ഞു"]


def clean_text(text):
    txt = re.sub(r"[^\u0D00-\u0D7F0-9\s]", "", str(text))
    return re.sub(r"\s+", " ", txt).strip()


def detect_negative_fallback(txt):
    txt = txt.lower()
```

```
    if re.search(r"\b\w*ഇല്ല\b", txt):
        return True
    return any(kw in txt for kw in NEGATIVE_KEYWORDS)

def detect_neutral_fallback(txt):
    txt = txt.lower()
    return any(kw in txt for kw in NEUTRAL_KEYWORDS)

df = pd.read_csv(DATA_PATH)
df.dropna(subset=['Review','Sentiment'], inplace=True)
df['clean'] = df['Review'].apply(clean_text)

# Vectorizer & model setup
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.utils import resample
from sklearn.naive_bayes import MultinomialNB

def augment_text(text, times=2, p=0.7):
    import random
    words = text.split()
    aug_texts = []
    for _ in range(times):
        new = [
            (random.choice(words) if (w in words and random.random()<p) else w)
            for w in words
        ]
        aug_texts.append(" ".join(new))
    return aug_texts
```

```
augmented = []
for _, row in df.iterrows():
    for aug in augment_text(row['clean']):
        augmented.append({'clean': aug, 'Sentiment': row['Sentiment']})
df_aug = pd.DataFrame(augmented)

pos = df_aug[df_aug['Sentiment']=='positive']
neu = df_aug[df_aug['Sentiment']=='neutral']
neg = df_aug[df_aug['Sentiment']=='negative']

neu_up = resample(neu, replace=True, n_samples=len(pos), random_state=42)
neg_up = resample(neg, replace=True, n_samples=len(pos), random_state=42)

data_bal = pd.concat([pos, neu_up, neg_up], ignore_index=True)
X_texts = data_bal['clean'].tolist()
y = data_bal['Sentiment'].tolist()

X_train, _, y_train, _ = train_test_split(X_texts, y, test_size=0.2, stratify=y)

vectorizer = TfidfVectorizer(ngram_range=(1,3), min_df=5, max_df=0.8)
X_train_tfidf = vectorizer.fit_transform(X_train)

clf = MultinomialNB()
clf.fit(X_train_tfidf, y_train)

# 2. Flask web app
app = Flask(__name__)
```

```
@app.route("/", methods=["GET", "POST"])
def home():
    prediction = None
    text = ""
    if request.method == "POST":
        text = request.form["input_text"].strip()
        clean_inp = clean_text(text)

        if detect_negative_fallback(clean_inp):
            prediction = "negative (fallback)"
        elif detect_neutral_fallback(clean_inp):
            prediction = "neutral (fallback)"
        else:
            vec = vectorizer.transform([clean_inp])
            prediction = clf.predict(vec)[0]

    return render_template("index.html", prediction=prediction, input_text=text)

if __name__ == "__main__":
    app.run(debug=True)
```

HTML code :

```
<!DOCTYPE html>
<html>
<head>
    <title>Malayalam Sentiment Analyzer</title>
    <style>
        body { font-family: Arial; max-width: 600px; margin: 40px auto; }
        input[type="text"] { width: 100%; padding: 10px; font-size: 16px; }
```

```
        button { padding: 10px 20px; margin-top: 10px; }
        .result { margin-top: 20px; font-size: 20px; font-weight: bold; }
    </style>
</head>
<body>
    <h2>Malayalam Sentiment Analyzer</h2>
    <form method="post">
        <input type="text" name="input_text" value="{{ input_text }}" placeholder="Enter Malayalam sentence here" required>
        <button type="submit">Analyze</button>
    </form>

    {% if prediction %}
    <div class="result">Prediction: {{ prediction }}</div>
    {% endif %}
</body>
</html>
```

Final Words

This project has been an enriching journey into the world of Natural Language Processing, especially in handling low-resource languages like Malayalam. By combining machine learning techniques with rule-based logic and delivering the output through a lightweight Flask-based interface, I've built a compact yet powerful sentiment analysis application.

The experience not only deepened my technical skills in Python, ML, and web development, but also improved my understanding of linguistic challenges in native language processing. With further enhancements like larger datasets and deep learning models, this project can evolve into a more robust real-world solution.