PROBLEM STATEMENT - 11

Intel Products Sentiment Analysis from Online Reviews

A PROJECT REPORT

Submitted by

KANISHK REDDY
[RA2211003010870] kk9032@srmist.edu.in

SUJAL MANGESH LIMJE
[RA2211003010821] sl5387@srmist.edu.in

Under the Guidance of

Debdyut Hazra

Project Mentor, Intel

Dr. M. Safa

Assistant Professor, Department of Networking And Communications

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR -603203

JULY 2024

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr T.V.Gopal**, for his invaluable support.

We express our gratitude to **Debdyut Hazra**, Project Mentor, Intel for supporting us during the critical stages of the project work.

We wish to thank **Dr Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department**, Dr. M. Pushpalatha**, Professor**,** Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We register our immeasurable thanks to our Faculty Advisor**, Dr. D.Shiny Irene**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. M. Safa,** Assistant Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship.

We sincerely thank the Department of Computing Technologies, Networking and Communications Department, staff and students, SRM Instituteof Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

KOOTURU KANISHK REDDY [RA2211003010870]

SUJAL MANGESH LIMJE [RA2211003010821]

ABSTRACT

User reviews and feedbacks stands as the bedrock for large companies and enterprises to improve their product and to maintain their positioning globally as the best company for particular segment. While the feedbacks helps to identify issues and enhance the product, companies gain competitive advantage and helps to provide a personalized experience by identifying trends and managing negative feedbacks. Review Analyzer, designed to analyse Intel product reviews from e-commerce websites such as Amazon and Flipkart, is a vital tool which advances prior technologies to capitalize over consumer reviews and gain insights to maintain competitive edge over other companies. The application obtains the user reviews directly from marketplaces and is sent to a trained BERT model to categorize reviews into positive, neutral, and negative sentiments to understand overall customer sentiment towards a product. The identified problem between consumers and company is lack of conversation which is bridged by user reviews. The Review Analyzer focuses on advancing User Sentiment Analysis to provide insights, trends and recommendations by scraping pages with APIs. The application is rigorously tested for accuracy in sentiment analysis, correct extraction of reviews, smooth interaction between frontend and backend, and user experience.

TABLE OF CONTENTS

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Background and Significance

The rapid advancement of technology has led to a surge in consumer electronics, with a significant focus on processors that drive the performance desktop devices. Intel Core processors have established a reputation for their exceptional performance, efficiency, and reliability, making them a preferred choice for a wide range of applications, from everyday computing tasks to high-performance gaming and professional workloads. As a result, they are integral components in many devices used by consumers and professionals alike.

With the proliferation of online shopping and e-commerce platforms, users have the opportunity to share their experiences and opinions about products through reviews. These reviews are rich sources of information, capturing user satisfaction, issues, and expectations in real-world scenarios. For Intel, understanding this feedback is crucial for several reasons:

1. **Product Development**: Insights from user reviews help in identifying strengths and weaknesses in current products. This information is invaluable for research and development teams to innovate and enhance future processor generations.
2. **Customer Satisfaction**: By analysing sentiments expressed in reviews, Intel can gauge overall customer satisfaction and address any prevalent issues, ensuring a better user experience.
3. **Market Competitiveness**: Reviews often include comparisons with competitor products. Understanding these sentiments helps Intel position its products more effectively in the market.
4. **Feature Enhancement**: Users often suggest features or improvements in their reviews. These suggestions can guide the development of new features that meet user needs and expectations.
5. **Trend Analysis**: Analysing the trends in sentiments over time allows Intel to track changes in user perception and respond proactively to shifting market dynamics.

Given the volume and complexity of these reviews, manual analysis is impractical. Sentiment analysis, which leverages natural language processing and machine learning techniques, offers a

systematic approach to extracting meaningful insights from large datasets. By categorizing reviews into positive, negative, and neutral sentiments, and identifying key themes and features associated with these sentiments, Intel can gain a comprehensive understanding of user feedback.

Project aims to harness the power of sentiment analysis to analyse online reviews of Intel Core processors, providing actionable insights that can drive product improvement and innovation. The focus is on desktop and mobile processors across several generations, ensuring a broad and deep understanding of user sentiments related to these critical components.

1.2 Objective

The primary objective of the project is to perform sentiment analysis on online reviews of Intel Core processors. This involves classifying sentiments into positive, negative, and competition-related categories, as well as understanding future expectations of users and tech experts. The project aims to achieve the following:

1. **Classify Sentiments through Classification Methodology**: Utilize advanced natural language processing (NLP) techniques to classify sentiments in the reviews. Trained BERT model is employed to ensure comprehensive sentiment categorization.
2. **Conduct Exploratory Data Analysis (EDA)**: Perform EDA on various dimensions such as Stock Keeping Units (SKUs), timelines, technical features, and ratings. This analysis will help identify patterns and trends, providing a deeper understanding of the data.
3. **Extract and Scrape Marketplaces using API**: Identify the URL , obtain webpage contents using scraperAPI , scraping relevant information and storing in proper formats which can be further analysed.
4. **Provide Recommendations on Key Improvements**: Based on the analysis, offer actionable recommendations for product improvements that align with user reviews and their future expectations. This will help Intel enhance its product offerings and address user needs more effectively.
5. **Identify and Analyse Trends in Sentiments Over Time**: Track and analyse the changes in sentiments over time to understand the evolving perceptions of Intel Core processors. This temporal analysis will help in identifying long-term trends and shifts in user sentiment.

1.3 Software Requirements Specification

1.3.1 Hardware Requirements

- **Compute Resources:**

Adequate computational resources are necessary to train and deploy deep learning models efficiently. A system with a dedicated GPU (Graphics Processing Unit) is recommended for accelerated model training.

Recommended: NVIDIA GPU with CUDA support.

- **Internet Connectivity:**

The system requires a stable internet connection for accessing the python libraries, npm modules, and any additional resources such as trained model.

1.3.2 Software Requirements

- **Operating System:**

  The system is platform-independent and can be run on Windows, macOS, or Linux operating systems.

- **Python Environment:**

  Python 3.6 or later is required for running the system.

  Recommended: Set up a virtual environment in ANACONDA environment.

- **Dependencies:**

  Install required Python libraries.

  Install required NPM modules and React libraries.

- **Additional Libraries:**

  Depending on the specific implementation, additional libraries for image processing, data augmentation, and visualization may be required.

- **Integrated Development Environment (IDE):**

  Choose a preferred IDE for code development. Recommended options include VS Code, Jupyter Notebooks or Google Colab for an integrated environment.

- **Web Browser:**

  Web Browser such as Google Chrome or Mozilla Firefox is required to run the review analyser web application locally.

- **Scraper Account:**

  A scraperapi account is required to access and download the webpage content for effortless scraping. Make sure to have a scraperapi account and follow API setup guidelines to contain API_KEY in .env file.

1.3.3 Data Requirements

- **Collecting Data:**

  User Reviews of specific product present on Amazon and Flipkart are scraped and stored in real-time to gain continuous consumer reviews at that particular time.

- **ALL Reviews Dataset:**

  This dataset contains around 2000 reviews, which are collected from various marketplaces such as Amazon, Flipkart and eBay, have been analysed for sentiments, trends and insights.

- **Data Preprocessing:**

  Perform necessary data preprocessing steps, including stop words removal, punctuation removal, special characters removal, tokenization, convert text to lowercase, convert special characters to their closest ASCII representation, Lemmatization and number removal.

CHAPTER 2

LITERATURE SURVEY

2.1 Web Scraping

Web scraping, also known as web harvesting or web data extraction, involves the automated retrieval of web content for analysis and integration. Early methods of web scraping relied heavily on pattern matching using regular expressions. These methods, while useful for simple and consistent web pages, are limited by the complexity and variability of HTML structures. Laender, A. H. F., et al. provide a comprehensive overview of these early techniques and highlight their limitations in handling dynamic and complex web content.

As web technologies evolved, DOM (Document Object Model) parsing became a more robust method for web scraping. The development of specialized web scraping tools and frameworks has significantly advanced the field. Popular tools such as Beautiful Soup, Scrapy, and Selenium provide powerful features for data extraction. Beautiful Soup is known for its simplicity and ease of use, particularly for beginners. Web scraping has a wide range of applications across various industries. In the e-commerce sector, companies use web scraping to monitor competitors' prices, gather customer reviews, and analyse market trends.

2.2 Introduction to Natural Language Processing

Fig 2.1: Natural Language Processing Pipeline

With the rapid development of the Internet and social media, a vast amount of text data containing rich emotional information has been generated. Sentiment analysis, a technique aimed at identifying and processing people's emotional tendencies, emotions, and perspectives, has become a significant research area in the field of natural language processing (NLP).

It plays a crucial role in various fields, such as product review analysis, public opinion monitoring,

and consumer behaviour prediction, helping companies and organizations better understand public sentiment and make more informed decisions.

2.3 Sentiment Analysis Techniques

Fig 2.2: Sentiments of Text : Positive , Neutral, Negative

There are various techniques present for analysing sentiments. Lexicon-based approaches use predefined lists of words (lexicons) where each word is associated with a sentiment score (positive, negative, or neutral). It utilizes sentiment dictionaries like SentiWordNet, AFINN, or the VADER (Valence Aware Dictionary and sEntiment Reasoner) lexicon to assign sentiment scores to words in a text. Machine learning approaches involve training algorithms on labelled datasets to automatically classify sentiment. Naive Bayes, Support Vector Machines (SVM), Decision Trees and Random Forests are some of the methods.

Deep learning approaches leverage neural networks, particularly for capturing complex patterns in large datasets. Convolutional Neural Networks (CNNs) are effective in capturing local features and patterns in text data. Recurrent Neural Networks (RNNs) are Suitable for sequential data, capturing dependencies between words. LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) are variants of RNNs that handle long-term dependencies and mitigate vanishing gradient problems. Transformers include Advanced models like BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) that leverage self-attention mechanisms for context-aware sentiment analysis.

In the realm of sentiment analysis research and application, the advent of deep learning technology has introduced new methods and perspectives for processing complex text data. Compared to traditional rule-based or machine learning methods, deep learning can automatically learn features from large-scale datasets, significantly enhancing the accuracy and efficiency of sentiment classification and analysis. Among these technologies, the BERT (Bidirectional Encoder Representations from Transformers) model, an advanced deep learning framework, excels by

capturing deep semantic information in text through pre-training. It has achieved breakthrough results in multiple NLP tasks.

2.4 BERT model

The BERT model has garnered significant attention in the field of natural language processing, primarily due to its bidirectional Transformer architecture, which enables a more comprehensive understanding of complex relationships within language contexts. This deep understanding allows BERT to perform exceptionally well in tasks such as sentiment analysis, accurately capturing the emotional tendencies of text. However, while BERT's effectiveness has been demonstrated across various fields, its optimal use and best practices for specific sentiment analysis tasks remain to be fully explored.

Through detailed analysis and experimental validation of the BERT model, we aim to uncover its strengths and potential limitations in identifying different emotional tendencies in text, thereby providing guidance for future research and practice.

2.4.1. BERT vs. Traditional Machine Learning Models

Traditional Models: These include algorithms like Logistic Regression, Support Vector Machines (SVM), Naive Bayes, and Decision Trees. They rely on hand-crafted features and Bag of Words (BoW) or TF-IDF representations.

Advantages of BERT:

- Contextual Understanding: BERT understands the context of a word in a sentence, while traditional models treat words independently.
- Pre-trained Knowledge: BERT is pre-trained on a large corpus, which means it already has a vast amount of linguistic knowledge.
- Fine-Tuning: BERT can be fine-tuned on a specific dataset, leading to better performance on domain-specific tasks.

2.4.2. BERT vs. RNN (Recurrent Neural Networks)

RNN Models: These include basic RNNs, LSTMs (Long Short-Term Memory), and GRUs (Gated Recurrent Units). They process sequences of data but often struggle with long-range dependencies.

Advantages of BERT:

- Bidirectionality: BERT processes text in both directions (left-to-right and right-to-left), capturing context more effectively than unidirectional RNNs.
- Parallelization: BERT, being a Transformer model, allows for parallel processing of data, unlike RNNs which process data sequentially.
- Long-Range Dependencies: BERT handles long-range dependencies better due to its self-attention mechanism.

2.4.3. BERT vs. CNN (Convolutional Neural Networks)

CNN Models: Typically used for image processing but also adapted for text classification tasks. They capture local features through convolutional layers.

Advantages of BERT:

- Global Context: BERT captures the global context of sentences, whereas CNNs focus more on local features.
- Pre-trained Embeddings: BERT uses pre-trained embeddings that provide a richer understanding of language compared to static word embeddings used in CNNs.
- Transfer Learning: BERT's architecture is better suited for transfer learning, leading to superior performance on specific tasks after fine-tuning.

2.4.4. BERT vs. GloVe/Word2Vec Embeddings

GloVe/Word2Vec: These are pre-trained word embeddings that capture semantic similarity but are context-independent.

Advantages of BERT:

- Contextual Embeddings: BERT generates dynamic, context-dependent embeddings, which means the same word can have different representations based on its usage in a sentence.
- Transfer Learning: BERT can be fine-tuned on specific tasks, leveraging its pre-trained knowledge for better performance.
- Deep Understanding: BERT's deeper architecture captures complex language nuances better than the relatively shallow GloVe/Word2Vec models.

2.4.5. BERT vs. Transformer Models (e.g., GPT)

GPT (Generative Pre-trained Transformer): Similar architecture to BERT but typically unidirectional (left-to-right).

Advantages of BERT:

- Bidirectionality: BERT's bidirectional approach captures context from both directions, enhancing understanding.
- Task-Specific Fine-Tuning: BERT is designed for fine-tuning on specific tasks, which often leads to better performance on those tasks compared to GPT.

2.4.6 Key Metrics and Performance

- Accuracy: BERT generally achieves higher accuracy in sentiment analysis tasks due to its contextual understanding.
- F1-Score: BERT shows improved precision and recall, reflected in higher F1-scores compared to other models.
- Flexibility: BERT's architecture allows it to be adapted for various NLP tasks beyond sentiment analysis, showcasing its versatility.

2.4.7 Conclusion for Choosing the Correct Model

BERT's bidirectional architecture, pre-training on vast text corpora, ability to capture context, and flexibility for fine-tuning make it superior to many traditional and deep learning models for sentiment analysis. Its ability to understand the nuances of language and context leads to more accurate and reliable sentiment predictions.

While BERT excels in many areas, it is computationally expensive. For resource-constrained environments, lightweight models like DistilBERT or other optimized versions of BERT may be considered, offering a trade-off between performance and computational efficiency.

This comparative study highlights BERT’s strengths and why it is often the model of choice for sentiment analysis in modern NLP applications.

CHAPTER 3

DATA COLLECTION

3.1 Naive Method

The process to get webpage content is by using requests method. This python module allows to send http requests to get webpage content. This is a widely used method but sometimes this technique may not work due to captchas, rate limiting actions taken by the marketplace servers to protect their data from web crawlers. This can be solved by changing headers constantly after specific time period but repetitive requests may lead to IP blocking .This affects the application productivity and loses the data to analyze reviews and get insights.

3.2 Using API

To overcome these problems, APIs (Application Programming Interface) can be used to bypass rate limiting system, or solve captchas as they are specially designed to scrape data and content from websites. But they also have some limitations based on credits and can be paid or free depending upon usage. In this project we used Scraper API (https://www.scraperapi.com/) which is almost free and easy to use to collect webpage content from public websites.

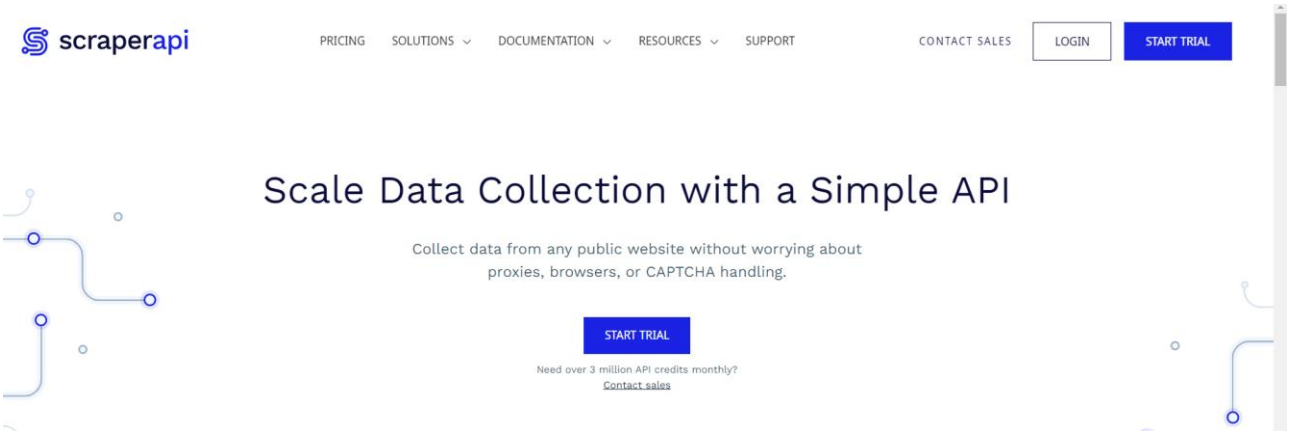

Fig 3.1: Scraper API website

3.3 BeautifulSoup (bs4)

Beatiful Soup is python library that makes it easy to scrape information from web pages. It sits atop an HTML or XML parser, and helps for iterating, searching, and modifying the parse tree. Beautiful Soup provides a better way to extract information from css selectors that are present in the webpage content.

3.4 SelectorLib Chrome Extension

The css selectors which contain the specific information about the reviews can be obtained by manually picking these selectors by inspecting the site or can be obtained by using selectorlib chrome extension which can be installed from chrome web store in Google Chrome web browser. It is a free to use extension which helps in extracting specific selectors.

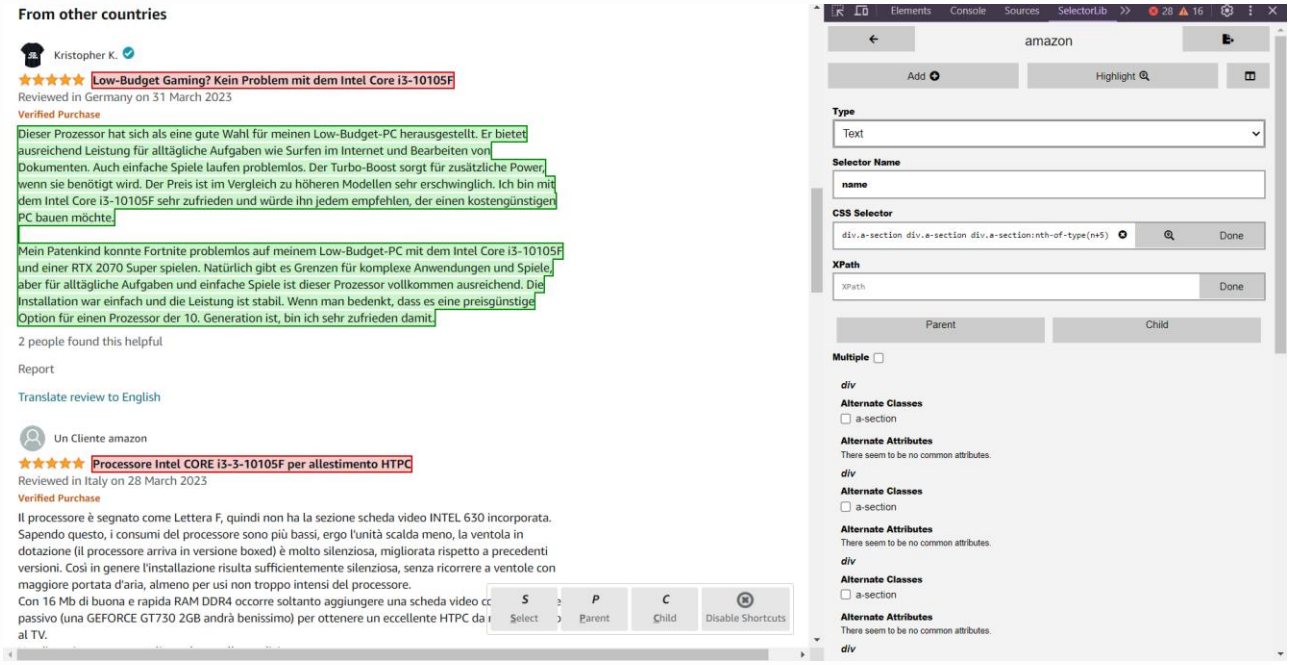

Fig 3.2: SelectorLib Extension

3.5 Data Scraper - Easy Web Scraping Chrome Extension

DataMiner is also a data extraction tool that lets you scrape any HTML web page. It is a free extension that directly runs on browser and no external scripts are required to scrape data. We can create a recipe that helps to automatically extract data from given URLs. We can extract tables and lists from any page and upload them to Google Sheets or Microsoft Excel. With this tool we can export web pages into XLS, CSV, XLSX or TSV files (.xls .csv .xlsx .tsv)

3.6 Storing Data

The extracted data can be stored in csv, json or xml format using python scripts. This makes the process easier to further analyse the data according to the extracted data.

3.7 Exploratory Data Analysis (Introduction)

Exploratory Data Analysis (EDA) is a crucial step in understanding and summarizing the main characteristics of a dataset. It helps in uncovering patterns, identifying relationships, and detecting anomalies or outliers. The analysis can be done for obtaining the following relationships present in the dataset:

- Display the first few rows and basic information about the dataset to understand the outline
- Check for missing values
- To get details for number of reviews for each SKU
- Distribution of Ratings by number of stars
- Distribution of Reviews by Date
- Polarity of reviews for each SKU
- Advanced review clustering distribution

CHAPTER 4

DATA PREPROCESSING

4.1 Convert Text to Lowercase:

Convert all text to lowercase to ensure consistency. This helps in treating words like "Good" and "good" as the same.

```
text = text.lower()
```

4.2 Remove Punctuation:

Remove punctuation marks (e.g., periods, commas, exclamation marks) to focus on the words themselves.

```
import string
text = text.translate(str.maketrans('', '', string.punctuation))
```

4.3 Remove Special Characters:

Remove any special characters that might not contribute to the meaning of the text.

```
import re
text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
```

4.4 Tokenization:

Split the text into individual words or tokens.

```
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
```

4.5 Remove Stop Words:

Remove common words that do not add significant meaning (e.g., "and", "the", "is").

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
tokens = [word for word in tokens if word not in stop_words]
```

4.6 Convert Special Characters to Closest ASCII Representation:

Convert special characters (e.g., accented characters) to their closest ASCII representation.

```
import unicodedata
def remove_accents(input_str):
    nfkd_form = unicodedata.normalize('NFKD', input_str)
    return u"".join([c for c in nfkd_form if not unicodedata.combining(c)])
```

```
tokens = [remove_accents(word) for word in tokens]
```

4.7 Lemmatization:

Convert words to their base or root form (e.g., "running" to "run").

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
tokens = [lemmatizer.lemmatize(word) for word in tokens]
```

4.8 Remove Numbers:

Remove numerical values from the text.

```
tokens = [word for word in tokens if not word.isdigit()]
```

CHAPTER 5

SENTIMENT ANALYSIS METHODOLOGY

5.1 Introduction

The primary purpose of this sentiment analysis project is to evaluate user reviews of Intel processors, identifying positive, negative, and neutral sentiments. By understanding user sentiments, we can derive insights into user satisfaction, common issues, and areas for improvement in Intel products.

Various techniques such as Random Forests, Naïve Bayes, SVM, XGBoost, Kmeans, DBScan, CNN, VADER and RoBERTa were used but to achieve higher accuracy in sentiment classification, we employed BERT (Bidirectional Encoder Representations from Transformers) model, a cutting-edge model in natural language processing. BERT's significance lies in its bidirectional approach to language modelling, which enables it to understand the context of a word based on its surrounding words. This deep understanding of context makes BERT particularly effective for tasks like sentiment analysis, where nuanced interpretations of text are crucial. By using BERT, we aim to achieve more accurate and contextually aware sentiment predictions compared to traditional models unlike other models which are not specialised for this task.

5.2 Data Preparation

After preprocessing the text data (lowercasing, removing punctuation, tokenization, etc.), two datasets were prepared with 500 balanced distribution and 2000 skewed distribution to serve across different models. Models such as Random Forest, SVMa and XGBoost typically perform well when classes are balanced, as they rely on the distribution of data points across classes for accurate classification. Models like VADER, Naïve Bayes, KMeans, DBScan techniques often rely on statistical properties or predefined lexicons that can benefit from larger datasets. Models like CNN and deep learning models, especially transformers like BERT and RoBERTa, benefit from larger datasets to learn nuanced patterns and dependencies in text. However, balance is less critical compared to traditional models due to their ability to learn complex relationships across classes.

Then these datasets were split into training (80%), validation (20%) sets to ensure an effective training and evaluation process. This split was chosen to provide enough data for training while reserving sufficient data for unbiased evaluation.

5.3 Model Selection

Various Models were used to perform sentiment analysis on the dataset. Figure 5.1 talks about different sentiment analysis techniques used in natural language processing.

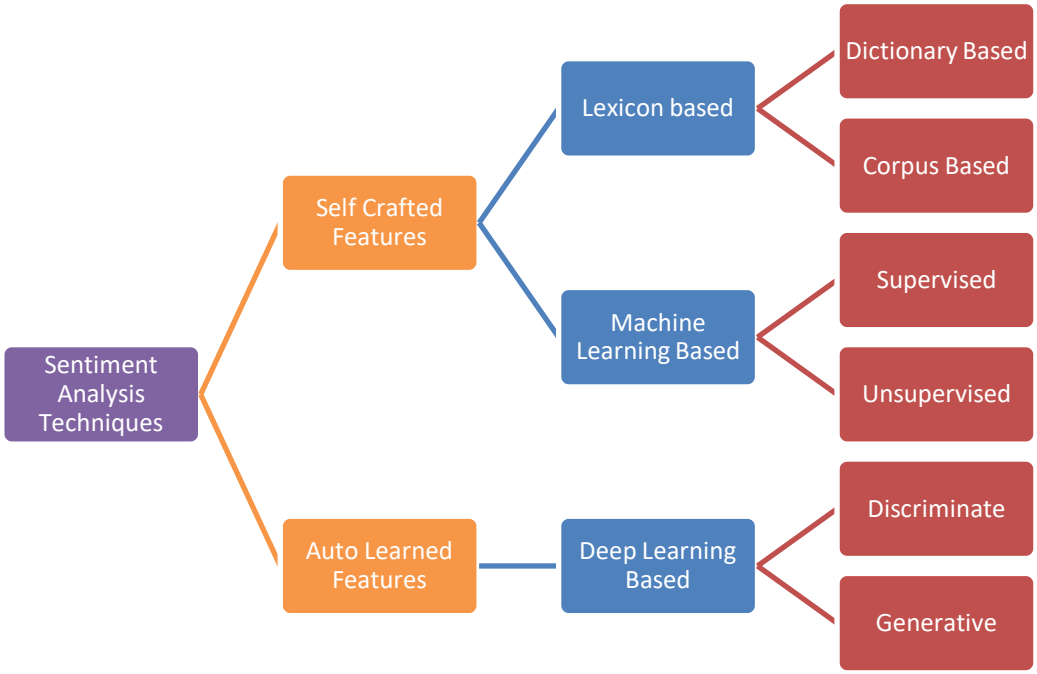

Fig 5.1: Sentiment Analysis Techniques

5.3.1 Naïve Bayes

- Feature Extraction: Convert text to numerical features using techniques like Bag of Words or TF-IDF.
- Model Training: Train a Naïve Bayes classifier on the training data.
- Prediction: Predict the labels for the test data.
- Evaluation: Evaluate the model performance using metrics such as accuracy, precision, recall, and F1-score.

5.3.2 Random Forest

- Model Training: Train a Random Forest classifier on the training data.
- Prediction: Predict the labels for the test data.
- Evaluation: Evaluate the model performance using metrics such as accuracy,

precision, recall, and F1-score.

5.3.3 SVM

- Model Training: Train a Support Vector Machine classifier on the training data.
- Prediction: Predict the labels for the test data.
- Evaluation: Evaluate the model performance using metrics such as accuracy, precision, recall, and F1-score.

5.3.4 XGBoost

- Model Training: Train an XGBoost classifier on the training data.
- Prediction: Predict the labels for the test data.
- Evaluation: Evaluate the model performance using metrics such as accuracy, precision, recall, and F1-score.

5.3.5 DBScan

- DBScan Clustering: Apply DBScan clustering algorithm to the feature matrix.
- Cluster Analysis: Analyse the clusters and interpret the results.

5.3.6 KMeans

- K-Means Clustering: Apply K-Means clustering algorithm to the feature matrix.
- Cluster Analysis: Analyse the clusters and interpret the results.

5.3.7 Neural Networks

- Model Training: Train a neural network model (CNN) on the training data.
- Prediction: Predict the labels for the test data.
- Evaluation: Evaluate the model performance using metrics such as accuracy, precision, recall, and F1-score.

5.3.8. VADER

- Sentiment Analysis: Apply VADER sentiment analysis to the text data.
- Result Interpretation: Interpret the sentiment scores and classify the text into positive, negative, or neutral sentiments.

5.3.9. RoBERTa

- Model Training: Fine-tune the RoBERTa model on the training data.
- Prediction: Predict the labels for the test data.
- Evaluation: Evaluate the model performance using metrics such as accuracy, precision, recall, and F1-score**.**

5.3.10 BERT

- Model Training: Fine-tune the BERT model on the training data.
- Prediction: Predict the labels for the test data.
- Evaluation: Evaluate the model performance using metrics such as accuracy, precision, recall, and F1-score.

BERT model was selected for its ability to capture contextual relationships in text, which is crucial for accurate sentiment analysis. We used the BERT-Base variant pre-trained on a large corpus of English text. The choice of BERT-Base was motivated by its balance of performance and computational efficiency.

5.4 Implementation Steps

We used the Hugging Face transformers library to implement the BERT model. The implementation steps include:

1. **Loading the Pre-trained BERT Model**: The BERT-Base model, pre-trained on a large corpus, was loaded to leverage its extensive language understanding capabilities.
2. **Adding a Classification Layer**: A new classification layer was added to the BERT model to predict sentiment classes (positive, negative, neutral).
3. **Fine-tuning**: The entire model, including the pre-trained BERT and the new classification layer, was fine-tuned on our sentiment analysis dataset. Fine-tuning allows the model to adapt specifically to the sentiment analysis task.

5.5 Training Process

The model was trained with the following hyperparameters:

- **Batch Size**: 16
- **Epochs**: 10

The training loop included:

- **Loss Function**: Cross-entropy loss was used to optimize the model's predictions.
- **Evaluation Metrics**: The model's performance was monitored using accuracy and F1-score.
- **Validation Strategy**: Early stopping based on validation loss was employed to prevent overfitting. If the validation loss did not improve for a set number of epochs, training was stopped to avoid overfitting.

5.6 Evaluation

The model's performance was evaluated using several metrics to ensure a comprehensive assessment:

- **Accuracy**: The overall correctness of the model's predictions.
- **Precision and Recall**: Metrics to evaluate the model's performance for each sentiment class.
- **F1-score**: The harmonic mean of precision and recall, providing a balanced measure of the model's performance.
- **Confusion Matrix**: A confusion matrix was used to analyse the distribution of true positive, true negative, false positive, and false negative predictions across different sentiment classes.
- **Cross-Validation**: Cross-validation techniques were employed to ensure robust evaluation and mitigate the risk of overfitting to the training set.

The training process was conducted on an NVIDIA GPU (CUDA enabled), which expedited the computations and allowed for more efficient model training. This hardware acceleration was crucial for handling the computational demands of fine-tuning a large model like BERT.

CHAPTER 6

IMPLEMENTATION

6.1 Tools and Applications Used

VS Code, Jupyter notebook, Python environment (Anaconda environment is recommended) and required packages and modules are necessary to run the application. Firstly, clone the repository from GitHub using

`git clone “repo-link”`

in terminal. Change directory using

`cd “/path/to/folder”`

Then install npm modules and python libraries using

`npm install` and

`pip install -r “py-requirements.txt`

The application works on react and flask. Use

`npm start`

to run the application.

6.2 Exploratory Data Analysis

The data is analysed with various techniques which is essential for understanding and summarizing main characteristics of dataset.

6.2.1 Load and Inspect the Dataset

Inspect the structure, columns, and some sample rows to understand its content.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset (replace 'data.csv' with your actual file path)
df = pd.read_csv('pre_processed_data.csv')
df['date'] = pd.to_datetime(df['date'], format = "%d-%m-%Y")
```

```
# Display the first few rows and basic information about the dataset
print(df.info())
print("\nGeneral Statistics:")
print(df.describe())
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2207 entries, 0 to 2206
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   sno       2207 non-null   int64
 1   product   2207 non-null   object
 2   title     2207 non-null   object
 3   content   2207 non-null   object
 4   date      2207 non-null   datetime64[ns]
 5   author    2207 non-null   object
 6   rating    2207 non-null   int64
 7   category  2207 non-null   object
dtypes: datetime64[ns](1), int64(2), object(5)
memory usage: 138.1+ KB
None

General Statistics:
               sno                           date       rating
count  2207.000000                           2207  2207.000000
mean   1104.077028  2021-09-28 07:16:30.122337792     4.594472
min       1.000000            2010-08-01 00:00:00     1.000000
25%     552.500000            2020-05-01 00:00:00     5.000000
50%    1104.000000            2022-08-01 00:00:00     5.000000
75%    1655.500000            2023-10-01 00:00:00     5.000000
max    2208.000000            2024-06-01 00:00:00     5.000000
std     637.373554                            NaN     1.003180
```

	sno	product	title	content	date	author	rating	category
0	1	Intel Core I5 12400F 12 Gen Generation Desktop...	great processor	processor power efficient develop android apps...	2024-04-01	ARUN	5	i5
1	2	Intel Core I5 12400F 12 Gen Generation Desktop...	got	try get check ranking detail section buying an...	2024-05-01	Ayush	5	i5
2	3	Intel Core I5 12400F 12 Gen Generation Desktop...	highest price performance processor gaming	upgrade performance gain huge use cooler maste...	2024-03-01	Hruaia	5	i5
3	4	Intel Core I5 12400F 12 Gen Generation Desktop...	nice processor	best aaaaaaaa gaming multitasking processor pr...	2024-04-01	Mr.GeAr	5	i5
4	5	Intel Core I5 12400F 12 Gen Generation Desktop...	title	run well productivity task core cpu minimum re...	2024-01-01	Joy Mukherjee	5	i5

```
Features : ['sno' 'product' 'title' 'content' 'date' 'author' 'rating' 'category']
```

Fig 6.2.1: Dataset Evaluation

Figure 6.21. states the basic information about the dataset and also tells about the general statistics.

6.2.2 Check for missing values

```
print(df.isnull().sum())
```

```
sno         0
product     0
title       0
content     0
date        0
author      0
rating      0
category    0
dtype: int64
```

Fig 6.2.2: Missing Values

Figure 6.2.2 states the output for missing values present in dataset.

6.2.3 Graphs

Various graphs are implemented to get detailed analysis of dataset.

```
sku_counts = df['category'].value_counts()
plt.figure(figsize=(12, 6))
sku_counts.plot(kind='bar')
plt.title('Number of Reviews per SKU')
plt.xlabel('SKU')
plt.ylabel('Number of Reviews')
plt.xticks(rotation=90)
plt.show()
```

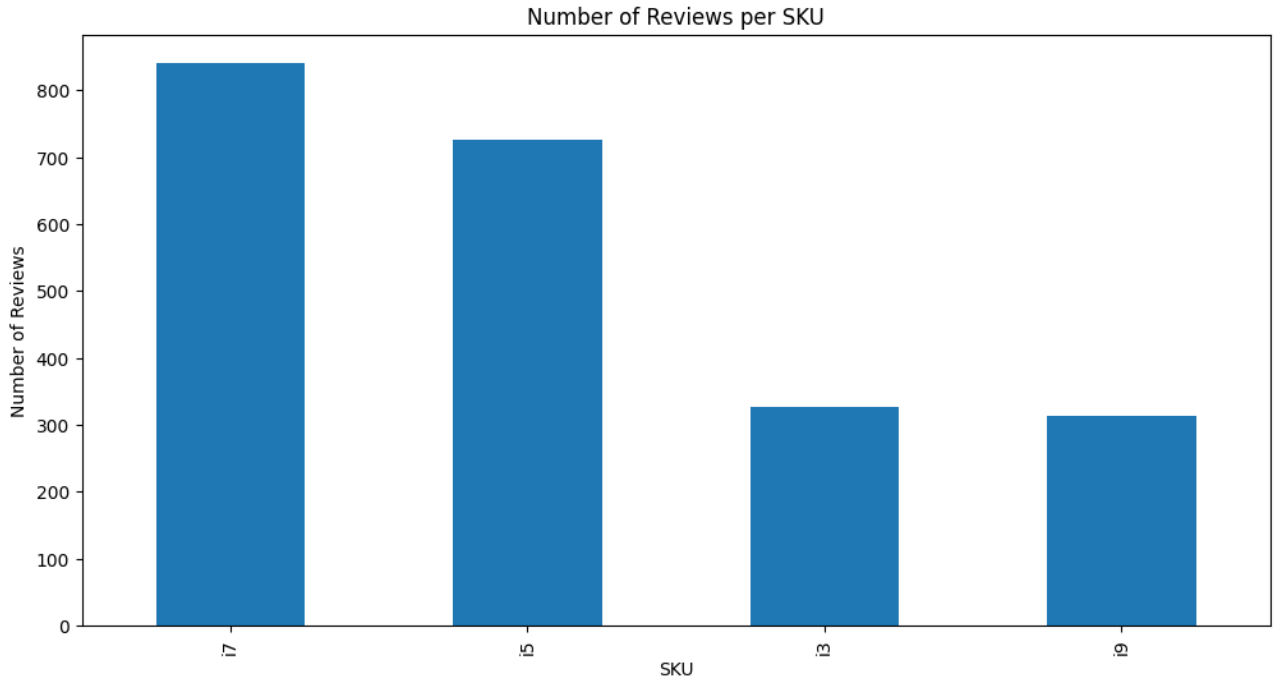

Fig 6.2.3: SKUs vs Reviews

Figure 6.2.3 represents the relation between the number of reviews and the SKUs in form of bar chart.

```
plt.figure(figsize=(8, 5))
sns.histplot(df['rating'], bins=5, kde=False)
plt.title('Distribution of Ratings')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()
```

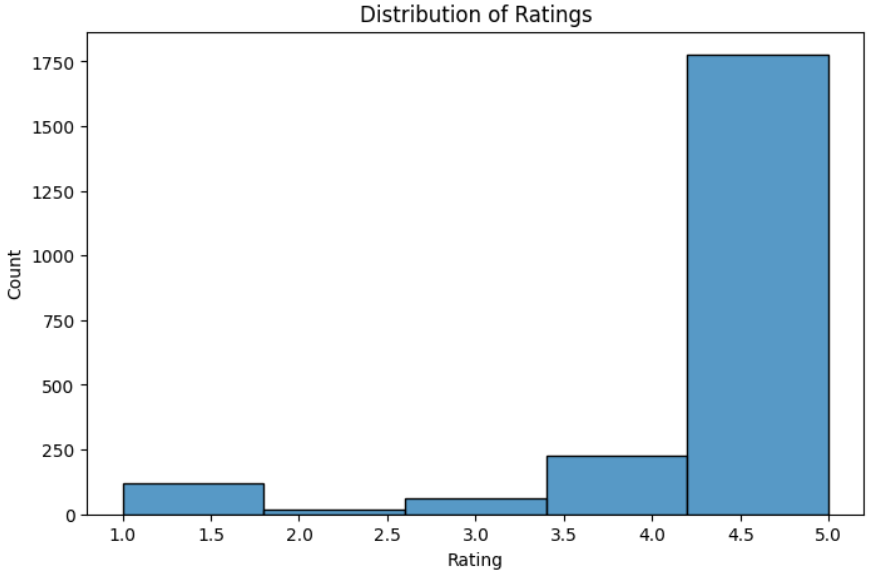

Fig 6.2.4: Rating vs Reviews

Figure 6.2.4 represents relation between the counts of reviews vs Rating received.

```
# Distribution of review dates
plt.figure(figsize=(12, 6))
df['date'].hist(bins=10)
plt.title('Distribution of Review Dates')
plt.xlabel('Date')
plt.ylabel('Number of Reviews')
plt.show()
```

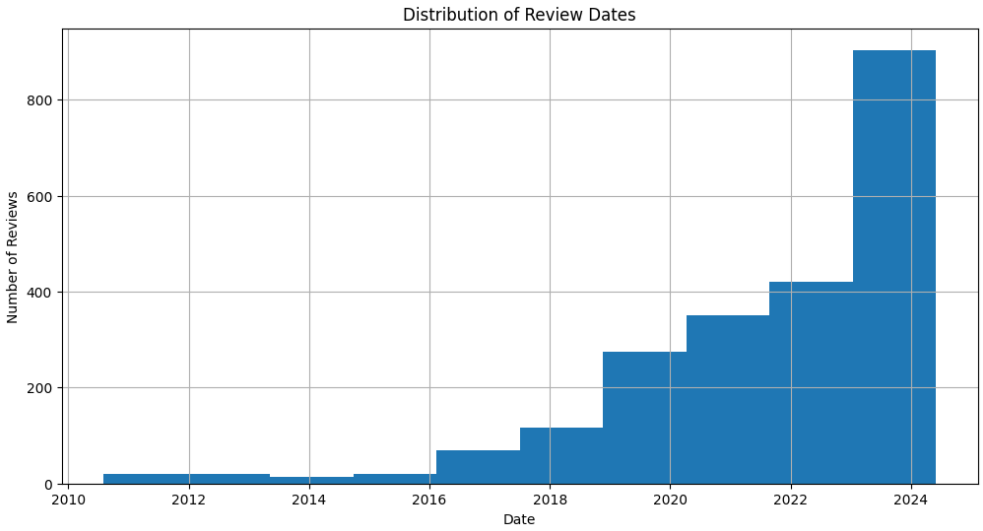

Fig 6.2.5: Distribution of Reviews by Date

Figure 6.2.5 depicts the distribution of reviews with respect to date.

```
import numpy as np
from textblob import TextBlob
df['polarity']=df['lemmatized'].apply(lambda
  x:TextBlob(x).sentiment.polarity)

product_polarity_sorted=pd.DataFrame(df.groupby('category')['polarit
  y'].mean().sort_values(ascending=True))

plt.figure(figsize=(16,8))
plt.xlabel('Polarity')
plt.ylabel('Products')
plt.title('Polarity of Different Amazon Product Reviews')
polarity_graph=plt.barh(np.arange(len(product_polarity_sorted.index)
  ),product_polarity_sorted['polarity'],color='purple',)

# Writing product names on bar
for bar,product in zip(polarity_graph,product_polarity_sorted.index):

  plt.text(0.005,bar.get_y()+bar.get_width(),'{}'.format(product),va
  ='center',fontsize=11,color='white')

# Writing polarity values on graph
for                          bar,polarity                          in
  zip(polarity_graph,product_polarity_sorted['polarity']):

  plt.text(bar.get_width()+0.001,bar.get_y()+bar.get_width(),'%.3f'%
  polarity,va='center',fontsize=11,color='black')

plt.yticks([])
plt.show()
```

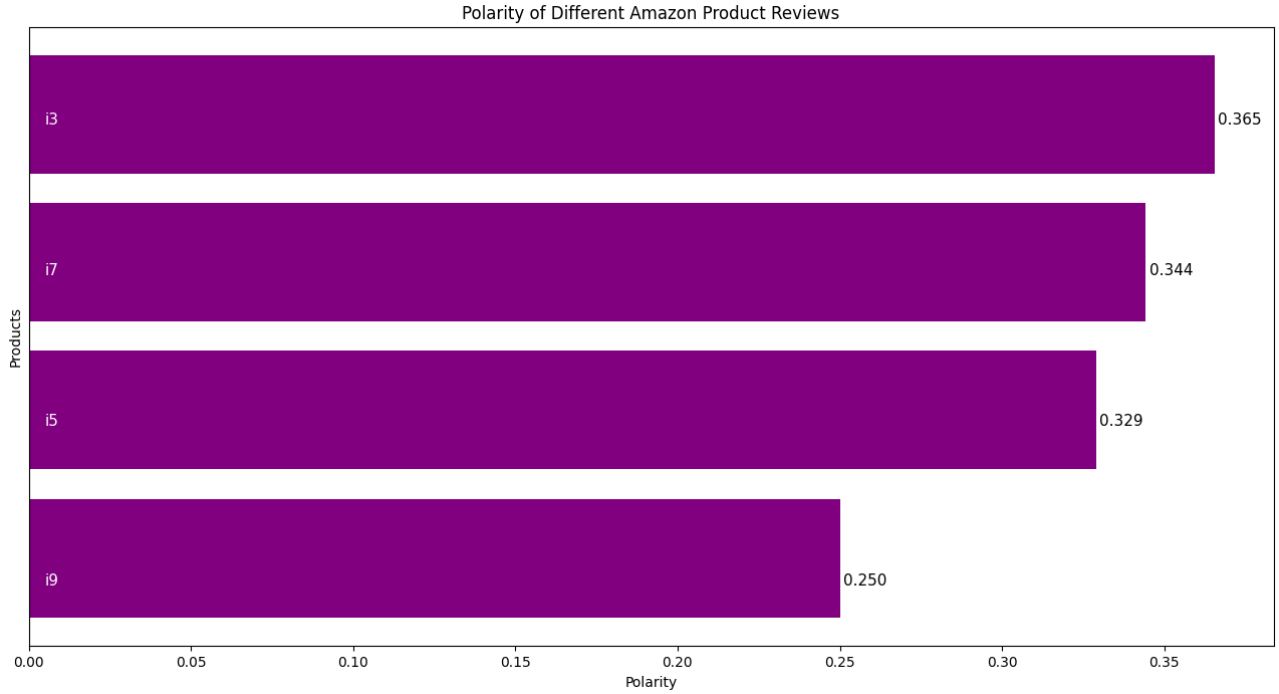

Fig 6.2.6: Polarity of reviews vs SKUs

Figure 6.2.6 represents the analysed polarity of reviews for each of the SKU.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# Example TF-IDF vectorization
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(df['content'])

# K-Means clustering
kmeans = KMeans(n_clusters=5, random_state=42)
clusters = kmeans.fit_predict(X)

# Add cluster labels to DataFrame
df['Cluster'] = clusters

# Visualize clusters using PCA
pca = PCA(n_components=2)
scatter_plot_points = pca.fit_transform(X.toarray())
colors = ["r", "b", "c", "y", "m"]

x_axis = [o[0] for o in scatter_plot_points]
y_axis = [o[1] for o in scatter_plot_points]
plt.figure(figsize=(10,10))
plt.scatter(x_axis, y_axis, c=[colors[d] for d in clusters])
plt.show()
```

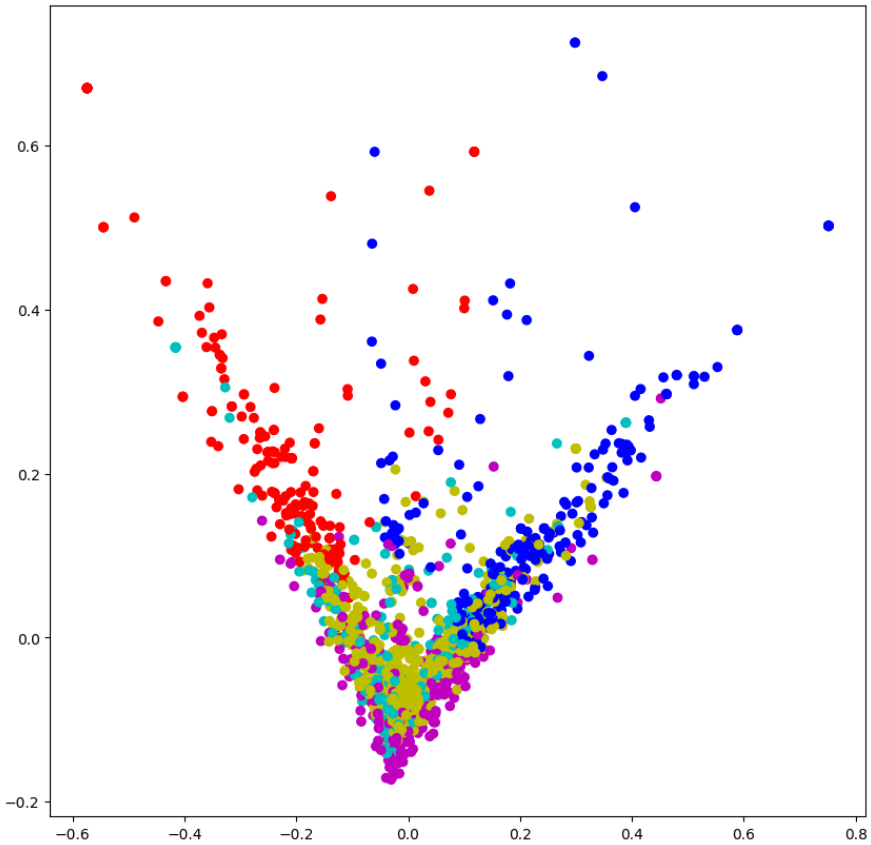

Fig 6.2.7: Feature clustered distribution

Figure 6.2.7 depicts the advanced clustering of feature extracted from review.

6.3 Naïve Bayes

Implementation of Naïve Bayes technique

```
def map_sentiment(rating):
    if rating >= 4.0:
        return 'positive'
    elif rating <= 2.0:
        return 'negative'
    else:
        return 'neutral'

df['sentiment'] = df['rating'].apply(map_sentiment)
X = df['content']
y = df['sentiment']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

tfidf_vectorizer = TfidfVectorizer(max_features=1000)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

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

y_pred = nb_classifier.predict(X_test_tfidf)

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

6.4 SVM

Implementation of support vector machine technique

```
neutral_upsampled = resample(neutral,
                             replace=True,     # sample with replacement
                             n_samples=len(positive),    # to match
majority class
                             random_state=42) # reproducible results
negative_upsampled = resample(negative,
                              replace=True,     # sample with
replacement
                              n_samples=len(positive),    # to match
majority class
                              random_state=42) # reproducible results

# Combine majority class with upsampled minority classes
upsampled = pd.concat([positive, neutral_upsampled, negative_upsampled])

# Text data for training
X = upsampled['content']
y = upsampled['sentiment']
# Split the data into training, validation, and test sets
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y,
```

```
test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_full,
y_train_full, test_size=0.2, random_state=42)

# Convert text data to TF-IDF features
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_val_tfidf = vectorizer.transform(X_val)
X_test_tfidf = vectorizer.transform(X_test)
# Train the SVM model
model = SVC(kernel='linear', random_state=42)
model.fit(X_train_tfidf, y_train)

# Validate the model
y_val_pred = model.predict(X_val_tfidf)
val_accuracy = accuracy_score(y_val, y_val_pred)
val_report = classification_report(y_val, y_val_pred)
val_conf_matrix = confusion_matrix(y_val, y_val_pred)

# Test the model
y_test_pred = model.predict(X_test_tfidf)
test_accuracy = accuracy_score(y_test, y_test_pred)
test_report = classification_report(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)
```

6.5 XGBoost

Implementation of XGBoost technique

```
# Split the data into training, validation, and test sets
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_full,
y_train_full, test_size=0.2, random_state=42)

# Convert text data to TF-IDF features
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_val_tfidf = vectorizer.transform(X_val)
X_test_tfidf = vectorizer.transform(X_test)

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

model_xgb = XGBClassifier()
model_xgb.fit(X_train_tfidf, y_train_encoded)

y_pred_train = model_xgb.predict(X_train_tfidf)
y_pred_test = model_xgb.predict(X_test_tfidf)
```

6.6 Random Forests

Implementation of Random Forests technique

```
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_full,
y_train_full, test_size=0.2, random_state=42)

# Convert text data to TF-IDF features
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_val_tfidf = vectorizer.transform(X_val)
X_test_tfidf = vectorizer.transform(X_test)

# Train the Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_tfidf, y_train)

# Validate the model
y_val_pred = model.predict(X_val_tfidf)
val_accuracy = accuracy_score(y_val, y_val_pred)
val_report = classification_report(y_val, y_val_pred)
val_conf_matrix = confusion_matrix(y_val, y_val_pred)

# Test the model
y_test_pred = model.predict(X_test_tfidf)
test_accuracy = accuracy_score(y_test, y_test_pred)
test_report = classification_report(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)
```

6.7 KMeans

Implementation of KMeans technique

```
tfidf_vectorizer = TfidfVectorizer(stop_words='english',
max_features=5000)
X = tfidf_vectorizer.fit_transform(data['content'])
for i in range(num_clusters):
    cluster_reviews = data[data['cluster'] == i]['content']
    print(f"Cluster {i} reviews:")
    print(cluster_reviews.head())
    print()
def assign_cluster(review, vectorizer, kmeans_model):
    review_vector = vectorizer.transform([review])
    cluster = kmeans_model.predict(review_vector)[0]
    return cluster
```

6.8 DBScan

Implementation of DBScan technique

```
tfidf_vectorizer = TfidfVectorizer(stop_words='english',
max_features=5000)
X = tfidf_vectorizer.fit_transform(data['content'])
dbscan = DBSCAN(eps=0.5, min_samples=5)
```

```
dbscan.fit(X)

# Add the cluster labels to the data
data['cluster'] = dbscan.labels_
def assign_cluster(review, vectorizer, dbscan_model):
    review_vector = vectorizer.transform([review])
    distances = dbscan_model.fit(review_vector)
    cluster = dbscan_model.labels_[0]
    return cluster
```

6.9 Random Forests

Implementation of Random Forests technique

```
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_full,
y_train_full, test_size=0.2, random_state=42)

# Convert text data to TF-IDF features
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_val_tfidf = vectorizer.transform(X_val)
X_test_tfidf = vectorizer.transform(X_test)

# Train the Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_tfidf, y_train)

# Validate the model
y_val_pred = model.predict(X_val_tfidf)
val_accuracy = accuracy_score(y_val, y_val_pred)
val_report = classification_report(y_val, y_val_pred)
val_conf_matrix = confusion_matrix(y_val, y_val_pred)

# Test the model
y_test_pred = model.predict(X_test_tfidf)
test_accuracy = accuracy_score(y_test, y_test_pred)
test_report = classification_report(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)
```

6.10 Neural Networks

```
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

X = df['content']
y = df['sentiment']

# Map sentiment labels to numeric values
label_mapping = {'positive': 2, 'neutral': 1, 'negative': 0}
y = y.map(label_mapping)

# 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=42)
```

```
# Tokenize and convert text data to sequences
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
X_train_sequences = tokenizer.texts_to_sequences(X_train)
X_test_sequences = tokenizer.texts_to_sequences(X_test)

max_sequence_length = 100
X_train_padded = pad_sequences(X_train_sequences,
maxlen=max_sequence_length, padding='post')
X_test_padded = pad_sequences(X_test_sequences,
maxlen=max_sequence_length, padding='post')

# Define model architecture
embedding_dim = 50
model = Sequential([
    Embedding(input_dim=len(tokenizer.word_index) + 1,
output_dim=embedding_dim, input_length=max_sequence_length),
    LSTM(units=128),
    Dense(units=3, activation='softmax')
])

# Compile model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train model
model.fit(X_train_padded, y_train, epochs=10, batch_size=64,
validation_split=0.1)

# Evaluate model on test data
loss, accuracy = model.evaluate(X_test_padded, y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

6.11 VADER

```
nltk.download('vader_lexicon')
sia = SentimentIntensityAnalyzer()
res = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    content = row['content']
    id = row['sno']
    res[id] = sia.polarity_scores(content)
6.3 RoBERTa

MODEL = "cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
res = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    content = row['content']
    id = row['sno']
    vader_result = sia.polarity_scores(content)
```

```
    vader_result_rename = {}
    for key, value in vader_result.items():
        vader_result_rename[f"vader_{key}"] = value
    roberta_result = polarity_scores_roberta(content)
    both = {**vader_result_rename, **roberta_result}
    res[id] = both
```

6.12 BERT

To train and fine-tune the model first we need to scrape the data, preprocess it and tokenize as we have done it in previous sections. Then the data is split into Train and Validation sets.

```
df_train, df_val = train_test_split(data, test_size=0.2, random_state=42,
stratify=data['label'])

train_data_loader = create_data_loader(df_train, tokenizer, max_len=160,
batch_size=16)

val_data_loader = create_data_loader(df_val, tokenizer, max_len=160,
batch_size=16)
```

Then this data is loaded into trainer to train the model according to following arguments.

```
Load pre-trained BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-
uncased', num_labels=3)

# Training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=10,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    evaluation_strategy="epoch"
)

# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data_loader.dataset,
    eval_dataset=val_data_loader.dataset
)

# Train the model
trainer.train()
```

The trained model is then saved and used for classifying the data into positive / negative / neutral categories. Model saves the output with probabilities and then the data is further analysed.

CHAPTER 7

MODEL DEPLOYMENT

7.1 Frontend

7.1.1 Built with React

The frontend of the application is built using React, a popular JavaScript library for building user interfaces. React allows for the creation of reusable components and efficient rendering, making it an ideal choice for developing dynamic web applications.

7.1.2 Key Components

The frontend implements several key components to facilitate user interaction and display analysis results:

- **HomePage**: Serves as the landing page as in figure 7.1, providing an overview of the application and a call-to-action button to navigate to the AnalyzerPage.

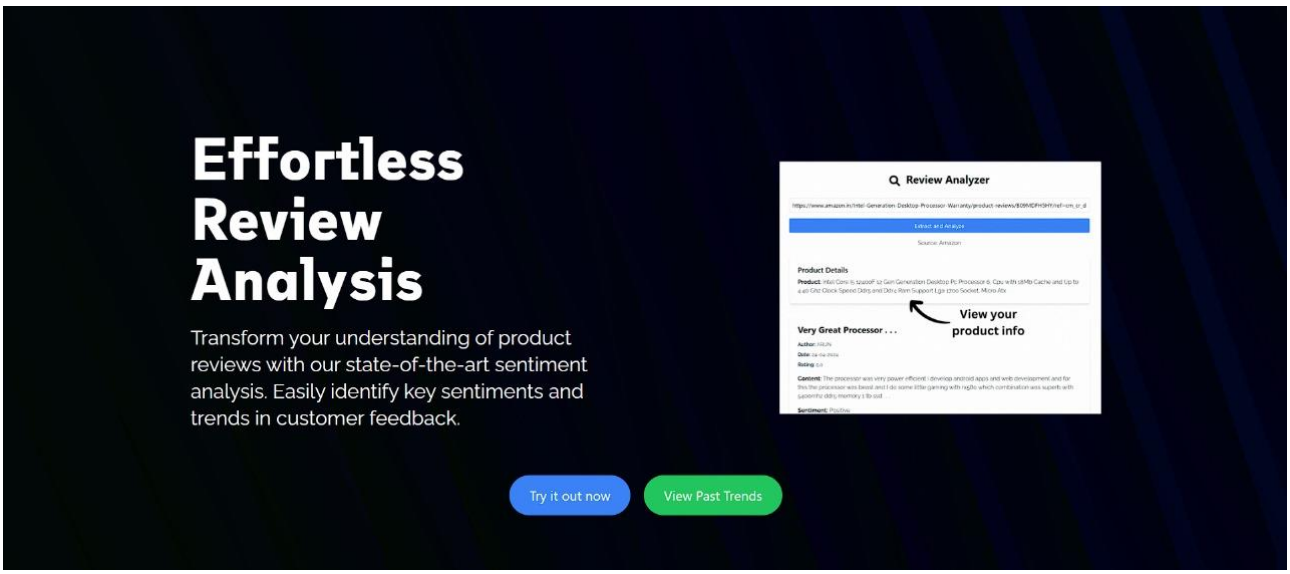

Fig 7.1 Homepage

- **AnalyzerPage**: The core component where users can input URLs of product review pages and view the analysis results. Manages state for URL input, loading status, extracted product information, reviews, and pagination. This component can be seen in figure 7.2.

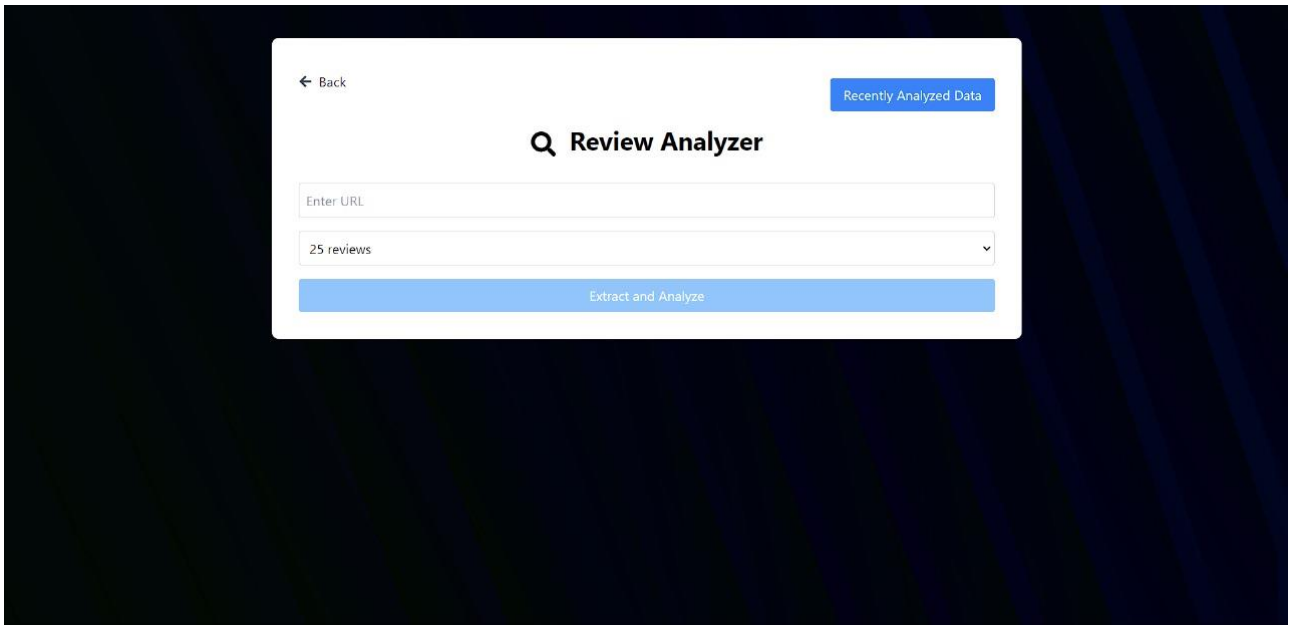

Fig 7.2 Analyser Page

- **ReviewList**: Renders a list of reviews with pagination controls. Displays details such as title, author, date, rating, content, sentiment, and suggestions for improvement. Figure 7.3 represents the Product details.

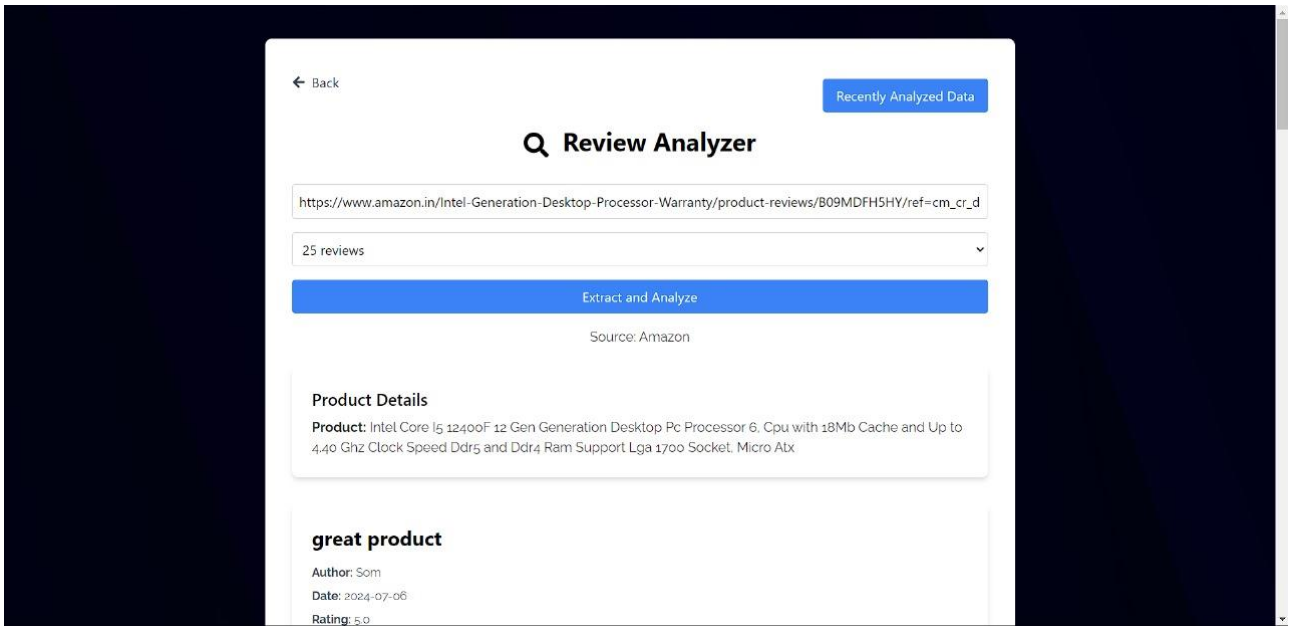

Fig 7.3: Analysed Review Data

Figure 7.4 represents the sentiments , highlights, issues, and sentiment probabilities.

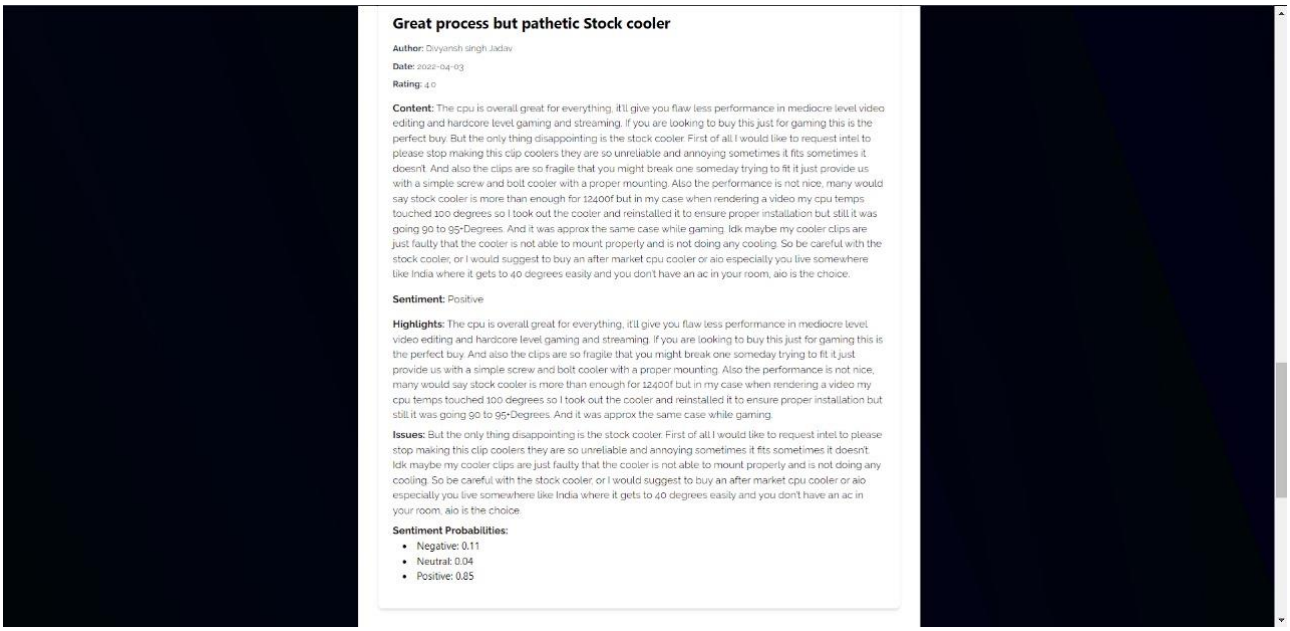

Fig 7.4: Analysed Sentiments per Review

- **EDA**: Provides exploratory data analysis (EDA) of the extracted reviews, displaying sentiment trends over time and sentiment distribution using Chart.js.

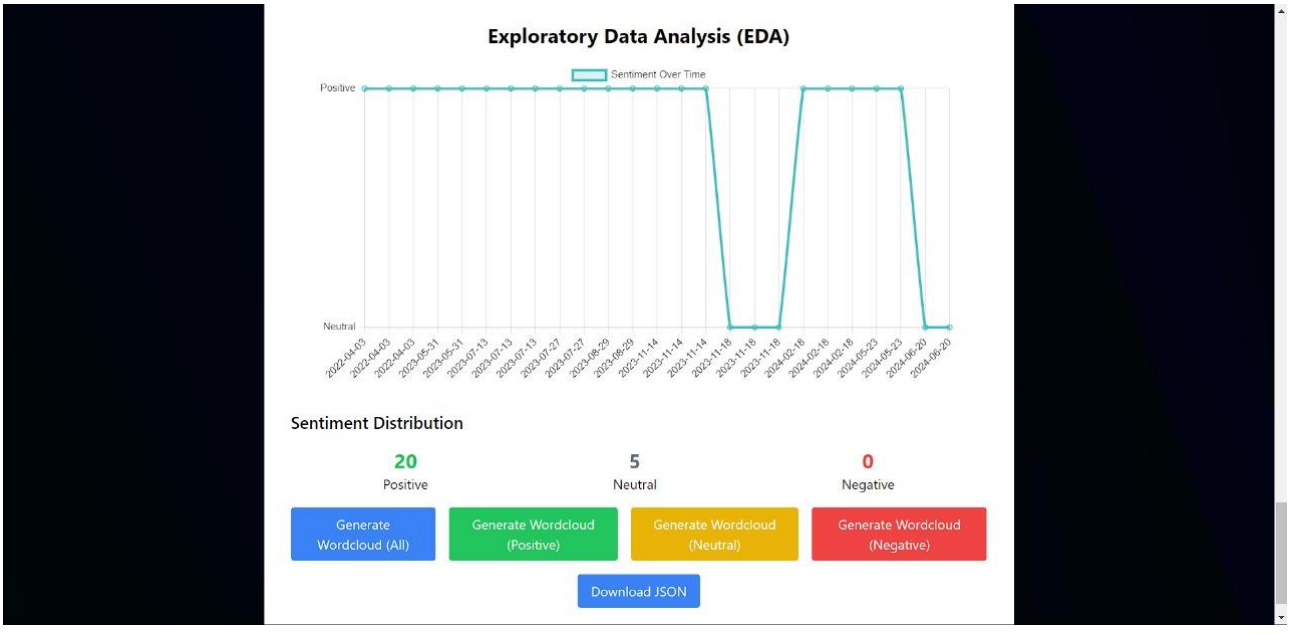

Fig 7.5: Exploratory Data Analysis

Figure 7.5 depicts the sentiment distribution of the scraped product with help of charts.

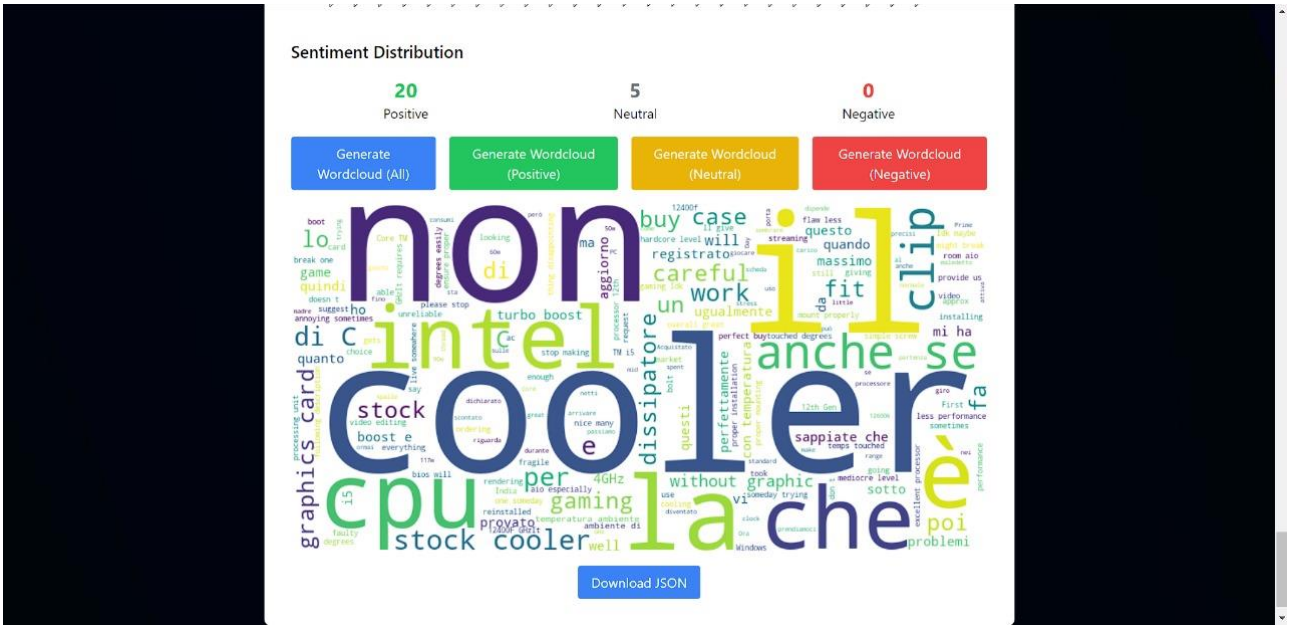

Fig 7.6: WordCloud generator

Figure 7.6 represents the generated wordcloud of positive sentiments.

- **PastTrends**: This page provides the insights based on data analysis of the Main Dataset used for training the model. Filters can be applied to obtain information about specific SKU or sorting by date or by sentiments which can be seen in Figure 7.7 and Figure 7.8.

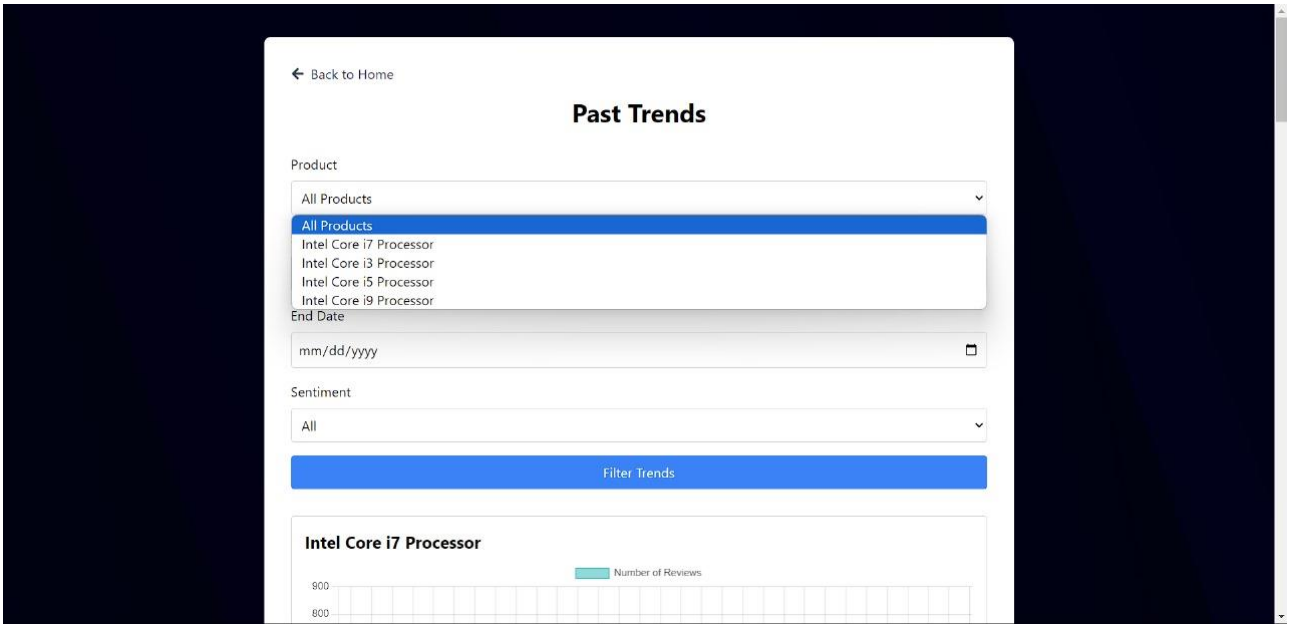

Fig 7.7: Past Trends

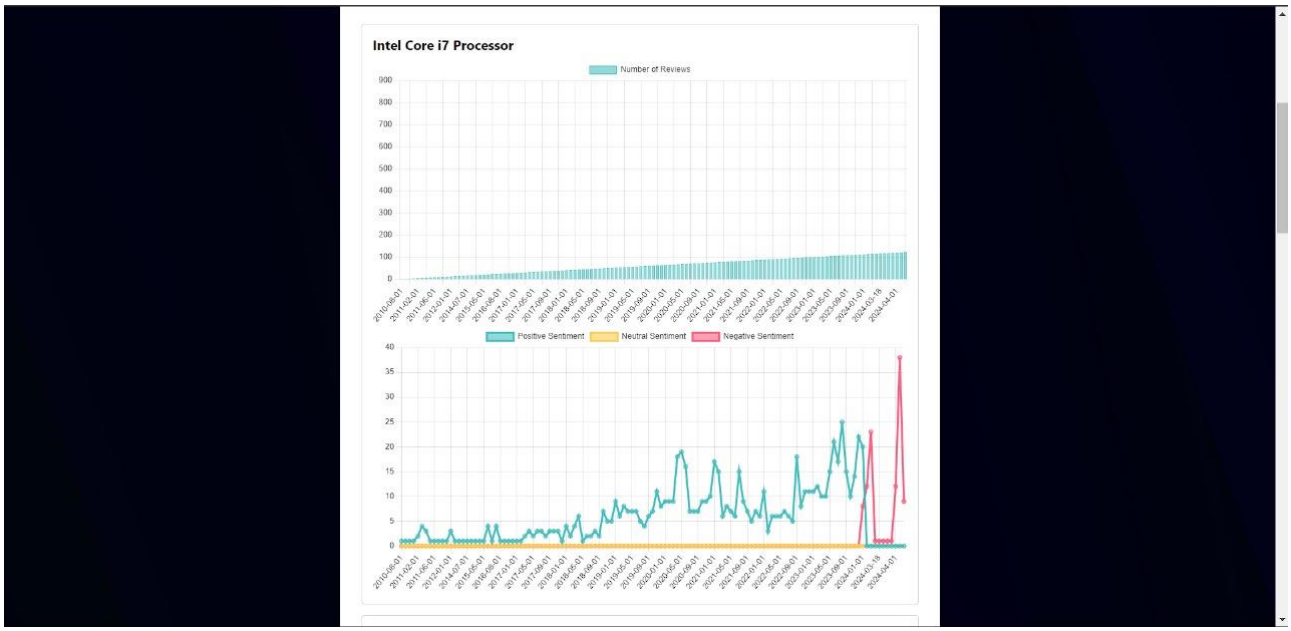

Fig 7.8: Insights of Intel Core i7 Processor

- **StoreData**: Manages the storage of extracted and analysed data, allowing users to save their analysis results for future reference. Implements functionalities for data export in various formats (e.g., CSV, JSON) and ensures data persistence across sessions.

7.1.3 Styling and User Experience

- **Tailwind CSS**: The application uses Tailwind CSS for styling, ensuring a responsive and modern design that adapts to different screen sizes.
- **Smooth Scrolling and Animations**: Implements smooth scrolling and animations to enhance the user experience, making interactions feel more natural and engaging.

7.2 Backend

7.2.1 Developed with Flask

The backend is developed using Flask, a lightweight and flexible Python web framework. Flask handles the URL scraping, sentiment analysis, and communication with the frontend.

7.2.2 Key Routes

- **/extract**: This route handles the extraction of reviews from the provided URLs. It processes the URL, scrapes the reviews, and sends the data back to the frontend.

- **/download**: This route allows users to download the analysed data in a structured format, such as CSV or JSON.

7.2.3 Cross-Origin Resource Sharing (CORS)

The backend manages cross-origin requests using Flask-CORS, allowing the frontend and backend to communicate seamlessly even when hosted on different domains.

7.3 Scraping Module

7.3.1 Review Extraction

The scraping module extracts reviews from Amazon and Flipkart using BeautifulSoup for parsing HTML content. It implements CSS selectors to accurately locate and extract review information from web pages.

7.3.2 Pagination Handling

The module handles pagination to ensure that all reviews are captured from multiple pages, providing a comprehensive dataset for analysis.

7.4 Sentiment Analysis Module

7.4.1 Fine-tuned BERT Model

The sentiment analysis module utilizes a fine-tuned BERT model which is trained in the last chapter. BERT (Bidirectional Encoder Representations from Transformers) is known for its high accuracy in natural language understanding tasks.

7.4.2 Sentiment Classification

The module classifies reviews into positive, neutral, and negative sentiments. It also calculates sentiment probabilities for each review, providing detailed insights into the sentiment distribution.

7.5 Data Visualization

7.5.1 Displaying Reviews and Analysis

The application displays the extracted reviews along with their sentiment analysis results. Users can see individual review details and overall sentiment trends.

7.5.2 Exploratory Data Analysis (EDA)

The EDA component provides visualizations such as sentiment distribution and trends over time. It uses Chart.js through the react-chartjs-2 library to create interactive and visually appealing charts.

7.6 Testing and Evaluation

7.6.1 Accuracy of Sentiment Analysis

The application was rigorously tested to ensure the accuracy of the sentiment analysis module. The BERT model's performance was validated on various review samples to ensure correct classification of sentiments.

7.6.2 Correct Extraction of Reviews

The scraping functionality was tested with multiple URLs from Amazon and Flipkart to ensure that all reviews were accurately extracted and processed.

7.6.3 Smooth Interaction between Frontend and Backend

The interaction between the React frontend and Flask backend was tested to ensure seamless communication. Cross-origin requests and response parsing were handled correctly.

7.6.4 User Experience and Interface Design

The application was tested for responsiveness and usability on different devices. Mentor feedback was implemented to enhance the interface and functionality.

CHAPTER 8

RESULTS AND DISCUSSION

8.1 Model Results

We begin by presenting the performance metrics of different models used during the project.

8.1.1 Naïve Bayes

Accuracy: 0.67

8.1.2 SVM

- Validation Accuracy: 0.9270833333333334

	precision	recall	f1-score	support
negative	1.00	0.93	0.96	41
neutral	0.85	0.97	0.91	36
positive	0.98	0.93	0.95	43
accuracy			0.94	120
macro avg	0.94	0.94	0.94	120
weighted avg	0.95	0.94	0.94	120

- Confusion Matrix

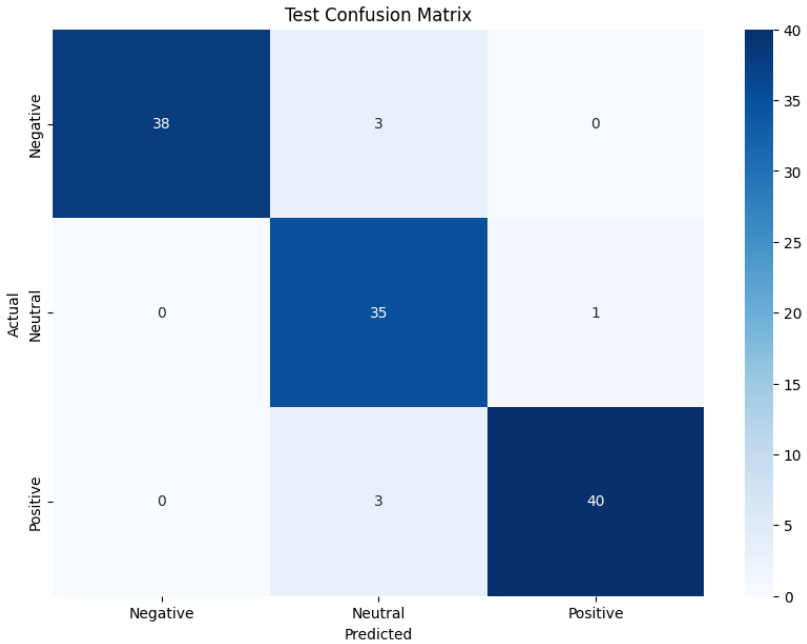

Fig 8.1 Confusion Matrix -SVM

Figure 8.1 denotes confusion matrix which is generated to visualize the model's classification performance across different classes.

8.1.3 XGBoost

- Training Accuracy: 0.9973958333333334
- Testing Accuracy: 0.8916666666666667

	precision	recall	f1-score	support
0	0.95	0.90	0.93	41
1	0.82	0.89	0.85	36
2	0.90	0.88	0.89	43
accuracy			0.89	120
macro avg	0.89	0.89	0.89	120
weighted avg	0.89	0.89	0.89	120

8.1.4 Random Forest

- Validation Accuracy: 0.93875
- Validation Classification Report:

	precision	recall	f1-score	support
negative	0.96	0.93	0.94	27
neutral	1.00	1.00	1.00	34
positive	0.94	0.97	0.96	35
accuracy			0.97	96
macro avg	0.97	0.97	0.97	96
weighted avg	0.97	0.97	0.97	96

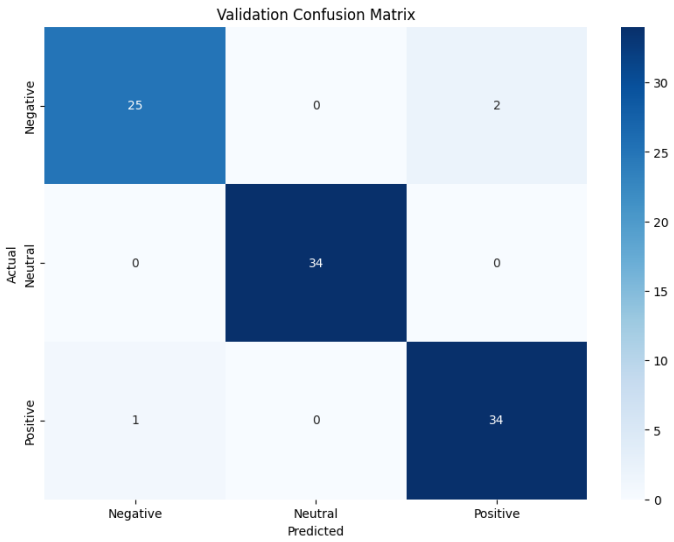

Fig 8.2 Confusion Matrix – Random Forest

- Test Accuracy: 0.8933333333333333
- Test Classification Report:

	precision	recall	f1-score	support
negative	1.00	0.88	0.94	41
neutral	0.92	0.97	0.95	36
positive	0.89	0.95	0.92	43
accuracy			0.93	120
macro avg	0.94	0.93	0.93	120
weighted avg	0.94	0.93	0.93	120

8.1.5 VADER

vader_sentiment	Negative	Neutral	Positive
rating			
1	50	24	46
2	7	2	11
3	15	8	41
4	21	22	184
5	64	194	1518

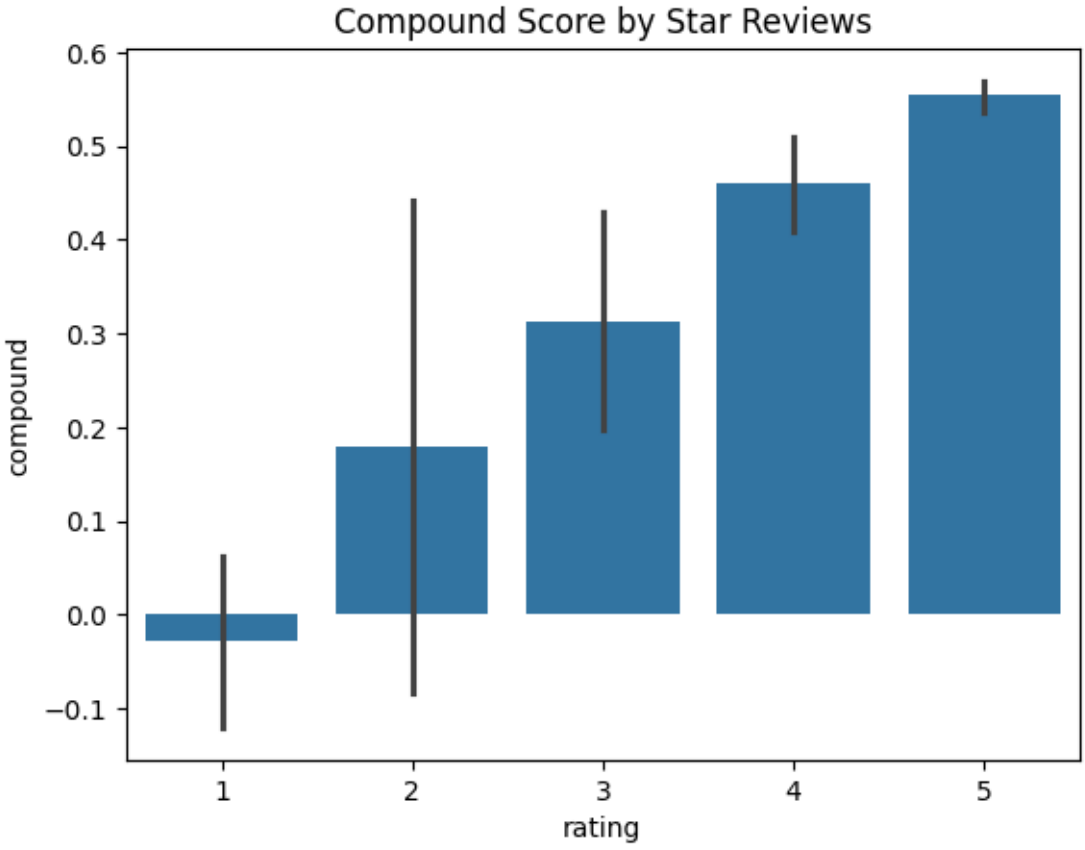

Fig 8.3 Rating vs Compound Score - VADER

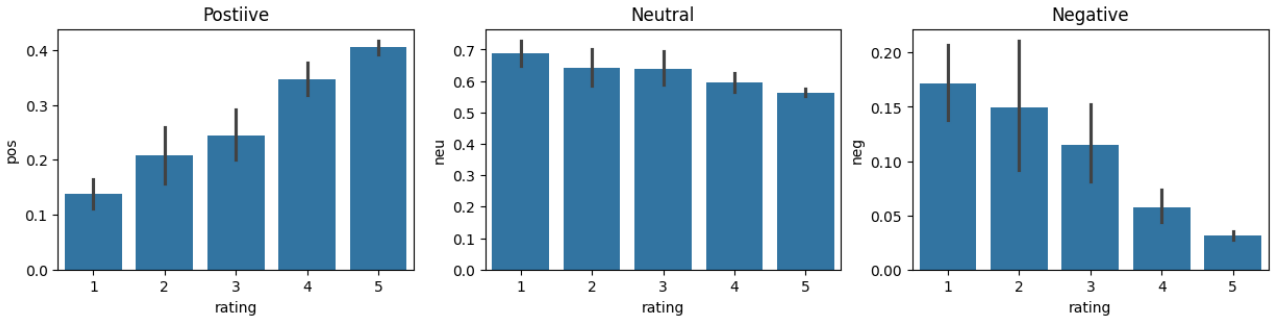

Fig 8.4 Rating vs Separate Scores – VADER

Figure 8.3 and figure 8.4 represents the graphs formed from the sentiment score by model vs ratings.

8.1.6 RoBERTa

roberta_predict rating	Negative	Neutral	Postive
1	47	65	8
2	7	8	5
3	12	30	22
4	11	77	139
5	24	427	1325

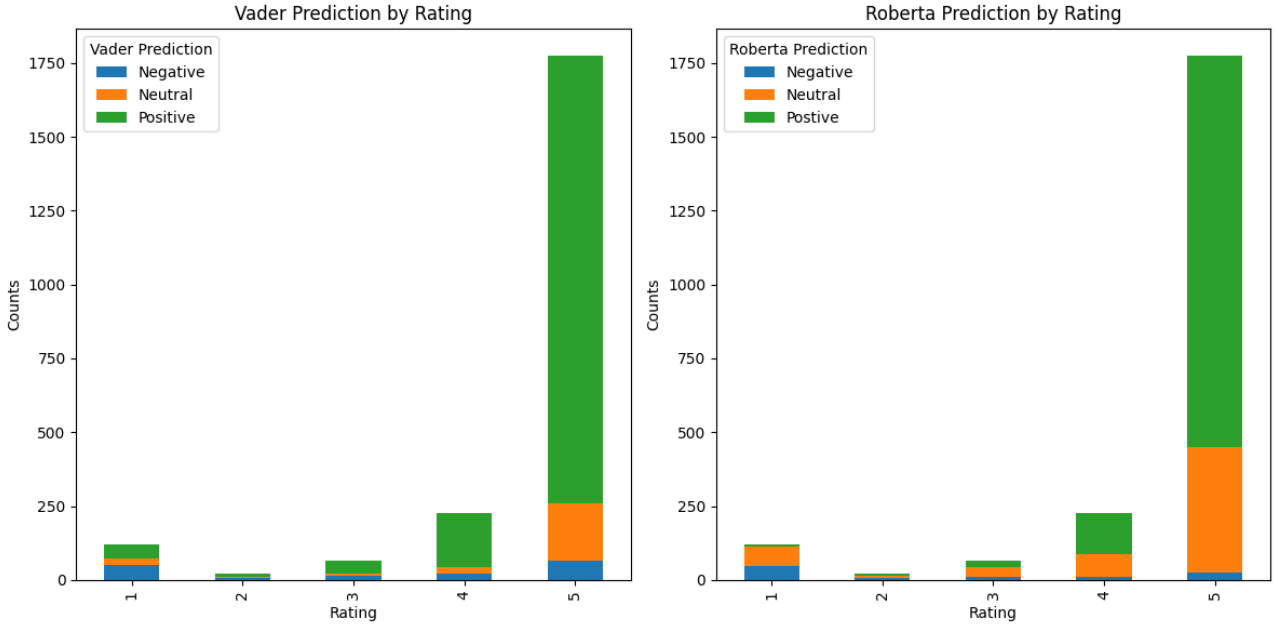

Fig 8.4 Rating vs Counts in VADER and RoBERTa

Figure 8.3 depicts the distribution of sentiment predictions made by the Vader and Roberta models across different product ratings.

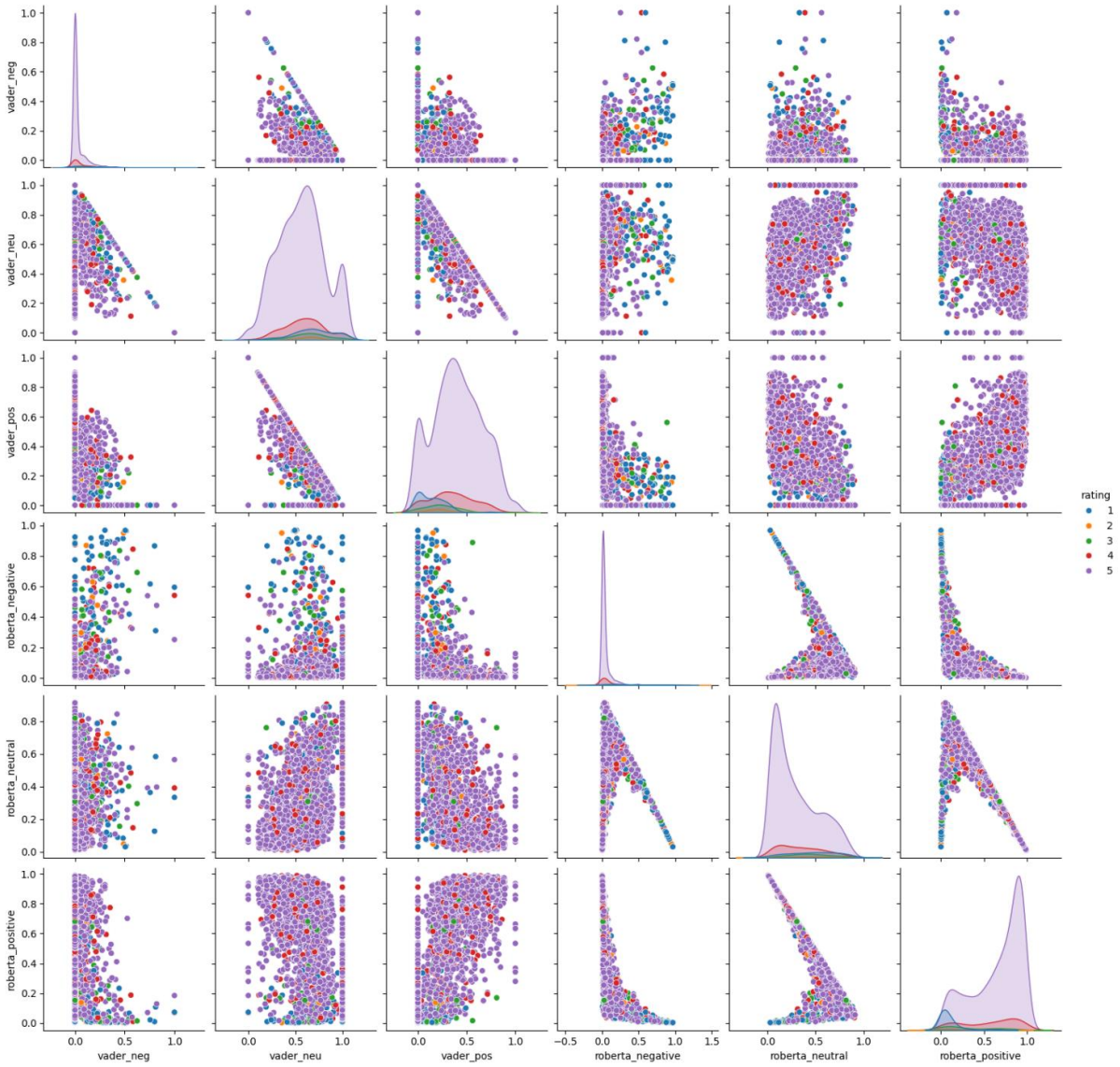

Fig 8.4 Comparisons in VADER and RoBERTa

Figure 8.4 visualizes the relationships between Vader and Roberta sentiment scores (negative, neutral, positive) and how these scores are distributed across different product ratings.

8.1.7 BERT

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.3831	0.332739	0.90724	0.863115	0.823084	0.90724
2	0.306	0.28844	0.90724	0.88583	0.868741	0.90724
3	0.1197	0.434042	0.920814	0.895794	0.888616	0.920814
4	0.3098	0.312972	0.902715	0.893704	0.88588	0.902715
5	0.1679	0.384503	0.925339	0.908875	0.895555	0.925339

Table 8.1 Comparisons in VADER and RoBERTa

Table 8.1 depicts the metrics of training loss, validation loss, Accuracy, F1, Precision and Recall Score during the training period during epochs.

- eval_loss: 0.3845028281211853
- eval_accuracy: 0.9253393665158371
- eval_f1: 0.9088747233873897
- eval_precision: 0.8955546479509034
- eval_recall: 0.9253393665158371

8.2 Dataset Insights

The insights by analysing the dataset are presented below.

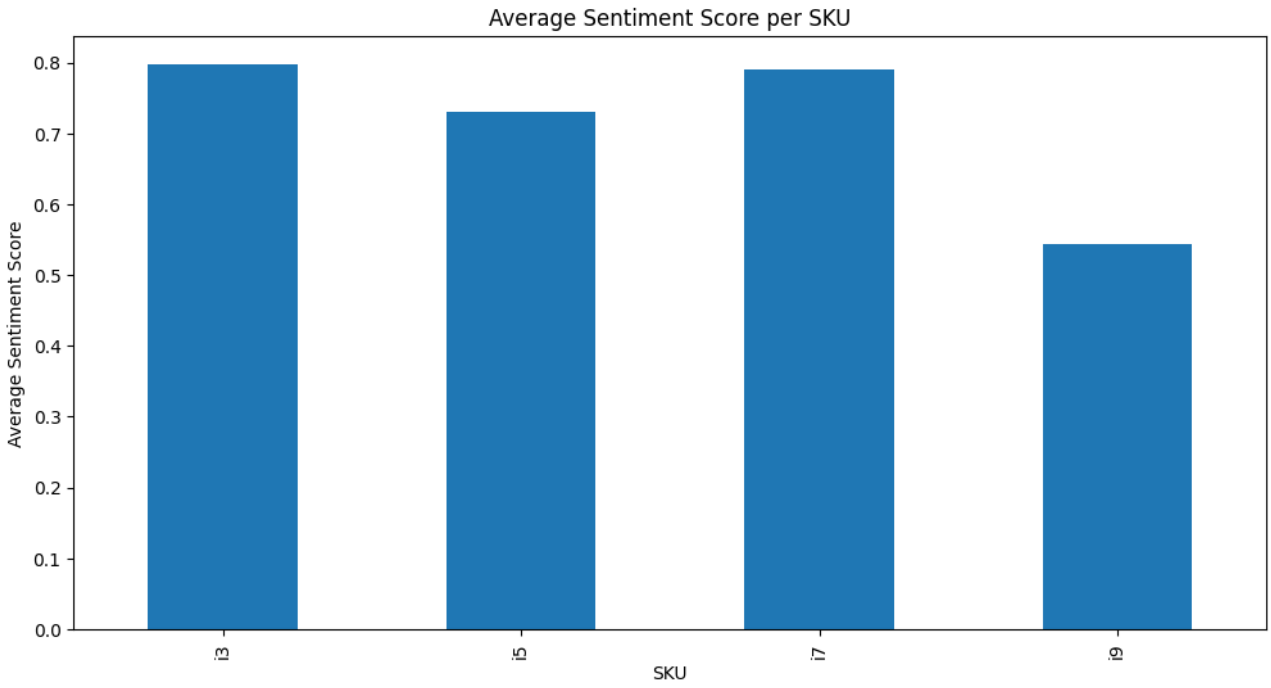

Fig 8.5 Average Sentiment Score per SKU

Figure 8.5 depicts the average sentiment scores for different Stock Keeping Units (SKUs).

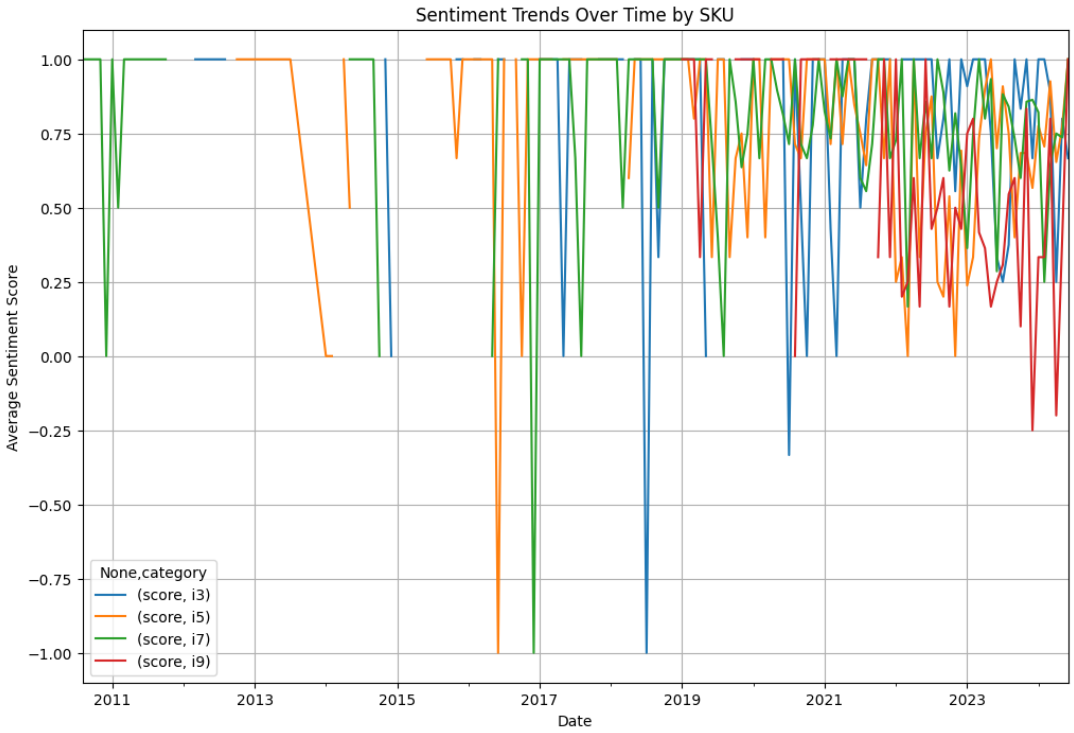

Fig 8.6 Average Sentiment Score vs Date

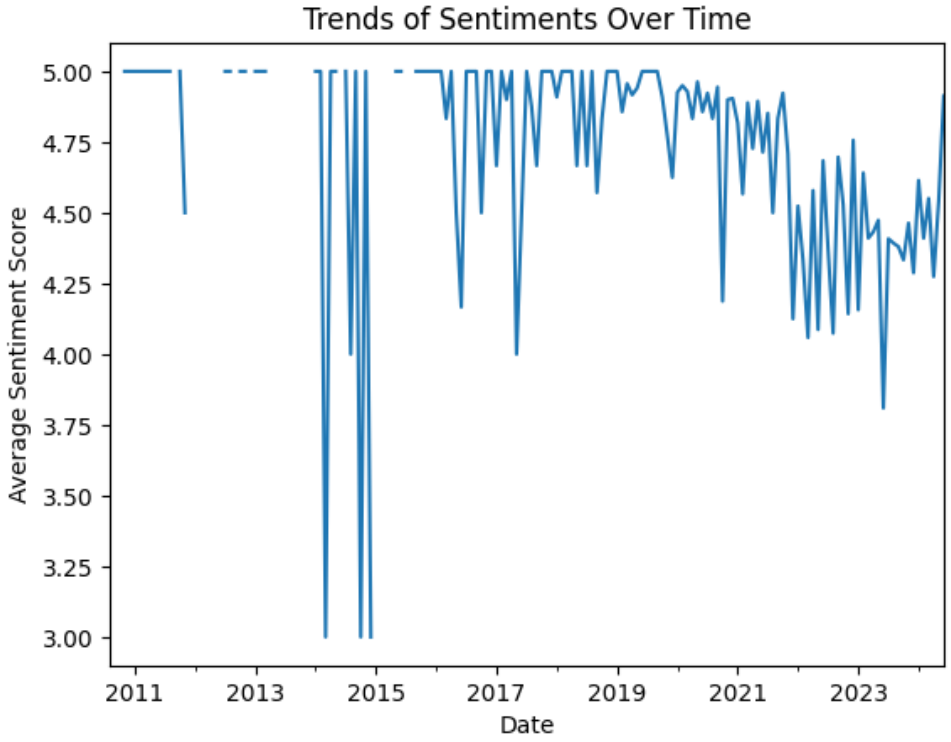

Fig 8.7 Trends of Sentiment over Time

Figure 8.6 and 8.7 depicts the trends of sentiment over time for each SKU and overall score respectively.

CHAPTER 9

CONCLUSION AND FUTURE ENHANCEMENTS

9.1 Conclusion

The project successfully addresses the challenge of analysing large volumes of customer reviews by leveraging advanced Natural Language Processing (NLP) techniques and machine learning models. By integrating the BERT model for sentiment analysis, the system provides precise classification of reviews into positive, neutral, or negative sentiments with a high accuracy of 92.533 percent . This classification is further enhanced by highlighting specific positive and negative aspects within each review, along with offering constructive improvement suggestions.

The project workflow begins with efficient data collection using Scraper API, which enables the extraction of reviews from various online platforms, complemented by the option to upload CSV files for batch processing. This data is then meticulously preprocessed to ensure accuracy and uniformity, involving steps like lowercasing, tokenization, stopword removal, and special character removal.

Post-preprocessing, the BERT model performs sentiment analysis, and the results are visualized through word clouds and trend graphs. This visualization not only provides a quick overview of common themes and topics but also illustrates how sentiments evolve over time, offering valuable insights into customer perceptions.

The system's frontend, built with React and integrated with backend APIs using Flask and Python, ensures a seamless user experience. It allows users to input data, view analyses, and download detailed reports in JSON format, facilitating further analysis and record-keeping.

Overall, our project stands out by providing a comprehensive solution that automates the process of review analysis, offers actionable insights, and supports businesses in enhancing customer satisfaction and improving their products and services.

9.2 Future Enhancements

The project can be enhanced by adding more features such as:

9.2.1 Enhanced Sentiment Analysis:

To improve the sentiment analysis capabilities, multi-language support should be integrated to analyze reviews written in different languages. This enhancement will broaden the scope of analysis and cater to a more diverse user base. Additionally, incorporating emotion detection will allow the system to classify reviews into finer-grained categories such as joy, anger, and sadness, providing deeper insights into customer sentiments.

9.2.2 Real-Time Analysis:

Real-time review fetching and analysis should be implemented for live sentiment tracking. This feature will enable businesses to monitor sentiments as they unfold. A dashboard for continuous monitoring and alerting of sentiment changes will be developed, ensuring that businesses can respond promptly to shifts in customer opinions.

9.2.3 User Feedback Integration:

Creating a feedback loop where users can validate and correct the sentiment analysis results will enhance the model's accuracy over time. This user feedback integration will allow the system to learn from corrections and improve its predictions, leading to more reliable sentiment analysis.

9.2.4 Advanced Visualization Tools:

Advanced data visualization libraries, such as D3.js, will be integrated to provide interactive and dynamic visualizations. These tools will offer more sophisticated ways to present data, making it easier to understand complex patterns. Additionally, geographic mapping will be included to visualize sentiments based on location data, providing a spatial dimension to the sentiment analysis.

9.2.5 Predictive Analysis:

Machine learning models will be used to predict future trends and sentiments based on historical data. This predictive analysis will help businesses anticipate changes and prepare accordingly. Predictive maintenance alerts will also be implemented, enabling businesses to proactively address potential issues before they escalate, thereby maintaining a high level of customer satisfaction.

REFERENCES

[1] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.

[2] Jurafsky, D., & Martin, J. H. (2021). Speech and Language Processing (3rd ed.). Draft chapters available at Stanford NLP.

[3] Pang, B., & Lee, L. (2008). Opinion Mining and Sentiment Analysis. Foundations and Trends in Information Retrieval, 2(1-2), 1-135.

[4] Laender, Alberto & Ribeiro-neto, Berthier & Silva, Altigran & Teixeira, Juliana. (2002). A Brief Survey of Web Data Extraction Tools.. SIGMOD Record. 31. 84-93. 10.1145/565117.565137.

[5] Heimerl, F., Lohmann, S., Lange, S., & Ertl, T. (2014). Word Cloud Explorer: Text Analytics Based on Word Clouds. Proceedings of the 47th Hawaii International Conference on System Sciences.

[6] McKinney, W. (2012). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media, Inc.

[7] Munzner, T. (2014). Visualization Analysis and Design. CRC Press.

[8] ScraperAPI. (2024). Documentation.

[9] React Documentation. (2024). Introduction to React.

[10] Flask Documentation. (2024). Flask: A Lightweight WSGI Web Application Framework.

[11] Kamath, U., Graham, K. L., & Emara, W. (2022). Bidirectional encoder representations from transformers (BERT). Transformers for Machine Learning, 43-70. https://doi.org/10.1201/9781003170082-3

[12] Zhou, M., Duan, N., Liu, S., & Shum, H. (2020). Progress in neural NLP: Modeling, learning, and reasoning. Engineering, 6(3), 275-290. https://doi.org/10.1016/j.eng.2019.12.014

[13] Borg, A., & Boldt, M. (2020). Using VADER sentiment and SVM for predicting customer response sentiment. Expert Systems with Applications, 162, 113746. https://doi.org/10.1016/j.eswa.2020.113746

[14] Dikilitaş, Y., Çakal, Ç., Okumuş, A. C., Yalçın, H. N., Yıldırım, E., Ulusoy, Ö. F., Macit, B., Kırkaya, A. E., Yalçın, Ö., Erdoğmuş, E., & Sayar, A. (2024). Performance analysis for web scraping tools: Case studies on Beautifulsoup, Scrapy, Htmlunit and Jsoup. Lecture Notes in Networks and Systems, 471-480. https://doi.org/10.1007/978-3-031-56728-5_39

[15] Vercel Documentation. (2024). Getting Started with Vercel.

APPENDIX

Abbreviations

API - Application Programming Interface

BERT - Bidirectional Encoder Representations from Transformers

CSS - Cascading Style Sheets

CSV - Comma-Separated Values

EDA - Exploratory Data Analysis

HTML - HyperText Markup Language

JSON - JavaScript Object Notation

NLP - Natural Language Processing

POS - Part of Speech

RNN - Recurrent Neural Network

RoBERTa - Robustly optimized BERT approach

SKU - Stock Keeping Unit

UI - User Interface

URL - Uniform Resource Locator

UX - User Experience

VADER - Valence Aware Dictionary and sEntiment Reasoner

XHR - XMLHttpRequest

CORS - Cross-Origin Resource Sharing

D3.js - Data-Driven Documents

Flask - A micro web framework written in Python

React - A JavaScript library for building user interfaces

Tailwind CSS - A utility-first CSS framework for styling

BeautifulSoup - A Python library for parsing HTML and XML documents

NLP - Natural Language Processing

EDA - Exploratory Data Analysis

ML - Machine Learning

TPU - Tensor Processing Unit

GPU - Graphics Processing Unit

TF-IDF - Term Frequency-Inverse Document Frequency