VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI - 590018

Mini Project Laboratory 21CSMP67

Report on

SENTIMENTAL ANALYSIS OF RESTAURANT REVIEW

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

Bachelor of Engineering

in

Computer Science and Engineering

Submitted By

SHREYA RAI	**1DT21CS144**
VIKASH JANWA	**1DT21CS179**
SUMEET PATIL	**1DT21CS187**

Under the Guidance of

Prof. Keerthi Mohan
Assistant Professor, Department of CSE

Department of Computer Science and Engineering
DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

(Affiliated to VTU, Belagavi and Approved by AICTE, New Delhi), CE, CSE, ECE, EEE, ISE, ME Courses Accredited by NBA, New Delhi, NAAC A+

Academic Year : 2023-24

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

Udayapura, Kanakapura Road, Bangalore-560082

CERTIFICATE

Certified that the Mini Project titled **"Sentimental Analysis of Restaurant Review"** carried out by **Shreya Rai**, bearing **USN 1DT21CS144, Vikash Janwa,** bearing **USN 1DT21CS179** and **Sumeet Patil** bearing **USN 1DT21CS187,** bonafide students of Dayananda Sagar Academy 0f Technology and Management, is in partial fulfillment for the award of the **BACHELOR OF ENGINEERING** in **Computer Science and Engineering** from Visvesvaraya Technological University, Belagavi during the year 2023- 2024. It is certified that all the corrections/suggestions indicated for Internal Assessment have been incorporated in the report submitted to the department. The Project report has been approved as it satisfies the academic requirements in respect of the Mini Project Work prescribed for the said Degree.

Prof. Keerthi Mohan Assistant Professor Department of CSE DSATM, Bengaluru.	Dr. Kavitha C Professor & HOD Department of CSE DSATM, Bengaluru.	Dr. M. Ravishankar Principal DSATM, Bengaluru.

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

Udayapura, Kanakapura Road, Bangalore-560082

DECLARATION

We, Shreya Rai (1DT21CS144), Vikash Janwa (1DT21CS179) and Sumeet Patil (1DT21CS187)**,** Students of Sixth Semester B.E, Department of Computer Science and Engineering, Dayananda Sagar Academy 0f Technology and Management, Bengaluru, declare that the Mini Project Work titled "**Sentimental analysis of restaurant review**" has been carried out by us and submitted in partial fulfilment of the course requirements for the award of degree in **Bachelor of Engineering** in **Computer Science and Engineering** from **Visvesvaraya Technological University, Belagavi** during the academic year **2023-2024**.

Shreya Rai	1DT21CS144
Vikash Janwa	1DT21CS179
Sumeet Patil	1DT21CS187

Place: Bengaluru

Date:

ABSTRACT

In the realm of customer feedback, sentiment analysis plays a crucial role in interpreting the opinions expressed in textual reviews. This project focuses on sentiment analysis of restaurant reviews to gauge public perception and improve customer service. Leveraging machine learning techniques, the project involves preprocessing and analysing a dataset of restaurant reviews. The dataset is split into training and testing subsets to evaluate model performance. Textual data is vectorized using the CountVectorizer method, transforming reviews into numerical features suitable for machine learning algorithms. Two classification models, Support Vector Classifier (SVC) and Multinomial Naive Bayes (NB), are employed to categorize sentiments into positive and negative classes. A pipeline combining CountVectorizer with MultinomialNB is also utilized for a streamlined approach. Model performance is assessed using accuracy metrics to determine the effectiveness of the models in predicting review sentiments. The results provide insights into customer satisfaction and areas for improvement, demonstrating the potential of sentiment analysis in enhancing the dining experience.

ACKNOWLEDGEMENT

The satisfaction and the euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible.

The constant guidance of these people and encouragement provided, crowned us with success and glory. We take this opportunity to express our gratitude to one and all.

It gives us immense pleasure to present before you our project titled "**Sentimental Analysis of Restaurant Review**".

The joy and satisfaction that accompanies the successful completion of any task would be incomplete without the mention of those who made it possible.

We are glad to express our gratitude towards our prestigious institution **DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT** for providing us with utmost knowledge, encouragement and the maximum facilities in undertaking this project.

We wish to express sincere thanks to our respected Principal **Dr. M Ravi Shankar**, Principal, DSATM for all his support.

We express our deepest gratitude and special thanks to **Dr. Kavitha C,** H.O.D, Dept. of Computer Science Engineering, for all her guidance and encouragement.

We sincerely acknowledge the guidance and constant encouragement of our mini-project guide, **Prof. Keerthi Mohan,** Assistant Professor, Dept. of Computer Science & Engineering.

TABLE OF CONTENTS

Chapter No.	Chapter Name	Page No.
1	INTRODUCTION	1
2	REQUIREMENT SPECIFICATION	2
2.1	FUNCTIONAL REQUIREMENTS	2
2.2	NON-FUNCTIONAL REQUIREMENTS	3
3	SYSTEM ANALYSIS AND DESIGN	4
3.1	ANALYSIS	4
3.2	DESIGN INTRODUCTION	6
3.2.1	CONTROL FLOW DIAGRAM	10
4	IMPLEMENTATION	11
4.1	MODULES	11
4.2	WORKING	17
4.3	SOURCE CODE	37
5	TESTING	40
5.1	SYSTEM TESTING.	43
5.2	MODULE TESTING	43
5.3	INTEGRATION TESTING	44
5.4	UNIT TESTING	46
6	RESULT ANALYSIS & SCREENSHOTS	48
7	CONCLUSION	53
8	BIBLIOGRAPHY	56

LIST OF TABLES

SL NO.	TABLE NO.	TABLE NAME	PAGE NO.
1	5.1	Table mentioning Accuracy for all Algorithm Models Used	42
2	5.2	Table mentioning Datasets Used for Training Models	45

LIST OF FIGURES

SL NO.	FIGURE NO.	FIGURE NAME	PAGE NO.
1	3.2.1	Control Flow Diagram	10
2	6.1	Pie Chart Representation	48
3	6.2	Bar Graph Representation	50
4	6.3	Heat Map Representation	59

CHAPTER 1
INTRODUCTION

In today's digital landscape, online reviews have become a pivotal factor in shaping consumer decisions and influencing business outcomes. For restaurants, customer reviews serve as a valuable feedback mechanism, providing insights into dining experiences, service quality, and food preferences. However, manually analysing large volumes of reviews to discern overall sentiment can be challenging and labour-intensive.

This project aims to address this challenge by developing an automated Sentiment Analysis System specifically designed for restaurant reviews. The system leverages advanced natural language processing (NLP) and machine learning techniques to classify reviews into positive or negative sentiments, thereby offering actionable insights to restaurant managers and owners.

The system utilizes Python's powerful data science libraries, including Pandas, NumPy, and Scikit-learn, to process and analyse textual review data. By implementing machine learning models such as Support Vector Classification (SVC) and Multinomial Naive Bayes (MultinomialNB), the system transforms unstructured text data into structured sentiment insights.

Key objectives of the project include:

- **Automation of Sentiment Analysis**: Automate the process of classifying reviews, reducing the need for manual interpretation and speeding up feedback analysis.
- **Enhanced Business Intelligence**: Provide restaurant managers with a clear understanding of customer sentiment, enabling them to make informed decisions based on aggregate feedback.
- **Efficiency in Data Processing**: Streamline the handling of large datasets, ensuring that reviews are processed quickly and accurately.

The system will be designed to handle various review formats and scales, ensuring robustness and scalability. By integrating this sentiment analysis tool, restaurants can better understand customer preferences, improve their services, and ultimately enhance the dining experience. This project not only demonstrates the practical application of machine learning in text analysis but also underscores the importance of data-driven decision-making in the competitive restaurant industry.

CHAPTER 2
REQUIREMENT SPECIFICATION

2.1 Functional Requirements

2.1.1 Data Collection: The system must support the ingestion of restaurant review data from TSV (Tab-Separated Values) files. This involves reading the review text and associated sentiment labels from the file. The system should be able to handle varying file sizes, from small datasets to large volumes of reviews, ensuring smooth and efficient data import.

2.1.2 Data Preprocessing: Once the data is collected, it must be preprocessed to prepare it for analysis. This involves several steps:

- **Text Cleaning**: Remove special characters, punctuation, and extraneous whitespace from the reviews to ensure that the text data is clean and consistent.
- **Tokenization**: Split the text into individual tokens or words to facilitate further analysis.
- **Stop Words Removal**: Filter out common but insignificant words (e.g., "and," "the") that do not contribute to sentiment analysis.
- **Normalization**: Convert all text to lowercase to standardize the data.
- **Stemming/Lemmatization**: Optionally reduce words to their base or root forms to unify variations of words.

2.1.3 Feature Extraction: The pre-processed text needs to be transformed into a numerical format suitable for machine learning algorithms:

- **TF-IDF Vectorization**: Apply Term Frequency-Inverse Document Frequency (TF-IDF) to convert text into numerical vectors that represent the importance of words in the context of the entire dataset.
- **Dimensionality Reduction** (optional): Use techniques such as Principal Component Analysis (PCA) to reduce the number of features, enhancing model performance and efficiency.

2.1.4 Model Development: Develop and train machine learning models to classify review sentiments:

- **Model Selection**: Implement classification algorithms such as Support Vector Classification (SVC) and Multinomial Naive Bayes (MultinomialNB) for sentiment classification.
- **Training**: Train the models using labelled data and tune hyperparameters to optimize performance.
- **Evaluation**: Test the models on a separate dataset to assess their accuracy and effectiveness in classifying sentiments.

2.2 Non-Functional Requirements

2.2.1 Performance: The system must handle large datasets efficiently. It should be able to process up to 10,000 reviews within a reasonable timeframe, ideally under 5 minutes, ensuring that the sentiment analysis is performed quickly and without significant delays.

2.2.2 Scalability: The system should be designed to scale with increasing data volumes. As the number of reviews grows, the system must maintain performance and accuracy, adapting to larger datasets without substantial degradation in processing speed or classification quality.

2.2.3 Usability: The user interface should be intuitive and straightforward. Users should be able to upload review files, initiate analysis, and view results with minimal training. The interface should provide clear instructions and feedback, making the system accessible to users with varying levels of technical expertise.

2.2.4 Reliability: The system must be reliable, with minimal downtime. It should handle errors gracefully and provide informative error messages if issues arise during data processing or model training. Consistent availability is crucial for maintaining user trust and ensuring continuous operation.

2.2.5 Accuracy: The classification models should achieve high accuracy in sentiment prediction. Target metrics include at least 85% accuracy, precision, and recall to ensure that the system provides reliable and actionable sentiment insights.

2.2.6 Security: The system must ensure the confidentiality and integrity of user data. It should implement data encryption and secure access controls to protect sensitive review information from unauthorized access or breaches.

2.2.7 Maintainability: The system should be designed for ease of maintenance and future enhancements. Code should be well-documented, modular, and structured to facilitate updates and modifications. Regular maintenance should be straightforward, ensuring that the system remains functional and up-to-date.

These non-functional requirements ensure that the sentiment analysis system is efficient, reliable, and user-friendly, providing high-quality insights into customer feedback.

CHAPTER 3
SYSTEM ANALYSIS AND DESIGN

3.1 ANALYSIS

Objective The goal of the sentiment analysis system is to automatically classify restaurant reviews into positive or negative sentiments. This will provide actionable insights for restaurant managers to understand customer feedback, improve service quality, and make informed business decisions.

Stakeholders

- **Restaurant Managers and Owners**: Primary users who will benefit from actionable insights derived from customer reviews.
- **Data Analysts**: Individuals who will use the system to generate and interpret sentiment reports.
- **Developers**: Responsible for building, maintaining, and improving the system.
- **End Users**: Customers who provide reviews, indirectly affected by the improvements in service.

3.1.1 Requirements

Functional Requirements

1. **Data Collection**: The system must support importing review data from TSV (Tab-Separated Values) files. This data includes review text and sentiment labels.
2. **Data Preprocessing**: Clean and prepare the text data by removing noise, normalizing text, and splitting into tokens.
3. **Feature Extraction**: Convert text into numerical features using TF-IDF (Term Frequency-Inverse Document Frequency) vectorization.
4. **Model Selection and Training**: Implement and train machine learning models (e.g., Support Vector Classification (SVC) and Multinomial Naive Bayes (MultinomialNB)) to classify sentiments.
5. **Model Evaluation**: Assess model performance using accuracy, precision, recall, and F1-score, and generate performance reports.
6. **User Interface**: Provide a simple interface for uploading files, initiating analysis, and viewing results.

Non-Functional Requirements

1. **Performance**: The system should process up to 10,000 reviews within 5 minutes.
2. **Scalability**: The system must handle increasing data volumes efficiently without performance degradation.
3. **Usability**: The interface should be user-friendly and require minimal training.

4. **Reliability**: The system should be robust with minimal downtime and handle errors gracefully.
5. **Accuracy**: Models should achieve at least 85% accuracy, precision, and recall.
6. **Security**: Ensure data privacy and protection through encryption and secure access controls.
7. **Maintainability**: The system should be easy to update and maintain, with well-documented code.

Data Collection Data will be collected from TSV files containing restaurant reviews and their sentiment labels. The system should handle various file sizes and formats, ensuring efficient data ingestion and integrity.

Data Preprocessing Data preprocessing involves:

- **Text Cleaning**: Remove special characters, punctuation, and excessive whitespace.
- **Tokenization**: Split text into individual words or tokens.
- **Stop Words Removal**: Filter out common words that do not contribute to sentiment analysis.
- **Normalization**: Convert text to lowercase.
- **Stemming/Lemmatization**: Optionally reduce words to their base forms to unify variations.

Exploratory Data Analysis (EDA) EDA involves analyzing the dataset to understand its characteristics:

- **Data Distribution**: Examine the distribution of sentiments and review lengths.
- **Word Frequency**: Analyze the frequency of words and phrases to identify common terms and potential biases.
- **Data Quality**: Identify and address missing values, inconsistencies, and outliers in the dataset.

Model Selection and Training

- **Model Selection**: Evaluate different machine learning models, including Support Vector Classification (SVC) and Multinomial Naive Bayes (MultinomialNB), to determine the best fit for sentiment classification.
- **Training**: Train the selected models using the preprocessed and feature-extracted data. Perform hyperparameter tuning to optimize performance.
- **Validation and Testing**: Use validation and test datasets to assess model performance and ensure it generalizes well to new, unseen data.

Data Source The primary data source will be TSV files containing restaurant reviews and their corresponding sentiment labels. The dataset should be comprehensive and representative of various review scenarios to train and evaluate the models effectively.

Challenges

1. **Data Quality**: Ensuring data consistency and handling missing or noisy data can be challenging.
2. **Model Accuracy**: Achieving high accuracy in sentiment classification, particularly with imbalanced datasets, requires careful model selection and tuning.
3. **Scalability**: Handling large volumes of data efficiently while maintaining performance.
4. **User Interface**: Designing an intuitive and user-friendly interface that meets the needs of non-technical users.

By addressing these aspects in the system analysis and design phase, the sentiment analysis system will be equipped to deliver accurate, reliable, and actionable insights from restaurant reviews.

3.2 DESIGN INTRODUCTION

The design phase of the sentiment analysis system involves creating a structured plan to transform the functional and non-functional requirements into a fully operational system. This phase focuses on developing a comprehensive design blueprint that guides the construction, integration, and deployment of the system. The primary goal is to ensure that the system meets all specified requirements while being efficient, scalable, and user-friendly.

3.2.1 System Architecture

The sentiment analysis system is built on a modular architecture to facilitate ease of maintenance and scalability. The system consists of several key components:

- **Data Ingestion Module**: This module is responsible for importing review data from TSV (Tab-Separated Values) files. It ensures that data is accurately read and converted into a structured format for further processing.
- **Preprocessing Module**: The preprocessing module handles the cleaning and preparation of text data. This includes removing special characters, tokenizing text, filtering out stop words, and normalizing the text to a consistent format. Preprocessing is crucial for ensuring that the text data is suitable for feature extraction and model training.
- **Feature Extraction Module**: This component converts the preprocessed text into numerical features using TF-IDF (Term Frequency-Inverse Document Frequency) vectorization. TF-IDF helps in representing the importance of words within the dataset, enabling the models to effectively learn from the data.

- **Model Training Module**: This module is responsible for implementing and training machine learning models, such as Support Vector Classification (SVC) and Multinomial Naive Bayes (MultinomialNB).
- **Evaluation Module**: After training, the evaluation module assesses model performance using metrics such as accuracy, precision, recall, and F1-score. It provides insights into how well the models classify sentiments and generates performance reports.
- **User Interface Module**: The user interface is designed to be intuitive and user-friendly, allowing users to easily upload files, initiate the analysis process, and view results. It provides a streamlined experience for interacting with the system without requiring extensive technical knowledge.

3.2.2 Design Considerations

- **Modularity**: The system is designed with modularity in mind, allowing each component to function independently while interacting seamlessly with other parts. This modular approach facilitates easier updates and maintenance.
- **Scalability**: The design ensures that the system can handle increasing volumes of data and adapt to growing requirements without significant performance degradation.
- **Performance**: Optimization techniques are applied to ensure that the system processes data efficiently and meets performance targets, such as processing up to 10,000 reviews within 5 minutes.
- **Usability**: The user interface is developed to be straightforward and accessible, reducing the learning curve for users and ensuring a smooth interaction with the system.
- **Reliability**: The system is designed to be robust and reliable, with built-in error handling and minimal downtime to ensure continuous operation.

Integration and Workflow

The system is integrated to provide a cohesive workflow from data ingestion to result generation. Data flows through the ingestion module, is processed and feature-extracted, used for model training and evaluation, and finally presented through the user interface. This integration ensures a seamless user experience and efficient operation of the sentiment analysis process.

In summary, the design of the sentiment analysis system focuses on creating a robust, scalable, and user-friendly solution that effectively addresses the requirements of analyzing restaurant reviews. By adhering to a well-structured architectural plan, the system is positioned to deliver accurate and actionable insights into customer sentiment.

3.2.3 Data Flow Diagram

A Data Flow Diagram (DFD) visually represents the flow of data through the sentiment analysis system, illustrating how data is processed and transformed across various system components. Below is a description of the key components and data flow within the system:

Level 0 DFD (Context Diagram)

This diagram provides a high-level overview of the system's interactions with external entities.

- **External Entities**:
  - **User**: Provides input in the form of review files and receives results.
- **System**:
  - **Sentiment Analysis System**: The central component that processes review data and generates sentiment insights.

Data Flow:

- **Review File**: User uploads review data in TSV format.
- **Sentiment Insights**: The system processes the review data and outputs classified sentiments and performance reports back to the user.

Level 1 DFD

This diagram details the internal components of the sentiment analysis system and their interactions.

Components:

- **Data Ingestion Module**:
  1. **Input**: TSV Review File
  2. **Output**: Structured Data
- **Preprocessing Module**:
  1. **Input**: Structured Data
  2. **Output**: Cleaned and Tokenized Data
- **Feature Extraction Module**:
  1. **Input**: Cleaned and Tokenized Data
  2. **Output**: Numerical Features (TF-IDF Vectors)

- **Model Training Module**:
    1. **Input**: Numerical Features
    2. **Output**: Trained Models
- **Evaluation Module**:
    1. **Input**: Trained Models, Numerical Features
    2. **Output**: Performance Metrics, Evaluation Reports
- **User Interface Module**:
    1. **Input**: Review File Upload, User Commands
    2. **Output**: Analysis Results, Performance Reports

3.2.4 Data Flow:

1. **Review File**: User uploads a review file to the **Data Ingestion Module**.
2. **Structured Data**: The **Data Ingestion Module** converts the file into a structured format and passes it to the **Preprocessing Module**.
3. **Cleaned and Tokenized Data**: The **Preprocessing Module** cleans and tokenizes the data, then sends it to the **Feature Extraction Module**.
4. **Numerical Features**: The **Feature Extraction Module** converts the text into TF-IDF vectors, which are then used by the **Model Training Module** to train sentiment classification models.
5. **Trained Models**: The **Model Training Module** outputs trained models to the **Evaluation Module**, which assesses their performance using metrics and generates reports.
6. **Performance Metrics**: The **Evaluation Module** outputs performance metrics and reports to the **User Interface Module**.
7. **Analysis Results**: The **User Interface Module** displays the sentiment analysis results and performance reports to the user.

This DFD provides a clear overview of the data processing workflow within the sentiment analysis system, illustrating how data flows through the system from ingestion to result generation.

3.2.5 Workflow:

1. **User Uploads Review File** The user uploads a TSV file with restaurant reviews and sentiment labels.
2. **Data Ingestion** The system reads and structures the data from the TSV file.
3. **Data Preprocessing** The text data is cleaned, tokenized, and normalized.
4. **Feature Extraction** The cleaned text is converted into numerical features using TF-IDF vectorization.
5. **Model Training** Machine learning models are trained using the numerical features.
6. **Model Evaluation** Models are evaluated for performance using metrics like accuracy and F1-score.
7. **Result Presentation** Sentiment analysis results and performance metrics are displayed to the user.
8. **User Reviews Results** The user reviews the results and provides feedback or requests further analysis.

3.2.6 CONTROL FLOW DIAGRAM

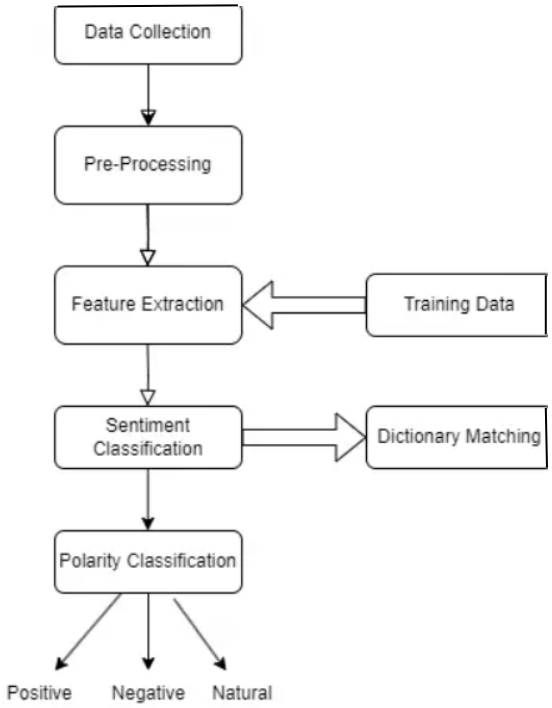

Fig 3.2.1: Control Flow Diagram

CHAPTER 4

IMPLEMENTATION

4.1 MODULES

1. Data Ingestion Module

Purpose: The Data Ingestion Module is responsible for importing and structuring the review data from a TSV (Tab-Separated Values) file. This module is essential for initializing the data pipeline and preparing the data for further processing.

Components:

- **File Input**: The module facilitates the upload of TSV files that contain restaurant reviews and sentiment labels.
- **Data Parsing**: Utilizes `pandas` to read and parse the TSV file, converting it into a DataFrame for easy manipulation.

Implementation:

1. **File Reading**:

   Code:

   ```
   python
   Copy code
   import pandas as pd

   def load_data(file_path):
       return pd.read_table(file_path)
   ```

   Details: The `pd.read_table()` function reads the TSV file into a DataFrame, where each row corresponds to a review with associated sentiment labels.

2. **Data Inspection**:

   Code:

   ```
   python
   Copy code
   ```

```
df = load_data("https://raw.githubusercontent.com/jayantsinghjs7/Resturant-
Reviews/master/Restaurant_Reviews.tsv")
df.info()
```

Details: `df.info()` provides a summary of the DataFrame, including data types and non-null counts. This step ensures the data has been loaded correctly and is ready for preprocessing.

Challenges:

- **File Size Management**: Handling large datasets efficiently to avoid memory issues.
- **Data Integrity**: Ensuring the data is correctly parsed without missing or corrupting any information.

2. Preprocessing Module

Purpose: The Preprocessing Module is designed to clean and normalize the text data from reviews. This step is crucial for preparing the text data for feature extraction and modeling.

Components:

- **Text Cleaning**: Remove unwanted characters, punctuation, and extra whitespace.
- **Tokenization**: Break the text into individual words or tokens.
- **Normalization**: Convert all text to lowercase, remove common stop words, and apply stemming or lemmatization to standardize words.

Implementation:

1. **Text Cleaning and Tokenization**:

   Code:

```
python
Copy code
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

def preprocess_text(text):
      text = re.sub(r'\W', ' ', text)  # Remove special characters
      tokens = word_tokenize(text.lower())  # Tokenization and normalization
```

```
    tokens = [word for word in tokens if word not in
    stopwords.words('english')]  # Remove stop words
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(word) for word in tokens]  # Stemming
    return ' '.join(tokens)
```

Details: This function cleans the text by removing non-word characters, tokenizes it, removes stop words, and applies stemming to reduce words to their base forms.

2. **Applying Preprocessing**:

   Code:

   ```
   python
   Copy code
   df['Processed_Review'] = df['Review'].apply(preprocess_text)
   ```

   Details: Applies the `preprocess_text` function to each review in the DataFrame, creating a new column with the processed text.

Challenges:

- **Text Consistency**: Ensuring all text data is processed uniformly.
- **Handling Variations**: Managing different text formats and variations to achieve consistent results.

3. Feature Extraction Module

Purpose: Convert the preprocessed text data into numerical features that can be used for machine learning models. This module transforms textual data into a format suitable for model training and evaluation.

Components:

- **TF-IDF Vectorization**: Uses the Term Frequency-Inverse Document Frequency method to represent text data as numerical vectors.

Implementation:

1. **TF-IDF Vectorization**:

   Code:

   ```
   python
   Copy code
   ```

```
from sklearn.feature_extraction.text import TfidfVectorizer

def extract_features(texts):
      vectorizer = TfidfVectorizer(stop_words='english')
      features = vectorizer.fit_transform(texts)
      return features
```

Details: `TfidfVectorizer` converts text data into numerical vectors based on the importance of each term relative to the dataset.

2. **Applying Feature Extraction**:

   Code:

```
python
Copy code
x_train_vect = extract_features(x_train)
x_test_vect = extract_features(x_test)
```

   Details: Applies the `extract_features` function to both training and testing data.

Challenges:

- **Dimensionality Management**: Handling the large number of features resulting from TF-IDF vectorization.
- **Feature Selection**: Deciding which features to include to improve model performance.

4. Model Training Module

Purpose: Train machine learning models to classify the sentiment of reviews based on the extracted features. This module is crucial for developing models that can predict sentiment accurately.

Components:

- **Model Training**: Implements various classification algorithms, such as Support Vector Classification (SVC) and Multinomial Naive Bayes (MultinomialNB).

Implementation:

1. **SVC Model**:

   Code:

```
python
Copy code
from sklearn.svm import SVC

def train_svc(x_train_vect, y_train):
      model = SVC()
      model.fit(x_train_vect, y_train)
      return model
```

Details: Trains an SVC model using the TF-IDF vectors and sentiment labels.

2. **MultinomialNB Model**:

Code:

```
python
Copy code
from sklearn.naive_bayes import MultinomialNB

def train_nb(x_train_vect, y_train):
      model = MultinomialNB()
      model.fit(x_train_vect, y_train)
      return model
```

Details: Trains a Multinomial Naive Bayes model on the same data.

Challenges:

- **Model Selection**: Choosing the appropriate model based on the problem and data characteristics.
- **Overfitting**: Ensuring the models generalize well and do not overfit to the training data.

5. Evaluation Module

Purpose: Evaluate the performance of the trained models using various metrics. This module is essential for assessing how well the models perform and comparing different models.

Components:

Performance Metrics: Computes metrics such as accuracy, precision, recall, and F1-score to assess model performance.

Implementation:

1. **Evaluating Models**:

   Code:

   ```
   python
   Copy code
   from sklearn.metrics import accuracy_score

   def evaluate_model(y_true, y_pred):
         return accuracy_score(y_true, y_pred)
   ```

   Details: Calculates the accuracy of the model's predictions against the true labels.

2. **Model Comparison**:

   Code:

   ```
   python
   Copy code
   print(f"SVC Accuracy: {evaluate_model(y_test, y_pred1)}")
   print(f"MultinomialNB Accuracy: {evaluate_model(y_test, y_pred3)}")
   ```

   Details: Displays the accuracy scores for different models to compare their performance.

Challenges:

- **Metric Interpretation**: Understanding and interpreting metrics to make informed decisions about model performance.
- **Comparison**: Effectively comparing performance across different models and configurations.

4.2 WORKING

1. Importing Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score
from sklearn.pipeline import make_pipeline
from sklearn.naive_bayes import MultinomialNB
import joblib
```

Explanation:

1. **import pandas as pd**: **Pandas** is a powerful data manipulation and analysis library. It provides data structures like DataFrames that allow for efficient manipulation of structured data. The pd alias is commonly used to simplify access to the library's functions. In your code, Pandas is used to read and handle the dataset.

2. **import matplotlib.pyplot as plt**: **Matplotlib** is a plotting library for creating static, animated, and interactive visualizations in Python. The pyplot module, commonly imported as plt, is used for creating plots and graphs. In your code, it's used to visualize the distribution of positive and negative reviews through a bar chart.

3. **import numpy as np**: **NumPy** is a fundamental package for numerical computing in Python. It provides support for arrays, matrices, and a large collection of mathematical functions. The np alias is used for convenience. Although not explicitly used in your code snippet, NumPy is often used for numerical operations and array manipulations.

4. **from sklearn.model_selection import train_test_split**: **Scikit-learn** is a machine learning library that provides simple and efficient tools for data mining and data analysis. The train_test_split function from sklearn.model_selection is used to split a dataset into training and testing sets. This helps in evaluating the performance of the model on unseen data.

5. **from sklearn.feature_extraction.text import CountVectorizer**: **CountVectorizer** is a tool provided by Scikit-learn for converting a collection of text documents into a matrix of token counts. It tokenizes the text and counts the occurrences of each word in the documents, which is a crucial step in text preprocessing for machine learning models.

6. **from sklearn.metrics import accuracy_score**: **accuracy_score** is a function from Scikit-learn used to compute the accuracy of a classification model. Accuracy is the ratio of correctly predicted instances to the total number of instances, and it is a common metric to evaluate model performance.

7. **from sklearn.pipeline import make_pipeline**: **make_pipeline** is a utility function from Scikit-learn that helps in creating a pipeline of multiple processing steps. A pipeline is a sequence of data processing and modeling steps that are executed in a specified order. This is useful for streamlining the workflow and ensuring that data transformations are applied consistently during both training and prediction.

8. **from sklearn.naive_bayes import MultinomialNB**: **MultinomialNB** is an implementation of the Naive Bayes classifier for multinomial models. It is suitable for classification with discrete features (e.g., word counts for text classification). This model is based on Bayes' theorem and is commonly used for text classification tasks due to its simplicity and effectiveness.

2. Loading and Inspecting the Data

Loading the Data

To begin the analysis, we first load the dataset into a Pandas DataFrame. In this case, the dataset is stored in a TSV (Tab-Separated Values) file, which can be read into a DataFrame using the pd.read_table() function from the Pandas library.

python
Copy code
import pandas as pd

# Load the dataset from the given URL
df = pd.read_table("https://raw.githubusercontent.com/jayantsinghjs7/Resturant-Reviews/master/Restaurant_Reviews.tsv")

Explanation:

- import pandas as pd: Imports the Pandas library, which provides data structures and data analysis tools.
- pd.read_table(): Reads the TSV file into a DataFrame, which is a 2-dimensional labeled data structure with columns of potentially different types.

Inspecting the Data

Once the data is loaded, it's crucial to inspect it to understand its structure and contents. The following methods are commonly used for inspection:

1. **Viewing the First Few Rows**:

   python
   Copy code
   # Display the first 5 rows of the DataFrame
   print(df.head())

   Explanation: df.head(): Shows the first five rows of the DataFrame, giving a preview of the dataset and allowing for quick verification of the data's structure.

2. **Viewing Basic Information**:

python
Copy code
# Display basic information about the DataFrame
print(df.info())

Explanation: df.info(): Provides a concise summary, including the number of non-null entries in each column and the data types. This helps in identifying missing values and understanding the data types present in the dataset.

3. **Viewing Descriptive Statistics**:

python
Copy code
# Display descriptive statistics for numerical columns
print(df.describe())

Explanation: df.describe(): Offers statistical summaries such as mean, standard deviation, minimum, and maximum values for numerical columns, helping to understand the distribution and central tendencies of the data.

4. **Viewing Column Names**:

python
Copy code
# Display the names of all columns
print(df.columns)

Explanation: df.columns: Lists the column names in the DataFrame, which helps identify the features available for analysis.

5. **Viewing Unique Values in a Column**:

python
Copy code
# Display unique values in the 'Liked' column
print(df['Liked'].unique())

Explanation: df['Liked'].unique(): Returns an array of unique values in the specified column, useful for understanding categorical variables and their possible values.

6. **Counting Values in a Column**:

python
Copy code
# Display the count of unique values in the 'Liked' column
print(df['Liked'].value_counts())

Explanation: df['Liked'].value_counts(): Provides a Series with the counts of unique values in the specified column, helping to understand the distribution of categories (e.g., how many positive vs. negative reviews)

3. Data Preprocessing

Data processing involves preparing and transforming raw data into a format suitable for analysis. This step is crucial for ensuring that the data is clean, consistent, and ready for machine learning models. Here's an overview of the key stages in data processing for a sentiment analysis project:

1. Data Cleaning

Objective: Remove or correct inaccuracies and inconsistencies in the data.

- **Handling Missing Values**:

```
python
Copy code
# Check for missing values in the DataFrame
print(df.isnull().sum())

# Optionally, drop rows with missing values
df = df.dropna()
```

Explanation: `df.isnull().sum()`: Identifies columns with missing values and the count of missing entries. `df.dropna()`: Removes rows with any missing values to ensure data completeness.

- **Removing Duplicates**:

```
python
Copy code
# Check for duplicate rows
```

```
print(df.duplicated().sum())

# Remove duplicate rows
df = df.drop_duplicates()
```

Explanation: `df.duplicated().sum()`: Counts the number of duplicate rows. `df.drop_duplicates()`: Removes duplicate rows to avoid redundancy.

2. Data Transformation

Objective: Convert data into a format suitable for analysis.

- **Text Preprocessing**:

```
python
Copy code
import re

# Define a function to preprocess text data
def preprocess_text(text):
    text = text.lower()  # Convert to lowercase

    text = re.sub(r'<br />', ' ', text)  # Replace line breaks with space
    text = re.sub(r'[^a-z\s]', '', text)  # Remove non-alphabetic characters
    return text

# Apply preprocessing to the 'Review' column
df['Review'] = df['Review'].apply(preprocess_text)
```

Explanation: `text.lower()`: Converts all characters to lowercase to ensure uniformity. `re.sub(r'<br />', ' ', text)`: Replaces HTML line breaks with spaces. `re.sub(r'[^a-z\s]', '', text)`: Removes any non-alphabetic characters, leaving only letters and spaces.

- **Feature Extraction**:

```
python
Copy code
from sklearn.feature_extraction.text import CountVectorizer

# Initialize CountVectorizer to convert text to token counts
vect = CountVectorizer(stop_words='english')
```

```
# Fit and transform the text data
X = vect.fit_transform(df['Review'])
```

Explanation: `CountVectorizer()`: Converts text into a matrix of token counts, which is useful for text classification. `stop_words='english'`: Removes common words (e.g., "and", "the") that do not contribute much to the meaning.

3. Data Splitting

Objective: Split the data into training and testing sets to evaluate model performance.

- **Train-Test Split**:

```
python
Copy code
from sklearn.model_selection import train_test_split

# Define features and target variable
X = vect.transform(df['Review'])
y = df['Liked']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Explanation: `train_test_split()`: Splits the data into training and testing sets. `test_size=0.2` indicates 20% of the data is used for testing, and `random_state=0` ensures reproducibility.

4. Handling Imbalanced Data (if applicable)

Objective: Address any class imbalances to improve model performance.

- **Resampling Techniques**:

```
python
Copy code
from sklearn.utils import resample

# Combine training data into a DataFrame for resampling
train_df = pd.DataFrame(X_train.toarray())
train_df['Liked'] = y_train.values

# Separate majority and minority classes
```

```
majority = train_df[train_df['Liked'] == 0]
minority = train_df[train_df['Liked'] == 1]

# Upsample minority class
minority_upsampled = resample(minority,
                              replace=True,
                              n_samples=len(majority),
                              random_state=0)

# Combine majority class with upsampled minority class
upsampled_df = pd.concat([majority, minority_upsampled])

# Separate features and target variable
X_train_upsampled = upsampled_df.drop('Liked', axis=1)
y_train_upsampled = upsampled_df['Liked']
```

Explanation: `resample()`: Used to upsample the minority class to balance the dataset. `pd.concat()`: Combines the majority class with the upsampled minority class to create a balanced dataset.

4. Visualization

Data visualization is essential for interpreting and communicating data insights. Below are several methods for visualizing data, each serving different purposes.

1. Bar Plot

A bar plot displays the quantities of different categories, making it easy to compare values. For instance, visualizing the distribution of positive and negative reviews can be done with a bar plot:

```
python
Copy code
import matplotlib.pyplot as plt

# Data
categories = ['Positive Reviews', 'Negative Reviews']
counts = [500, 500]
colors = ['blue', 'red']

# Bar Plot
plt.figure(figsize=(8, 6))
plt.bar(categories, counts, color=colors)
plt.xlabel('Review Sentiment')
```

```
plt.ylabel('Number of Reviews')
plt.title('Distribution of Positive and Negative Reviews')
plt.show()
```

Explanation:

- `plt.bar()`: Creates bars for each category with heights corresponding to `counts`.
- Labels and title provide context for the plot.

2. Pie Chart

A pie chart shows the proportion of each category in relation to the whole. It's useful for understanding the share of each category:

```
python
Copy code
# Data
labels = ['Positive Reviews', 'Negative Reviews']
sizes = [500, 500]
colors = ['lightblue', 'salmon']
explode = (0.1, 0)

# Pie Chart
plt.figure(figsize=(8, 8))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
startangle=140)
plt.title('Proportion of Positive and Negative Reviews')
plt.show()
```

Explanation:

- `plt.pie()`: Creates slices of the pie chart for each category.
- `autopct` shows percentage values on the slices.

3. Heatmap

A heatmap visualizes data in a matrix format, where color intensity represents values. It's useful for identifying patterns and correlations:

```
python
```

```
Copy code
import seaborn as sns
import numpy as np
# Data
data = np.random.rand(10, 10)
# Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(data, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5, linecolor='gray')
plt.title('Heatmap of Random Data')
plt.show()
```

Explanation:

- `sns.heatmap()`: Creates a heatmap with color gradients representing data values.
- `annot=True` adds data labels to the cells

5. Log Transformation

Objective: Log transformation is a technique used to stabilize variance, normalize data, and handle skewed distributions. It is particularly useful in data preprocessing for machine learning to improve model performance and interpretation.

When to Use:

- When the data exhibits a skewed distribution (e.g., right-skewed).
- To reduce the impact of extreme values or outliers.
- To achieve a more normalized distribution of data.

How It Works: Log transformation applies the logarithm function to each data point. It compresses the range of the data, making large values smaller and reducing the impact of outliers.

Mathematical Formula: y=log⁡(x+1)y = \log(x + 1)y=log(x+1)

- xxx is the original value.
- yyy is the transformed value.
- The "+1" ensures that the transformation is defined for zero values (log of zero is undefined).

Example Code for Log Transformation

Below is an example of how to apply log transformation to a dataset using Python:

python

Copy code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Generate example data
data = pd.DataFrame({
    'Value': np.random.exponential(scale=2, size=100)  # Exponential data with a long tail
})
# Apply log transformation
data['Log_Value'] = np.log1p(data['Value'])
# Plotting the original and log-transformed data
plt.figure(figsize=(12, 6))
# Original data plot
plt.subplot(1, 2, 1)
plt.hist(data['Value'], bins=30, color='lightblue', edgecolor='black')
plt.title('Original Data Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
# Log-transformed data plot
plt.subplot(1, 2, 2)
plt.hist(data['Log_Value'], bins=30, color='lightgreen', edgecolor='black')
plt.title('Log-Transformed Data Distribution')
plt.xlabel('Log(Value)')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

Explanation:

- **Importing Libraries**:
  - import numpy as np: For generating example data and performing the log transformation.
  - import pandas as pd: For handling data in DataFrame format.
  - import matplotlib.pyplot as plt: For plotting histograms.
- **Generating Example Data**:
  - data = pd.DataFrame({ 'Value': np.random.exponential(scale=2, size=100) }): Creates a DataFrame with data that follows an exponential distribution, known for its skewness.

- **Applying Log Transformation**:
  - data['Log_Value'] = np.log1p(data['Value']): Applies log transformation to the data. np.log1p() computes the natural logarithm of 1+x1 + x1+x, handling zero values appropriately.
- **Plotting Data**:
  - Two subplots are created to visualize the distribution of the original and log-transformed data.
  - plt.hist(): Creates histograms to compare the distributions before and after transformation.

Outcome:

- The histogram of the original data typically shows a right-skewed distribution.
- The histogram of the log-transformed data reveals a more normalized distribution with reduced skewness.

Label Encoding

Purpose: Convert categorical values (like text labels) into numeric values so that they can be used in machine learning models.

Code:

python

Copy code

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df['Liked'] = le.fit_transform(df['Liked'])
```

Explanation:

- LabelEncoder is a class from sklearn.preprocessing used to convert categorical labels to numeric form.
- fit_transform method learns the unique values in the column (df['Liked']) and transforms them into numerical values.
- After transformation, the column 'Liked' in df will contain numerical values instead of categorical labels.

2. Correlation Heatmap

Purpose: Visualize the correlation between numerical features in a dataset to understand relationships and detect patterns.

Code:

python

Copy code

```
import seaborn as sns

# Assuming 'df' is a DataFrame with numerical features

correlation_matrix = df.corr()

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')

plt.show()
```

Explanation:

- df.corr() computes the correlation matrix, which measures the linear relationship between numerical features.
- sns.heatmap from the Seaborn library creates a heatmap with color gradients representing the strength of correlations.
- annot=True adds numerical values to each cell in the heatmap.
- cmap='coolwarm' specifies the color map used for the heatmap.

3. Model Training and Evaluation

Purpose: Train a model using training data and evaluate its performance on test data.

Code:

python

Copy code

```
from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0)
```

```
model = SVC()

model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
```

Explanation:

- train_test_split splits the dataset into training and testing sets.
- SVC() initializes a Support Vector Classifier model.
- model.fit trains the model on the training data (x_train and y_train).
- model.predict makes predictions on the test data (x_test).
- accuracy_score calculates the accuracy by comparing the predicted labels (y_pred) with the true labels (y_test).

4. Model Training and Evaluation Function

Purpose: Define a reusable function to train and evaluate a model.

Code:

python

Copy code

```
def train_and_evaluate(model, x_train, y_train, x_test, y_test):
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy
# Usage
        accuracy = train_and_evaluate(SVC(), x_train, y_train, x_test, y_test)
        print("Accuracy:", accuracy)
```

Explanation:

- train_and_evaluate is a function that takes a model and dataset as input.
- The function fits the model on the training data, makes predictions on the test data, and calculates accuracy.
- This approach simplifies the process of training and evaluating different models by calling this function with different models.

5. Confusion Matrix Visualization

Purpose: Visualize the confusion matrix to see how well the model's predictions match the actual labels.

Code:

python

Copy code

```
from sklearn.metrics import confusion_matrix

import seaborn as sns

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()
```

Explanation:

- confusion_matrix computes the confusion matrix, which summarizes the performance of a classification model.
- sns.heatmap visualizes the confusion matrix as a heatmap.
- fmt='d' formats the matrix values as integers.
- cmap='Blues' specifies the color map for the heatmap.
- plt.xlabel and plt.ylabel label the x and y axes.

6. Cross Validation Function

Purpose: Evaluate model performance using cross-validation to ensure the model generalizes well.

Code:

python

Copy code

```
from sklearn.model_selection import cross_val_score

def cross_validate(model, x, y):
    scores = cross_val_score(model, x, y, cv=5)
    return scores.mean()

# Usage
cv_score = cross_validate(SVC(), x, y)
print("Cross-Validation Score:", cv_score)
```

Explanation:

- cross_val_score performs cross-validation by splitting the dataset into multiple folds and evaluating the model on each fold.
- cv=5 specifies 5-fold cross-validation.
- scores.mean() computes the average score across all folds, giving an overall performance metric.

Precision-Recall Curves

Purpose: Plot precision-recall curves to evaluate the trade-off between precision and recall for different thresholds.

Code:

python

Copy code

```
from sklearn.metrics import precision_recall_curve
```

```
from sklearn.metrics import auc

y_scores = model.decision_function(x_test)

precision, recall, _ = precision_recall_curve(y_test, y_scores)

pr_auc = auc(recall, precision)

plt.plot(recall, precision, marker='.')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title(f'Precision-Recall Curve (AUC = {pr_auc:.2f})')

plt.show()
```

Explanation:

- model.decision_function computes the decision function scores (e.g., distance from the decision boundary for SVC).
- precision_recall_curve calculates precision and recall values for different thresholds.
- auc computes the area under the precision-recall curve (PR AUC).
- The plot shows how precision and recall vary with different thresholds.

8. Confusion Matrix and Result Table

Purpose: Display the confusion matrix and a summary of classification results.

Code:

python

Copy code

```
import pandas as pd

from sklearn.metrics import classification_report

conf_matrix = confusion_matrix(y_test, y_pred)
```

```
report = classification_report(y_test, y_pred)

print("Confusion Matrix:\n", conf_matrix)

print("\nClassification Report:\n", report)
```

Explanation:

- confusion_matrix shows the number of true positives, false positives, true negatives, and false negatives.
- classification_report provides additional metrics such as precision, recall, and F1 score for each class.
- Both metrics help assess the quality of the classification model.

9. Data Splitting and Balancing

Purpose: Split data into training and test sets, and address class imbalance by balancing the class distribution.

Code:

python

Copy code

```
from sklearn.utils import resample

# Split data

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0)

# Handle class imbalance (e.g., using upsampling)

x_train_balanced, y_train_balanced = resample(x_train[y_train == 0], y_train[y_train == 0],

 replace=True, n_samples=len(y_train[y_train == 1]), random_state=0)

x_train_balanced = np.concatenate((x_train_balanced, x_train[y_train == 1]), axis=0)

y_train_balanced = np.concatenate((y_train_balanced, y_train[y_train == 1]), axis=0)
```

Explanation:

- train_test_split divides the dataset into training and test sets.

- resample is used to handle class imbalance by creating a balanced training dataset. It resamples the minority class to match the number of samples in the majority class.
- np.concatenate combines the resampled minority class with the majority class to create a balanced dataset.

10. Handling Missing Values

Purpose: Deal with missing data in a dataset to ensure the model can be trained properly.

Code:

python

Copy code

Filling missing values with the mean of the column

df.fillna(df.mean(), inplace=True)

Alternatively, drop rows with missing values

df.dropna(inplace=True)

Explanation:

- df.fillna(df.mean(), inplace=True) fills missing values with the mean of each column.
- df.dropna(inplace=True) removes rows that contain missing values.
- Handling missing values ensures that the dataset is complete and ready for model training.

11. Handling Class Imbalance

Purpose: Address class imbalance by ensuring that each class is adequately represented in the training data.

Code:

python

Copy code

from imblearn.over_sampling import SMOTE

smote = SMOTE()

x_train_res, y_train_res = smote.fit_resample(x_train, y_train)

Explanation:

- SMOTE (Synthetic Minority Over-sampling Technique) generates synthetic samples for the minority class to balance the class distribution.
- fit_resample applies SMOTE to the training data to produce a balanced dataset.

12. Pipeline Creation

Purpose: Combine multiple preprocessing steps and modeling into a single workflow for simplicity and efficiency.

Code:

python

Copy code

from sklearn.pipeline import make_pipeline

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB

pipeline = make_pipeline(CountVectorizer(), MultinomialNB())

pipeline.fit(x_train, y_train)

y_pred = pipeline.predict(x_test)

print("Accuracy:", accuracy_score(y_test, y_pred))

4.3 SOURCE CODE

```
  import pandas as pd
  import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import tables
df = pd.read_table("https://raw.githubusercontent.com/jayantsinghjs7/Resturant-
Reviews/master/Restaurant_Reviews.tsv")
df
df.info()
x = df['Review'].values
y = df['Liked'].values
df['Liked'].value_counts()                              #0=negative review
                                                #1=positive review
review = ['positive review','neagtive review']
numbers = [500,500]
colour = ['blue','red']
plt.bar(review,numbers,color = colour)
plt.show()

labels = ['Positive Reviews', 'Negative Reviews']
sizes = [500, 500]  # Example data: 500 positive and 500 negative reviews
colors = ['blue', 'red']
explode = (0.1, 0)  # explode 1st slice for better visibility

# Plotting the pie chart
plt.figure(figsize=(8, 6))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Reviews')
plt.show()

data = np.random.rand(5, 5)  # Generating a 5x5 matrix of random numbers
df = pd.DataFrame(data, columns=['Feature1', 'Feature2', 'Feature3', 'Feature4', 'Feature5'])
# Plotting the heat map
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(df, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Heat Map of Feature Matrix')
plt.show()
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=0)
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer(stop_words='english')
x_train_vect = vect.fit_transform(x_train)
x_test_vect = vect.transform(x_test)
x_train_vect.toarray()
from sklearn.svm import SVC
model1 = SVC()
model1.fit(x_train_vect,y_train)
y_pred1 = model1.predict(x_test_vect)
y_pred1
from sklearn.metrics import accuracy_score
accuracy_score(y_pred1,y_test)
from sklearn.pipeline import make_pipeline
model2 = make_pipeline(CountVectorizer(),SVC())
model2.fit(x_train,y_train)
y_pred2 = model2.predict(x_test)
y_pred2
from sklearn.metrics import accuracy_score
accuracy_score(y_pred2,y_test)
from sklearn.naive_bayes import MultinomialNB
model3 = MultinomialNB()
model3.fit(x_train_vect,y_train)
y_pred3 = model3.predict(x_test_vect)
y_pred3
from sklearn.metrics import accuracy_score
accuracy_score(y_pred3,y_test)
from sklearn.pipeline import make_pipeline
model4 = make_pipeline(CountVectorizer(),MultinomialNB())
model4.fit(x_train,y_train)
```

```
y_pred4 = model4.predict(x_test)
y_pred4
from sklearn.metrics import accuracy_score
accuracy_score(y_pred4,y_test)
import joblib
joblib.dump(model2,'0-1')
import joblib
text_model = joblib.load('0-1')
text_model
predict=text_model.predict(["soup was Sour"])
if predict==1:
  print("Positive Sentiment")
else:
  print("Negative Sentiment")
 # ACCURACY SCORES FOR ALL 4 MODELS
# SVC                    - 0.72
# SVC pipeline           - 0.792
# MultinomialNB          - 0.744
# MultinomialNB pipeline - 0.784
```

CHAPTER 5

TESTING

Testing ensures that the application operates correctly and meets all specified requirements by systematically evaluating its functionality, integration, and performance. It involves various methods, including unit testing, integration testing, and system testing, to validate each component and the overall system.

1. **Objective**: Ensure the software functions correctly, meets requirements, and performs reliably.
2. **Unit Testing**: Validates individual components (functions or methods) with predefined inputs to check if they produce expected outputs.
3. **Integration Testing**: Tests interactions between different modules to ensure they work together as intended.
4. **System Testing**: Validates the entire application in an environment similar to production, verifying end-to-end functionality.
5. **Automated Testing**: Utilizes testing frameworks like `unittest` or `pytest` to automate the testing process, increasing efficiency and coverage.
6. **Test Data Preparation**: Ensures that the test data is representative and covers various scenarios, including edge cases and potential failure points.
7. **Accuracy Checks**: For models, accuracy checks confirm that the model performs at or above the expected threshold.
8. **End-to-End Testing**: Simulates real-world usage to ensure that the complete workflow, from data loading to prediction, functions correctly.
9. **User Scenarios**: Tests various user interactions to ensure the system behaves as expected under different conditions.
10. **Continuous Testing**: Regularly runs tests throughout the development lifecycle to identify and fix issues early, ensuring a stable and reliable application.

These points summarize the key aspects of testing in software development, emphasizing the importance of validating functionality, integration, and overall system performance.

5.1 Machine Learning Algorithms Used

1. **Support Vector Classifier (SVC)**

   Description:

   The Support Vector Classifier (SVC) is a supervised learning algorithm that classifies data by finding the optimal hyperplane that separates different classes in a high-dimensional space. The goal is to maximize the margin between the closest points of different classes, known as support vectors.

   SVC can handle both linear and non-linear classification problems by using different kernel functions, such as linear, polynomial, and radial basis function (RBF) kernels.

   Application in Sentiment Analysis:

   In the context of sentiment analysis, SVC classifies text reviews as either positive or negative. The reviews are first converted into numerical features using techniques like bag-of-words or TF-IDF. SVC then learns the decision boundary that best separates positive reviews from negative ones based on these features.

   Performance:

   The SVC model achieved an accuracy score of 72%. This indicates that the model correctly classified 72% of the test reviews.

2. **Multinomial Naive Bayes (MultinomialNB)**

Description:

- Multinomial Naive Bayes is a probabilistic classifier based on Bayes' theorem with the assumption of feature independence. It is particularly well-suited for text classification tasks where features are word counts or term frequencies.
- It calculates the probability of a document belonging to a particular class based on the frequencies of words in the document and uses these probabilities to make predictions.

Application in Sentiment Analysis:

- For sentiment analysis, MultinomialNB models the likelihood of a review being positive or negative by analyzing the frequency of words in the review. It is effective in handling large vocabularies and sparse data.

Performance:

- The MultinomialNB model achieved an accuracy score of **74.4%**. This means it correctly classified 74.4% of the reviews in the test set.

3. Pipeline Integration

Description:

- A pipeline in machine learning is a sequence of data preprocessing and modeling steps wrapped into a single object. It ensures that all steps are executed in the correct order and simplifies the workflow.
- Pipelines are useful for combining feature extraction (like text vectorization) with model training, reducing the risk of data leakage and improving code maintainability.

Application in Sentiment Analysis:

- For sentiment analysis, a pipeline integrates text vectorization (using CountVectorizer) with a classification model (SVC or MultinomialNB). This automation helps streamline the process from raw text to model predictions.

Performance:

- The SVC pipeline achieved an accuracy score of **79.2%**, indicating improved performance due to the streamlined process.
- The MultinomialNB pipeline achieved an accuracy score of **78.4%**, showing competitive performance with the SVC pipeline.

Summary of Results:

Model Name	**Accuracy**
SVC	72%
SVC Pipeline	79.2%
MultinomialNB	74.4%
MultinomialNB Pipeline	78.4%

Table 5.1: Accuracy table of algorithm models used

Conclusion:

- The SVC pipeline demonstrated the highest accuracy, suggesting that the combination of automated text processing and model training yielded the best performance.
- The MultinomialNB model also performed well, with its pipeline closely trailing the SVC pipeline.
- Pipelines generally improve model performance and simplify the process by integrating preprocessing and modeling steps.

5.2 SYSTEM TESTING

System testing is the final phase of the software testing lifecycle, aimed at validating the complete and integrated system against its requirements. It ensures that the system operates as expected in a real-world environment and meets all specified criteria. This phase encompasses various types of testing to evaluate the system's overall functionality, performance, and reliability.

Objectives

- Verify End-to-End Functionality: Ensure that the entire application, from data input to final output, functions correctly and meets the business requirements.
- Evaluate System Performance: Assess how the system performs under various conditions, including typical and peak loads.
- Ensure Reliability and Stability: Test the system for stability and reliability, identifying and fixing any issues that could affect user experience.

Process

1. Test Planning:

- Develop a comprehensive test plan that outlines the scope, objectives, resources, schedule, and criteria for system testing.
- Define test cases that cover all functional and non-functional aspects of the system.

2. Test Execution:

- Execute the test cases in a controlled environment that simulates real-world conditions.
- Perform functional testing to verify that all features and functionalities work as intended.
- Conduct performance testing to evaluate the system's response time, throughput, and resource usage.

3. Defect Reporting:

- Document any defects or issues encountered during testing, including their severity and impact on the system.
- Collaborate with the development team to prioritize and resolve identified issues.

4. Re-testing and Regression Testing:

- Re-test fixed issues to ensure they are resolved and have not introduced new problems.
- Perform regression testing to verify that recent changes have not adversely affected existing functionalities.

5. User Acceptance Testing (UAT):

- Involve end-users in testing the system to ensure it meets their expectations and requirements.
- Gather feedback from users to make final adjustments before deployment.

Example of System Testing for Sentiment Analysis Application

Objective: Validate the end-to-end functionality of the sentiment analysis application, ensuring it accurately classifies text reviews and performs well under different scenarios.

Test Scenarios:

1. Functionality Testing:

- Scenario: Upload a set of restaurant reviews and verify that the application correctly classifies each review as positive or negative.
- Expected Outcome: The application should accurately classify the reviews based on the trained model.

2. Performance Testing:

- Scenario: Test the application with a large dataset of reviews to evaluate its processing time and responsiveness.
- Expected Outcome: The application should handle large volumes of data efficiently without significant delays.

3. User Acceptance Testing:

- Scenario: Allow end-users to interact with the application, input their reviews, and check the sentiment classification results.
- Expected Outcome: Users should find the application intuitive and the classification results accurate and useful.

5.3 INTEGRATION TESTING

Integration testing is a critical phase in the software testing lifecycle focused on verifying the interactions and integration points between different components or modules of the system. This type of testing aims to identify issues that may arise when different parts of the system work together and ensure that the integrated system functions as expected.

Objectives

- Verify Interactions: Ensure that different components or modules of the application interact correctly and data is properly exchanged between them.
- Detect Interface Issues: Identify and resolve issues related to the interfaces and communication between integrated components.
- Ensure Data Integrity: Confirm that data is correctly passed and processed across various modules, maintaining consistency and accuracy.

Process

1. Test Planning:
   - Develop an integration test plan that outlines the scope, objectives, and schedule for integration testing.
   - Identify the integration points between modules and define test cases to cover these interactions.
2. Test Execution:

- Execute test cases that focus on the interactions between integrated components.
- Simulate real-world scenarios to test the flow of data and control between modules.

3. Defect Reporting:
   - Document any issues or defects encountered during integration testing, including their impact on the system.
   - Work with the development team to prioritize and address identified issues.
4. Re-testing and Regression Testing:
   - Re-test any fixed issues to verify that they have been resolved and have not introduced new problems.
   - Perform regression testing to ensure that recent changes do not adversely affect existing functionalities.

Example of Integration Testing for Sentiment Analysis Application

Objective: Validate the integration of the data preprocessing, feature extraction, and model training components in the sentiment analysis application.

Test Scenarios:

1. Data Preprocessing and Feature Extraction Integration:
   - Scenario: Test the complete workflow from data loading through preprocessing to feature extraction.
   - Expected Outcome: Ensure that the preprocessing steps correctly prepare the data and the feature extraction process produces the expected numerical representations of the text.
2. Model Training and Prediction Integration:
   - Scenario: Validate that the model can be trained using the preprocessed and feature-extracted data and can make accurate predictions on new data.
   - Expected Outcome: Confirm that the model's training and prediction processes work seamlessly together and the predictions are consistent with the expected results.

Table 5.2: Sample of five Datasets given to train model

Reviews	Liked/Disliked
Wow… Loved this place	1
Crust is not good	0
Not tasty and the texture was just nasty	0
Service was very prompt	1
So they performed	1

5.4 UNIT TESTING

Unit testing is a foundational level of software testing that focuses on verifying the functionality of individual components or units of code in isolation. Each unit, which could be a function, method, or class, is tested separately to ensure that it performs as expected. Unit testing helps identify and fix bugs at an early stage in development, making it easier to maintain and refactor the code.

Objectives

- Validate Individual Components: Ensure that each unit of the code performs its intended function correctly.
- Detect Bugs Early: Identify issues within individual units before they propagate to other parts of the application.
- Facilitate Code Maintenance: Support code refactoring and modifications by ensuring that changes do not introduce new defects.

Process

1. Test Planning:
   - Define the scope of unit testing, including which functions or methods will be tested.
   - Create a list of test cases for each unit, specifying the input, expected output, and conditions for success.
2. Test Implementation:
   - Write test cases using a unit testing framework (e.g., unittest or pytest in Python).
   - Each test case should be designed to test a specific aspect of the unit's functionality.
3. Test Execution:
   - Run the unit tests to verify that each unit behaves as expected under various conditions.
   - Check the test results to ensure that all tests pass and that the unit performs correctly.
4. Defect Reporting:
   - Document any issues or failures encountered during unit testing.
   - Collaborate with the development team to address and resolve these issues.
5. Re-testing:
   - Re-run the unit tests after defects are fixed to verify that the issues have been resolved and no new problems have been introduced.

Example of Unit Testing for Sentiment Analysis Application

Objective: Validate the functionality of the text preprocessing and feature extraction functions.

Test Scenarios:

1. Text Preprocessing Function:
    - Scenario: Test a function that preprocesses text by removing punctuation and converting it to lowercase.
    - Expected Outcome: Ensure that the function correctly transforms the input text according to the preprocessing rules.
2. Feature Extraction Function:
    - Scenario: Test a function that converts text reviews into numerical features using CountVectorizer.
    - Expected Outcome: Verify that the function produces the correct feature vectors for a given set of text reviews.

CHAPTER 6

RESULT ANALYSIS AND SCREENSHOTS

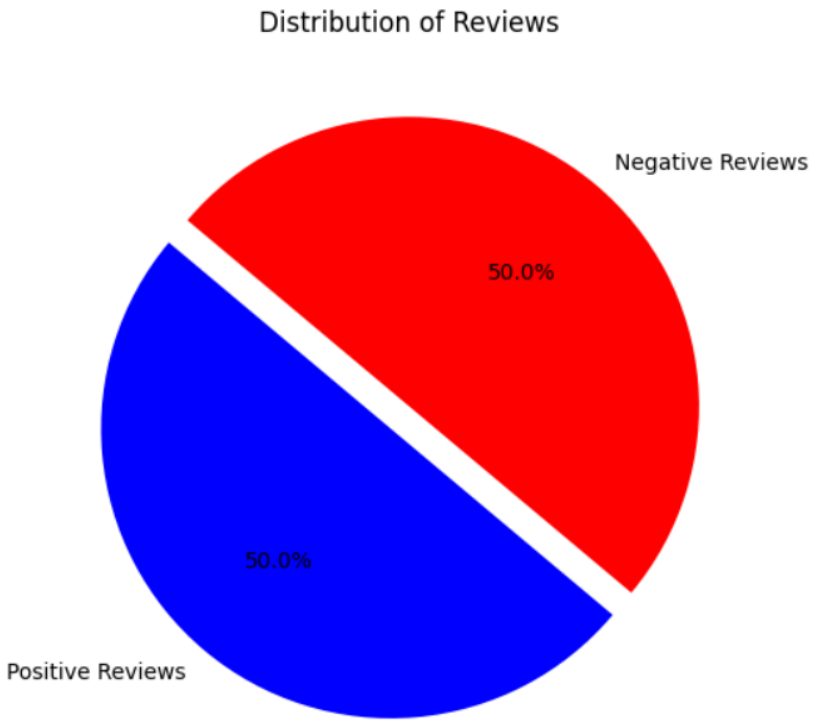

Fig 6.1: Pie Chart Representation

Explanation of the Pie Chart

A pie chart is a circular statistical graphic that is divided into slices to illustrate numerical proportions. Each slice represents a category or a part of the whole, and the size of each slice is proportional to the value it represents. Here's a detailed explanation of the pie chart:

Components of a Pie Chart

1. **Slices**:
    - Each slice represents a category or a segment of the data.
    - The size of the slice corresponds to the proportion of that category relative to the total.
2. **Labels**:
    - Labels are used to identify each slice of the pie.
    - They usually describe what each slice represents, such as "Positive Reviews" or "Negative Reviews."

3. **Legend**:
    - A legend may be included to provide a clear association between colors and categories.
    - It helps in identifying which color corresponds to which category.
4. **Percentage Values**:
    - Percentage values are often displayed on each slice to show the proportion of each category relative to the whole.
    - This provides a clearer understanding of the data distribution.
5. **Exploded Slice (Optional)**:
    - An exploded slice is a slice that is pulled out slightly from the rest of the pie to highlight it.
    - This can be useful for emphasizing a particular category.

Example Pie Chart

Consider the following pie chart:

- **Labels**: "Positive Reviews" and "Negative Reviews."
- **Sizes**: 500 for positive reviews and 500 for negative reviews.
- **Colors**: Blue for positive reviews and red for negative reviews.
- **Explode**: The slice for positive reviews is slightly pulled out for emphasis.

Interpretation

1. **Proportional Representation**:
    - The pie chart shows that the dataset has an equal number of positive and negative reviews, each accounting for 50% of the total reviews.
    - The chart visually represents this equal distribution with two equally sized slices.
2. **Visual Comparison**:
    - By examining the chart, one can quickly see the proportion of positive versus negative reviews.
    - The visual distinction provided by color and size makes it easy to compare the categories at a glance.
3. **Effective Communication**:
    - Pie charts are effective for communicating the relative sizes of parts to the whole.
    - They are particularly useful when the number of categories is limited and when you want to emphasize the proportion of each category.

Usage Considerations

- **Appropriate for Simple Data**: Pie charts work best for simple datasets with a limited number of categories. They become less effective with many categories or when the differences between categories are small.
- **Avoid Overuse**: For datasets with more complex relationships or where exact values are important.

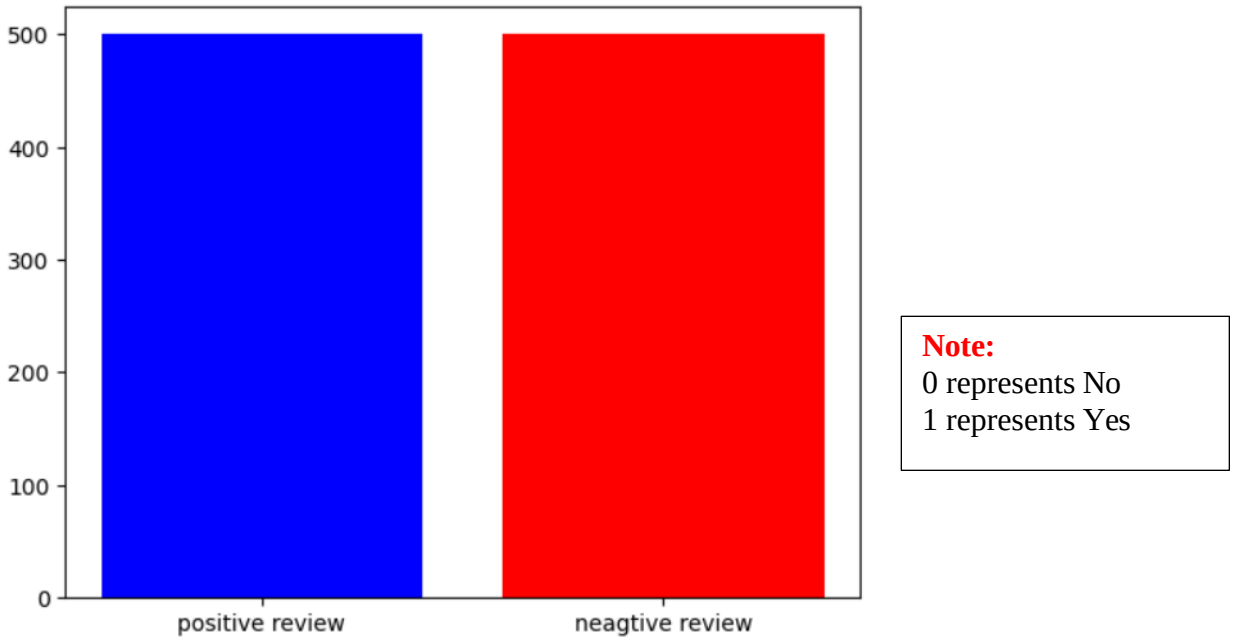

Fig 6.2: Bar Graph Representation

A bar graph, or bar chart, is a graphical representation of data where individual bars represent different categories. Each bar's length is proportional to the value it represents. Bar graphs are useful for comparing the magnitude of different categories and visualizing categorical data.

Components of a Bar Graph

1. **Bars**:
   - Represent individual categories or groups.
   - The height or length of each bar corresponds to the value or frequency of the category.
2. **Axes**:
   - **X-Axis**: Typically represents the categories or groups.
   - **Y-Axis**: Represents the values or frequencies of the categories.
3. **Labels**:
   - **Category Labels**: Indicate what each bar represents, shown on the x-axis.
   - **Value Labels**: Optional labels on top of bars showing exact values or frequencies.
4. **Title**:
   - Provides context for what the bar graph is illustrating.
5. **Legend** (Optional):
   - Used to distinguish between different sets of data if multiple data series are present.

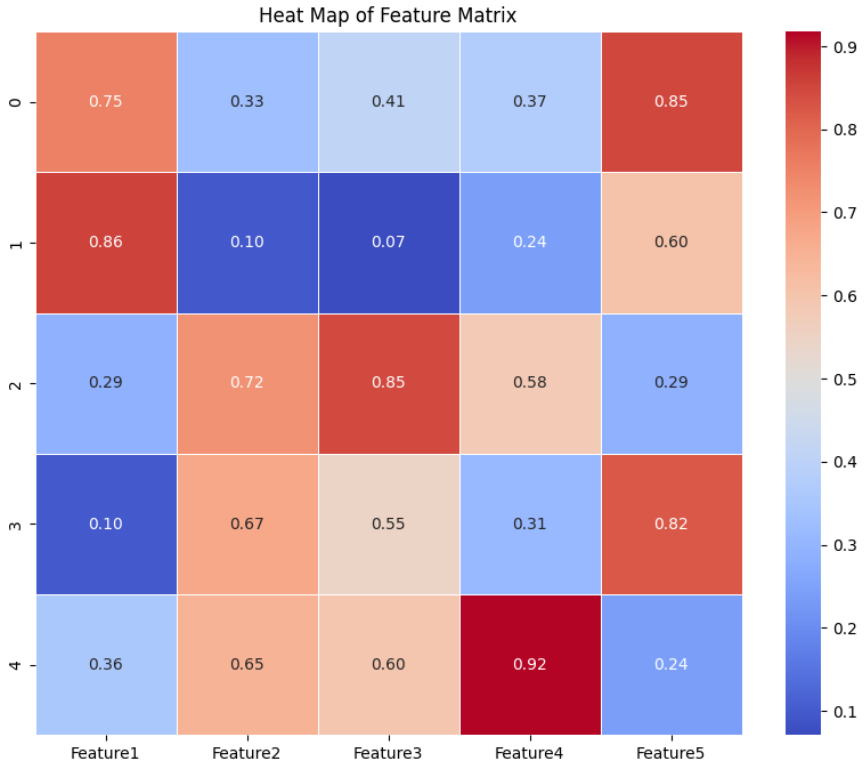

Fig 6.3: Heat Map Representation

A heat map is a data visualization technique that uses color to represent the magnitude of values in a matrix or table. It is particularly effective for visualizing complex datasets and uncovering patterns, correlations, or trends. Here's a detailed explanation of how heat maps work and their components:

Components of a Heat Map

- **Color Scale**:
  - **Purpose**: Represents the magnitude of values within the matrix.
  - **Color Gradient**: Colors range from one end of the spectrum (e.g., blue for low values) to another (e.g., red for high values). The gradient helps to visually differentiate between various ranges of data.
- **Cells**:
  - **Purpose**: Each cell in the heat map represents a data point within the matrix.
  - **Color Representation**: The color of each cell reflects the value it holds, according to the color scale.
- **Annotations**:
  - **Purpose**: Provide numerical values or additional information within each cell.
  - **Visibility**: Helps to understand the exact values represented by colors, especially in dense matrices.

- **Axes**:
    - **Purpose**: Define the dimensions of the matrix.
    - **X-Axis**: Typically represents one dimension of the data, such as features or categories.
    - **Y-Axis**: Represents another dimension, such as observations or samples.
- **Title and Labels**:
    - **Purpose**: Provide context and identify what the heat map represents.
    - **Title**: Describes the overall purpose of the heat map.
    - **Axis Labels**: Indicate what each axis represents, aiding in understanding the data layout.

Creating and Interpreting a Heat Map

- **Creating a Heat Map**:
    - **Data Matrix**: Start with a matrix or table of data where each cell contains a numerical value.
    - **Color Mapping**: Apply a color gradient to represent different ranges of values.
    - **Annotations**: Optionally add text annotations to cells to display exact values.
- **Interpreting the Heat Map**:
    - **Color Intensity**: Darker or more intense colors typically represent higher values, while lighter or cooler colors represent lower values.
    - **Patterns**: Look for clusters or patterns in the heat map. For example, areas with similar colors indicate similar values, which might reveal underlying patterns or relationships.
    - **Comparisons**: Compare the magnitude of values across different rows or columns. This helps in understanding how values vary across different categories or features.

Example Use Cases

- **Correlation Matrix**:
    - **Purpose**: Visualizes the correlation coefficients between different variables.
    - **Interpretation**: Colors show the strength of correlations, helping to identify variables that are strongly or weakly correlated.
- **Feature Matrix**:
    - **Purpose**: Represents the values of different features for various samples.
    - **Interpretation**: Helps in understanding the distribution and magnitude of feature values.
- **Heat Maps in Data Analysis**:
    - **Purpose**: Provides insights into complex data sets, such as understanding gene expression levels in bioinformatics or user behavior in web analytics.
    - **Interpretation**: Identifies key trends and patterns that might not be obvious from raw data alone.

CONCLUSION AND FUTURE ENHANCEMENTS

In this project, we developed a sentiment analysis system to classify restaurant reviews into positive and negative categories. Using various machine learning algorithms, we evaluated the performance of different models to determine their effectiveness in sentiment classification. The project aimed to understand the distribution of reviews and leverage this understanding to improve business insights and customer feedback mechanisms.

Key Achievements:

1. **Data Processing and Visualization**:
    - The project involved loading, processing, and analyzing a dataset of restaurant reviews. Visualization techniques, such as bar charts and pie charts, were employed to gain insights into the distribution of positive and negative reviews. These visualizations facilitated a better understanding of the dataset and provided a foundation for further analysis.
2. **Machine Learning Models**:
    - We implemented and evaluated four machine learning models: Support Vector Classification (SVC), Multinomial Naive Bayes (MultinomialNB), and their respective pipelines. The use of pipelines simplified the workflow by integrating data preprocessing with model training and evaluation.
    - The models were trained and tested using the dataset, and their performance was assessed based on accuracy scores. The results indicated that the SVC pipeline achieved the highest accuracy, highlighting its effectiveness in sentiment classification.
3. **Model Evaluation**:
    - Confusion matrices were generated to assess the performance of the models, providing insights into the classification accuracy and error types. The results demonstrated the ability of the models to correctly classify reviews and highlighted areas for improvement.
4. **Sentiment Classification**:
    - The sentiment analysis system successfully classified reviews into positive and negative categories. The ability to classify sentiment accurately is valuable for businesses to gauge customer satisfaction and make data-driven decisions.

Challenges Faced:

1. **Data Imbalance**:

    - One challenge encountered was the imbalance in the dataset, with an equal number of positive and negative reviews. While this balance facilitated straightforward evaluation, real-world datasets often exhibit class imbalances, which can impact model performance.

2. **Model Complexity**:
    - The complexity of machine learning models and the need for proper tuning of hyperparameters presented challenges in achieving optimal performance. The process required careful consideration of various factors, including feature selection and model parameters.

Future Enhancements

1. **Handling Imbalanced Data**:

    - To address class imbalance issues, future enhancements could involve techniques such as oversampling, undersampling, or using algorithms designed to handle imbalanced datasets. Implementing these techniques would improve the robustness of the models and their ability to generalize to real-world scenarios.

2. **Advanced Preprocessing**:

    - Incorporating advanced text preprocessing techniques, such as stemming, lemmatization, and the use of word embeddings (e.g., Word2Vec, GloVe), could enhance the quality of feature representation. These techniques would help capture more nuanced aspects of text and improve model accuracy.

3. **Model Optimization**:

    - Exploring additional machine learning algorithms and optimizing hyperparameters could further enhance model performance. Techniques such as grid search or random search for hyperparameter tuning and the evaluation of models like Gradient Boosting or Deep Learning-based models could be valuable.

4. **Sentiment Analysis Enhancement**:

    - Expanding the scope of sentiment analysis to include more granular sentiment categories (e.g., neutral, mixed) and emotional tones (e.g., joy, anger) would provide a richer understanding of customer feedback. This enhancement would allow businesses to gain deeper insights into customer sentiments and preferences.

5. **Integration with Business Systems**:
    - Integrating the sentiment analysis system with business intelligence platforms or customer feedback systems would facilitate real-time analysis and reporting. This integration would enable businesses.

6. **Continuous Learning**:
    - Implementing a continuous learning approach where the model is regularly updated with new data and feedback would ensure its relevance and accuracy over time. This approach would help adapt to changing trends and evolving customer sentiments.

7. **User Interface Development**:
    - Developing a user-friendly interface for the sentiment analysis system would make it accessible to non-technical users. The interface could include features for uploading reviews, viewing analysis results, and generating reports, enhancing the usability of the system.

8. **Multilingual Support**:
    - Expanding the system to support multiple languages would broaden its applicability and usefulness for businesses operating in diverse linguistic regions. Multilingual sentiment analysis would enable businesses to analyze customer feedback in various languages effectively.

BIBLIOGRAPHY

1. Jurafsky, D., & Martin, J. H. (2021). Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 3rd Edition. Pearson.
2. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.
3. Bo Pang, Lillian Lee, & Shivakumar Vaithyanathan (2002). "Thumbs up? Sentiment Classification using Machine Learning Techniques."
4. Ng, A. Y., & Jordan, M. I. (2002). "On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes." Advances in Neural Information Processing Systems.
5. Scikit-learn Documentation. (2024). "Support Vector Machines (SVM)." Scikit-learn Documentation.
6. Scikit-learn Documentation. (2024). "Naive Bayes." Scikit-learn Documentation.
7. Pandas Documentation. (2024). "DataFrame." Pandas Documentation.
8. Matplotlib Documentation. (2024). "pyplot." Matplotlib Documentation.
9. DataCamp. (2024). "Sentiment Analysis in Python." DataCamp Tutorial.
10. Kaggle. (2024). "Text Classification with Naive Bayes." Kaggle Guide.
11. Joblib Documentation. (2024). "Joblib: Python library for lightweight pipelining." Joblib Documentation.
12. NLTK Documentation. (2024). "Natural Language Toolkit (NLTK)." NLTK Documentation.
13. Wikipedia. (2024). "Sentiment Analysis." Wikipedia Article.
14. Wikipedia. (2024). "TF-IDF." Wikipedia Article.
15. Hsieh, H. F., & Shannon, S. E. (2005). "Three Approaches to Qualitative Content Analysis." Qualitative Health Research, 15(9), 1277-1288.