SENTIMENT DETECTION FROM TEXT DATA

Submitted by

Name	Roll No.	Registration No.
Arghya Chowdhury	**CSE2021 L02**	**211170100120005 (2021 - 22)**
Anuradha Adhikari	**CSE2021 L16**	**211170100120001 (2021 - 22)**
Pratanu Ghorui	**CSE2021 L13**	**211170100120010 (2021 - 22)**
Rittick Roy	**CSE2021 L14**	**211170100120013 (2021 - 22)**

UNDER THE SUPERVISION OF

DR. Dipankar Majumdar

(Assistant Professor (CSE), RCCIIT)

PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY
[Affiliated to MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY (MAKAUT)] CANAL SOUTH ROAD, BELIAGHATA, KOLKATA-700015

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
RCC INSTITUTE OF INFORMATION TECHNOLOGY

TO WHOM IT MAY CONCERN

I hereby recommend that the Project entitled **“Sentiment Detection from Text Data”** prepared under my supervision by Arghya Chowdhury (Roll No.: CSE2021L02), Anuradha Adhikari (Roll No.: CSE2021L16), Pratanu Ghorui (Roll No.: CSE2021L13), Rittick Roy (Roll No.: CSE2021L14) of B. Tech (7th Semester), may be accepted in partial fulfillment for the degree of **Bachelor of Technology in Computer Science & Engineering** under Maulana Abul Kalam Azad University of Technology (MAKAUT).

…………………………………………

Project Supervisor Department of
Computer Science and Engineering
RCC Institute of Information
Technology

Countersigned:

………………………

Head Department of
Computer Sc. & Engg.,
RCC Institute of
Information and
Technology, Kolkata-
700015

CERTIFICATE OF APPROVAL

The foregoing Project is hereby accepted as a credible study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve the project only for the purpose for which it is submitted.

Name of the Examiner	**Signature with Date**
1. ……………………………..…	………. …………………………….
2. ………………………………..	………………………………………
3. ………………………………..	…….………………………………...
4. ………………………………..	……..……………………………….

ACKNOWLEDGEMENT

We acknowledge our overwhelming gratitude & immense respect to our revered guide, Dr. Dipankar Majumder (Assistant Professor, RCC Institute of Information and Technology) under whose scholarly guideline, constant encouragement & untiring patience; we have proud privilege to accomplish this entire project work. We feel enriched with the knowledge & sense of responsible approach we inherited from our guide & shall remain a treasure in our life.

Date: ……………………

Arghya Chowdhury
Reg. No.: 211170100120005
Roll No.: CSE2021 L02

Anuradha Adhikari
Reg. No.: 211170100120001
Roll No.: CSE2021 L16

Pratanu Ghorui
Reg. No.: 211170100120010
Roll No.: CSE2021 L13

Rittick Roy
Reg. No.: 211170100120013
Roll No.: CSE2021 L

Table of Contents

1. ABSTRACT

This study explores sentiment analysis, a crucial facet of natural language processing, with the goal of comprehending and interpreting the emotions conveyed in textual data. The paper offers a thorough examination of a number of sentiment analysis methods, such as deep learning architectures, machine learning models, and rule-based strategies. It also looks at the various ways that sentiment analysis is used in different businesses, including market trend prediction, social media monitoring, and customer feedback analysis.

The article discusses the primary issues and concerns surrounding sentiment analysis, such as context awareness, language subtleties, and domain-specific difficulties. Additionally, it looks into how model selection, feature extraction strategies, and pre-processing approaches affect the precision and resilience of sentiment analysis systems.

The report demonstrates the efficacy of sentiment analysis in various contexts by highlighting case studies and real-world examples in the context of practical application. Additionally, it addresses moral issues surrounding sentiment analysis, including privacy issues and possible biases in algorithmic decision-making.

In conclusion, the study explores future prospects and new trends in sentiment analysis research, such as the use of multimodal data integration, cross-lingual sentiment analysis, and explainability and interpretability in sentiment analysis models. The extensive research yielded findings that facilitate the evolution of sentiment analysis approaches and foster a greater understanding of this dynamic field.

2. INTRODUCTION

Sentiment analysis has emerged as a vital tool for understanding public opinion and gauging sentiment within textual data. In the contemporary era of data-driven decision-making, sentiment analysis holds significant importance across various domains, including marketing, customer feedback analysis, and social media monitoring. The ability to accurately discern sentiment from text data can provide valuable insights for businesses and organizations. In this context, the utilization of advanced deep learning techniques, particularly BERT (Bidirectional Encoder Representations from Transformers) neural network, presents a compelling approach for sentiment analysis. BERT, renowned for its ability to capture contextual information and effectively process bidirectional sequences, has shown remarkable performance in natural language processing tasks. By leveraging the power of BERT, this project endeavors to delve into the domain of sentiment analysis and demonstrate the efficacy of BERT in accurately deciphering sentiment within textual data. The primary objective of this documentation is to outline the methodology, implementation, and outcomes of deploying the BERT neural network for sentiment analysis. By elucidating the process of training and fine-tuning the BERT model, along with the subsequent analysis of results, this documentation aims to provide a comprehensive insight into the potential and applicability of BERT in sentiment analysis tasks.

In this context, the utilization of advanced deep learning techniques, particularly BERT (Bidirectional Encoder Representations from Transformers) neural network, presents a compelling approach for sentiment analysis. BERT, renowned for its ability to capture contextual information and effectively process bidirectional sequences, has shown remarkable performance in natural language processing tasks. By leveraging the power of BERT, this project endeavors to delve into the domain of sentiment analysis and demonstrate the efficacy of BERT in accurately deciphering sentiment within textual data. The primary objective of this documentation is to outline the methodology, implementation, and outcomes of deploying the BERT neural network for sentiment analysis. By elucidating the process of training and fine-tuning the BERT model, along with the subsequent analysis of results, this documentation aims to provide a comprehensive insight into the potential and applicability of BERT in sentiment analysis tasks. Through this project, we seek to not only showcase the capabilities of BERT neural network in sentiment analysis but also to contribute to the advancement of sentiment analysis techniques, thereby fostering a deeper understanding of public sentiment within textual data.

3. REVIEW OF LITERATURE

There has been a constant growth in the public and private information stored within the internet. This includes textual data expressing people's opinions on review sites, forums, blogs, and other social media platforms. Review based prediction systems allow this unstructured information to be automatically transformed into structured data reflecting public opinion. These structured data can be used subsequently as a measure of users' sentiments about specific applications, products, services, and brands. They can hence provide important information for product and services refinement. This kind of sentiment analysis was conducted in the following studies.

Kumara and other researchers used the naïve bayes (nb) classifier to classify opinions as positive, negative, or neutral. Wang and others argued that a rating is not entirely determined by a review content. For example, a user may well intend to give a positive review by employing positive words, and yet issue a comparatively lower rating.

Dave and others proposed a method for extracting the polarity in user reviews of products, expressed as poor, mixed, or good. The classifier used was naïve bayes (nb). According to pang et al although machine learning approaches perform far better for traditional topic-based categorization, they're less successful for sentiment analysis. Information extraction technologies have also been exploring do identify and organize opinions contained.

A recent study investigated the application of a machine learning algorithm to dataset covering, for example, the app category, the numbers of reviews and downloads, the size, type, and android version of an app, and the content rating, to predict a google app ranking. Decision trees, linear regression, logistic regression, support-vector machine, nb classifiers, means clustering, k- nearest neighbors, and artificial neural networks were studied for that purpose.

Other authors suggested adopting a statistical analysis based on a spin model, to extract the semantic orientations of words. Mean field approximations were used to compute the approximate probability in the spin model. Semantic orientations are then evaluated as desirable or undesirable. A smaller number of seed words for the proposed model produce highly accurate semantic orientations based on the English lexicon. In contrast to the above-cited studies.

4. PROBLEM ANALYSIS

4.1. EXISTING SYSTEMS

Historically, sentiment analysis has relied on traditional lexicon-based methods and statistical algorithms to interpret the sentiment of textual data. Lexicon-based techniques involved the usage of sentiment lexicons and dictionaries to assign sentiment scores to words within the text. However, these approaches often encountered difficulties in handling contextual nuances and keeping pace with evolving language usage trends. Early sentiment analysis systems leveraged machine learning algorithms such as Naive Bayes, Support Vector Machines (SVM), and Logistic Regression. These algorithms relied on features extracted from text data, including n-grams and term frequency-inverse document frequency (TF-IDF), to classify sentiment. While effective to a certain extent, statistical approaches had limitations in capturing complex linguistic patterns and discerning intricate contextual dependencies within the text**.**

With the emergence of deep learning, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) gained prominence for sentiment analysis tasks. These neural network architectures were instrumental in capturing sequential data and inferring contextual dependencies within text, marking a significant leap forward in sentiment analysis capabilities.

1. **Naive Bayes:** A probabilistic approach that calculates the likelihood of words occurring in positive or negative contexts to classify text.

2. **Support Vector Machines (SVM):** Leveraging hyperplane separation in high dimensional spaces to divide text into sentiment categories.

3. **Decision Trees:** A decision-making approach that categorizes text based on a series of rules.

4. **CNN's:** Convolutional Neural Networks (CNNs) analyse local and global features in text to capture patterns relevant to sentiment.

5. **RNN's:** Recurrent Neural Networks (RNNs) excel in understanding sequential data, making them suitable for capturing contextual dependencies in sentiment analysis**.**

4.2. DRAWBACKS

1. **Limited Contextual Understanding:** Traditional lexicon-based methods and statistical algorithms often struggle to capture the nuanced contextual understanding required for accurate sentiment analysis, leading to potential misinterpretation of sentiment in complex texts.

2. **Difficulty in Handling Sarcasm and Irony:** Existing systems may struggle to discern sentiment in instances of sarcasm and irony, as these rely heavily on contextual and cultural understanding, posing challenges for accurate sentiment classification.

3. **Feature Engineering:** Traditional methods require manual feature engineering, where experts select relevant features for sentiment classification. This approach is timeconsuming, subjective, and may not capture all the nuances of language that impact sentiment.

4. **Computational Complexity:** Some approaches, particularly those involving deep learning models, may pose computational challenges due to high computational requirements for training and inference, impacting practical scalability and efficiency.

5. **Computational Complexity:** Some approaches, particularly those involving deep learning models, may pose computational challenges due to high computational requirements for training and inference, impacting practical scalability and efficiency.

4.3. PROPOSED SYSTEMS

In our project, we introduce a revolutionary shift in sentiment analysis by leveraging the power of BERT. By integrating BERT into sentiment analysis, our project aims to usher in a new era of precision and context-aware sentiment classification, enhancing decision support and insights for individuals and businesses. The landscape of sentiment analysis underwent a paradigm shift with the introduction of BERT (Bidirectional Encoder Representations from Transformers) and transformer-based architectures. BERT, renowned for its bidirectional processing and attention mechanism, has showcased remarkable performance in capturing intricate contextual information and discerning sentiment within textual data. This revolutionized the sentiment analysis landscape, opening doors to more accurate and nuanced sentiment interpretation.The utilization of BERT and transformer-based models for transfer learning has empowered sentiment analysis in tailored domains. By fine-tuning pre-trained models on domain-specific datasets, sentiment analysis systems can now capture and comprehend sentiment nuances distinct to various industries and sectors, leading to more accurate and domain-relevant sentiment insights.

In contemporary sentiment analysis, BERT and its variants, alongside transformer-based models such as Generative Pre-trained Transformer (GPT), have become pivotal in sentiment analysis tasks. Additionally, the prevalence of transfer learning techniques, where pre-trained models are fine-tuned on domain-specific data, has substantially advanced the state-of-theart in sentiment analysis. The integration of BERT and transformer-based models with transfer learning strategies has revolutionized sentiment analysis methodologies, empowered the accurate interpretation and understood of sentiment within textual data.

4.4. ADVANTAGES

1. **Enhanced Contextual Understanding:** The utilization of BERT facilitates a deeper comprehension of the contextual nuances within textual data, enabling the accurate interpretation of sentiment within diverse and complex language structures.

2. **Improved Accuracy and Consistency:** BERT's bidirectional processing empowers the model to capture intricate dependencies within the text, leading to heightened accuracy and consistency in sentiment analysis compared to traditional methods.

3. **Adaptability to Varied Textual Data:** BERT's pre-trained nature and transfer learning capabilities allow for the seamless adaptation to diverse domains and languages, thereby

   enhancing the system's versatility in analysing sentiment across a wide spectrum of textual content.

4. **Efficient Handling of Ambiguity:** Through its ability to capture bidirectional contexts, BERT excels in effectively discerning and processing ambiguous language constructs, resulting in more nuanced and precise sentiment analysis outcomes.

5. **Efficient Handling of Ambiguity:** Through its ability to capture bidirectional contexts, BERT excels in effectively discerning and processing ambiguous language constructs, resulting in more nuanced and precise sentiment analysis outcomes.

4.5. SYSTEM REQUIREMENTS

Hardware Requirements:

CPU:	Core i5 / i7 processor
RAM:	At least 8GB
HDD/SSD:	At least 256GB

Software Requirements:

Language:	Python 3.x
Libraries:	BERT, SCI-KIT learn, Pandas, Numpy, Matplotlib
IDE:	Google Colab, VS Code

5. SYSTEM DESIGN

BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an encoder that reads the text input and a decoder that produces a prediction for the task. Since BERT's goal is to generate a language model, only the encoder mechanism is necessary.

BERT is a bi-directional transformer for pre-training over a lot of unlabeled textual data to learn a language representation that can be used to fine-tune for specific machine learning tasks. While BERT outperformed the NLP state-of-the-art on several challenging tasks, its performance improvement could be attributed to the bidirectional transformer, novel pre-training tasks of Masked Language Model and Next Structure Prediction along with a lot of data and Google's compute power.

5.1. ARCHITECTURE

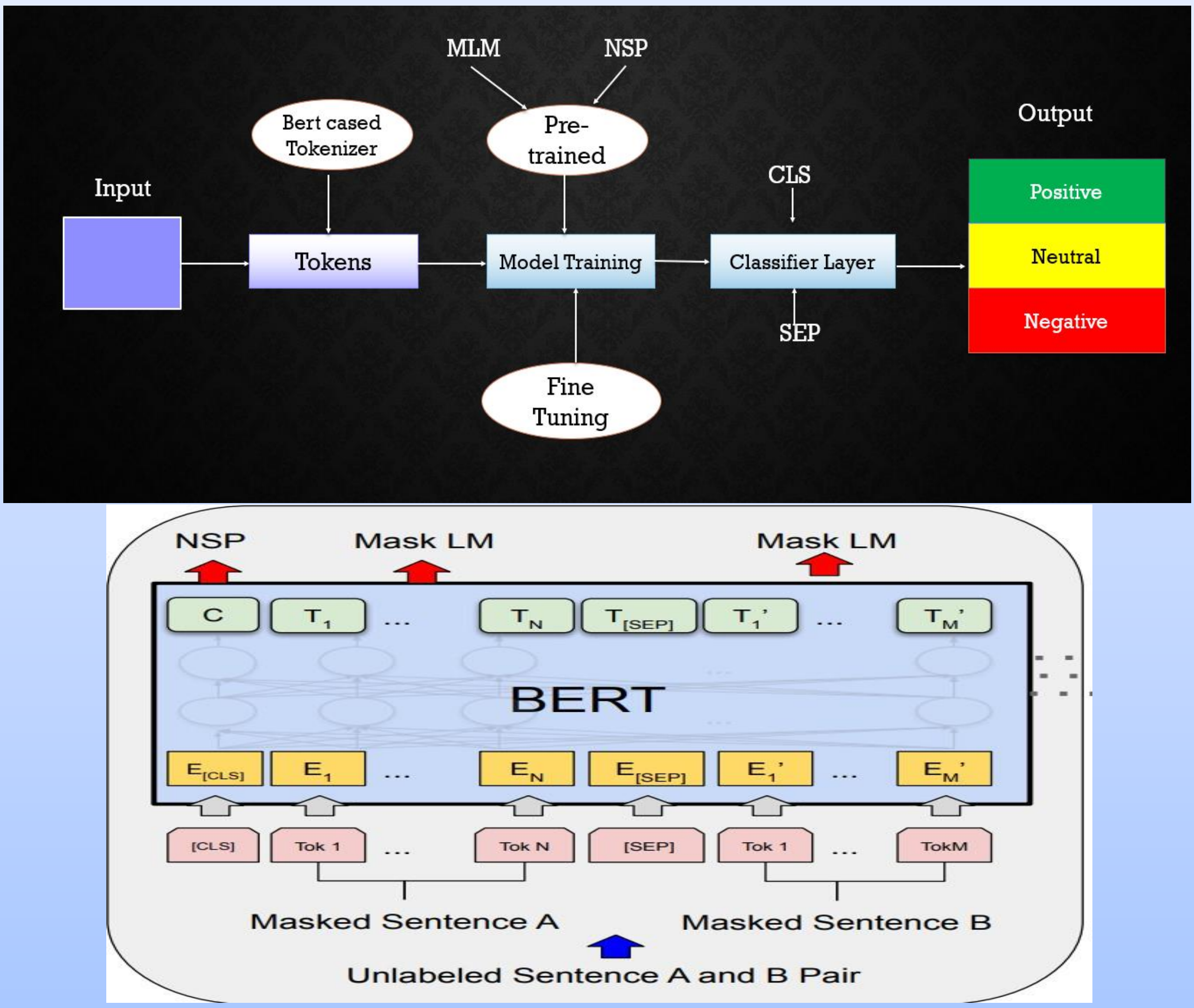


Fig. 5.1.1

1. Input: the input is the text data you want to perform sentiment analysis on. This text data is what you want BERT to analyze and make predictions about.

2. Token: In NLP, a token is a unit of text, typically a word or a subword. BERT tokenizes your input text into these smaller units, which helps the model understand the text better.

3. BERT Cased Tokenization: BERT uses a specific tokenization method, often called "WordPiece" tokenization. It tokenizes text into subword pieces, including both lowercase and uppercase versions. This helps capture case-related information.

4. Model Training: This refers to the process where the BERT model is trained on a large corpus of text data. During this phase, BERT learns to predict missing words (Masked Language Model, MLM) and understand sentence relationships (Next Sentence Prediction, NSP).

5. Pre-training (MLM, NSP): BERT is pre-trained on a vast amount of text data using two tasks - MLM, where it predicts masked words in a sentence, and NSP, where it predicts whether two sentences are contiguous in the original text. This pre-training helps BERT capture general language understanding**.**

6. Fine-tuning: this is where you adapt the pre-trained BERT model to your specific task. You fine-tune the model on a labeled dataset for sentiment analysis**.**

7. Classifier Layer: During fine-tuning, you typically add a classifier layer on top of the BERT model. This layer maps the BERT model's output to sentiment labels (positive, negative, neutral) in your case**.**

8. CLS, SEP: In BERT, [CLS] and [SEP] tokens are special tokens used to denote the beginning and separation of sentences. The [CLS] token is particularly important, as it is used for classification tasks. It encapsulates information about the entire input sequence.

9. Output: Positive, Negative, Neutral: In your sentiment analysis project, the output refers to the predictions made by the BERT model. It will assign sentiment labels to the input text, such as "positive," "negative," or "neutral," based on the training it received.

5.2. MODULES

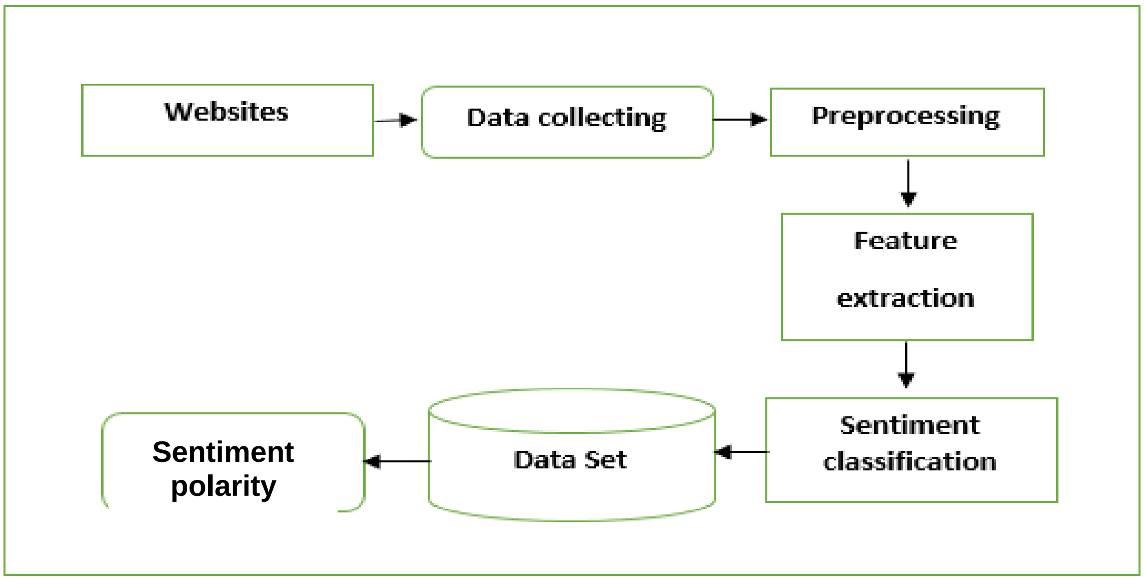


Fig. 5.2.1

A. Data Collection

This is the initial data source for your project. It refers to the text reviews or comments about a product, which serve as the input for sentiment analysis. These reviews can come from various sources like e-commerce websites, social media, or customer feedback forms.

B. Pre-processing

The pre-processing is nothing but filtering the extracted data before analysis. It includes identifying and eliminating non-textual content and content that is irrelevant to the area of study from the data.

C. Features Extraction

Feature extraction is the process of choosing relevant aspects or attributes from the product reviews that contribute to sentiment analysis. It involves identifying keywords or phrases that are indicative of sentiment, such as positive or negative words. These features will be used in sentiment classification.

D. Sentiment Classification:

This is where the sentiment labels are assigned to each review based on the extracted features. In your case, it would be classifying the reviews as "positive," "negative," or "neutral." Machine learning models, like the fine-tuned BERT model mentioned earlier, are commonly used for this task.

E. Sentiment Polarity:

Sentiment polarity refers to the degree or strength of sentiment within a review. It can be positive, negative, or neutral, but it can also have different levels of intensity. For example, a review might express a strong positive sentiment, a weak positive sentiment, a strong negative sentiment, or a weak negative sentiment. Understanding sentiment polarity provides more detailed insights into the sentiment expressed in the review.

5.3. UML DIAGRAMS

UML stands for Unified Modeling Language. It is a standardized, general-purpose modeling language in the field of software engineering that is used to visually represent a system along with its main components, relationships, and behaviors. UML provides a set of graphical notations for creating visual models of object-oriented software systems.

Key elements of UML include:

1. **Class Diagrams:** Represent the static structure of a system, showing classes, their attributes, methods, and relationships.

2. **Use Case Diagrams:** Illustrate the interactions between different actors (users or external systems) and a system, focusing on the system's functionality.

3. Sequence Diagrams: Display the interactions between objects in a specific scenario over time, showing the order in which messages are exchanged.

4. **Activity Diagrams:** Describe the flow of activities in a system, including actions, decisions, and concurrent behavior.

5. Component Diagrams: Illustrate the organization and dependencies between software components in a system.

6. Deployment Diagrams: Represent the physical deployment of software components to hardware nodes.

1. CLASS DIAGRAM:

A class diagram in UML serves as a foundational tool for object-oriented software design, providing a visual representation of a system's static structure. The diagram consists of rectangles, each representing a class, with three key compartments conveying the class name, attributes, and methods. Attributes define the properties of a class, while methods encapsulate the behaviors it can perform. Connections between classes are illustrated through lines, depicting associations or inheritances. For example, associations reveal how classes interact, while inheritance relationships signify an "is-a" connection between a superclass and its subclasses. Multiplicity notations indicate the cardinality of these relationships, detailing the number of instances involved. Overall, class diagrams offer a concise and insightful means of communicating the architecture and relationships within an object-oriented system.

In practical terms, class diagrams aid software developers in the design phase by providing a clear overview of the system's building blocks. Stakeholders can grasp the interconnections between classes, fostering better collaboration and understanding of the overall system structure. As an integral part of UML, class diagrams contribute to improved communication and documentation throughout the software development lifecycle, ensuring a shared understanding among team members and facilitating the creation of robust, well-structured software systems.

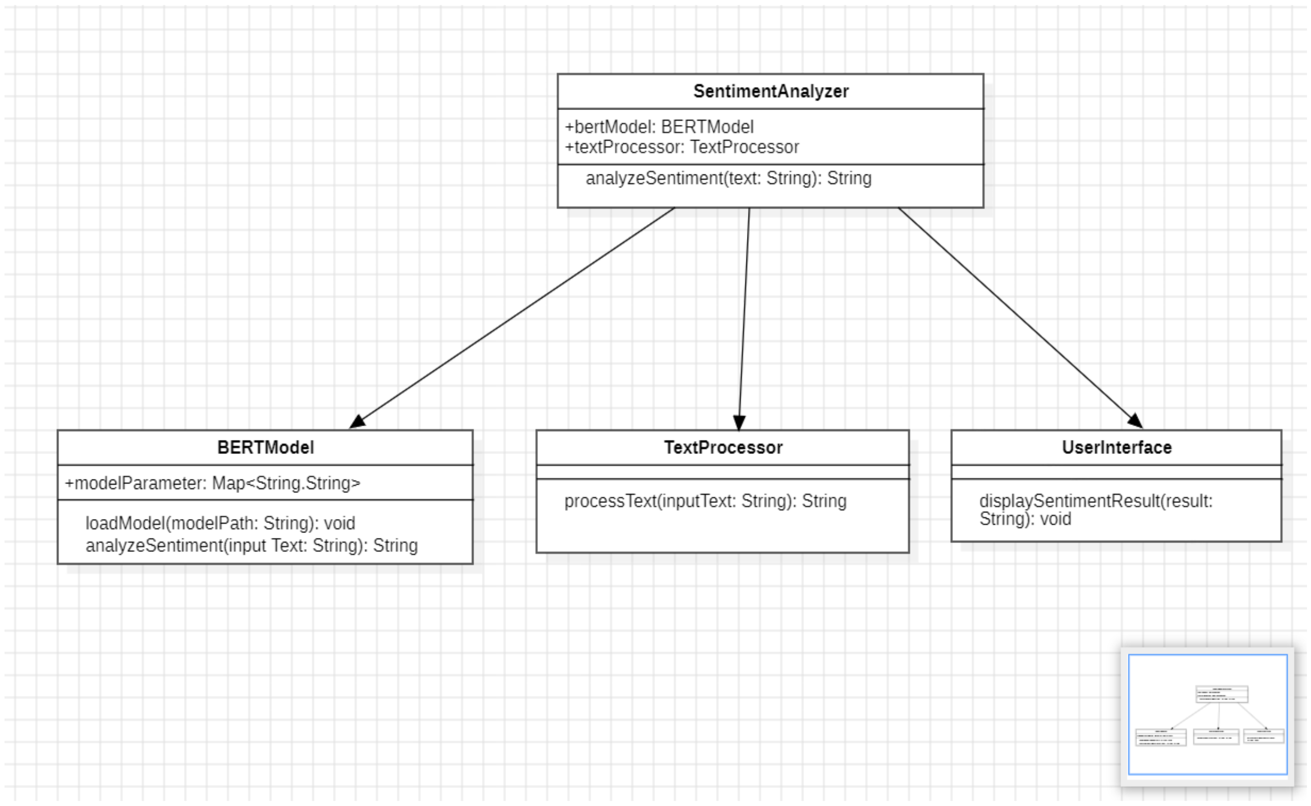

SentimentAnalyzer
+bertModel: BERTModel
+textProcessor: TextProcessor
analyzeSentiment(text: String): String
BERTModel
+modelParameter: Map<String.String>
loadModel(modelPath: String): void
analyzeSentiment(input Text: String): String
TextProcessor
processText(inputText: String): String
UserInterface
displaySentimentResult(result: String): void

Fig. 5.3.1

2. USECASE DIAGRAM:

A use case diagram is a type of Unified Modeling Language (UML) diagram that illustrates the interactions between different actors (users or external systems) and a system. It provides a high-level view of a system's functionality by focusing on the specific ways in which external entities interact with it. In a use case diagram, actors are represented as stick figures, and use cases are depicted as ovals. The lines connecting actors and use cases signify the interactions or relationships, showcasing the specific functionalities or services the system provides to its users.

The primary purpose of a use case diagram is to capture and communicate the system's functional requirements from a user's perspective. It helps stakeholders understand the various ways in which users or external systems can interact with the system and the specific features or services offered by the system in response. Use case diagrams are particularly valuable during the early stages of software development, aiding in requirements analysis, communication among team members, and providing a foundation for more detailed system design and development activities.

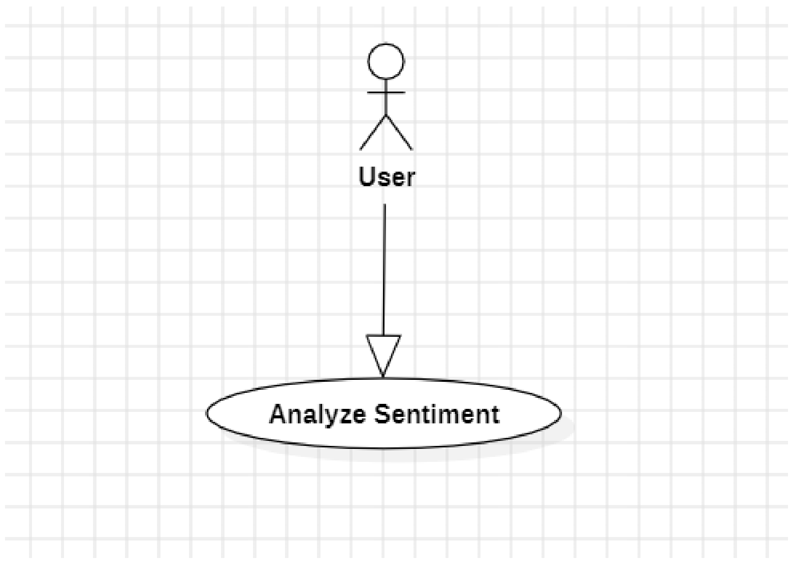


Fig. 5.3.2

3. SEQUENCE DIAGRAM:

A sequence diagram is a type of Unified Modeling Language (UML) diagram that illustrates the interactions between objects or components in a system over time. It represents a dynamic view of a system, showing the sequence of messages exchanged between different entities to accomplish a specific functionality or scenario. Sequence diagrams are particularly useful for visualizing the flow of control and the order of messages passed between objects during the execution of a use case or a specific operation.

In a sequence diagram, objects or components are represented by vertical lifelines, and the messages exchanged between them are depicted as horizontal arrows. The sequence of these arrows indicates the chronological order of interactions. Additionally, activation bars are used to represent the period during which an object is active or processing a message. Sequence diagrams help in understanding the collaboration between different parts of a system, identifying potential bottlenecks, and ensuring that the system behaves as intended during runtime. They are valuable tools in the analysis and design phases of software development, offering a visual representation of the dynamic aspects of a system's behavior.

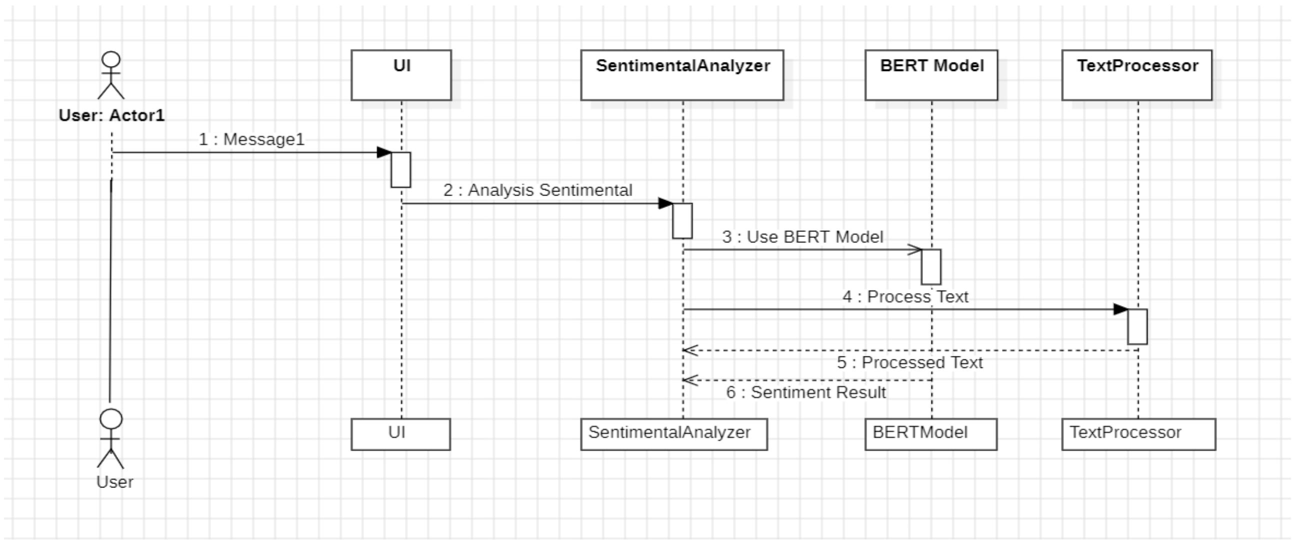


Fig. 5.3.3

4. ACTIVITY DIAGRAM:

An activity diagram is a type of Unified Modeling Language (UML) diagram that represents the dynamic aspects of a system by illustrating the flow of activities or actions. It is particularly useful for modeling the business processes, workflows, and the sequential steps within a system. Activity diagrams focus on the coordination of activities rather than the objects that perform them, making them suitable for modeling both business processes and software systems.

In an activity diagram, activities are represented as rounded rectangles, and transitions between activities are depicted by arrows. Control flow is shown through the direction of these arrows, indicating the order in which activities occur. Decision points and branching are represented using diamond shapes, allowing for the modeling of conditional behavior. Swimlanes can be used to separate activities performed by different entities or components.

Activity diagrams are valuable for visualizing and understanding complex processes, identifying potential bottlenecks, and improving the efficiency of workflows. They are commonly used during the analysis and design phases of software development, offering a clear and intuitive way to communicate the dynamic aspects of a system's behavior to stakeholders and development teams.

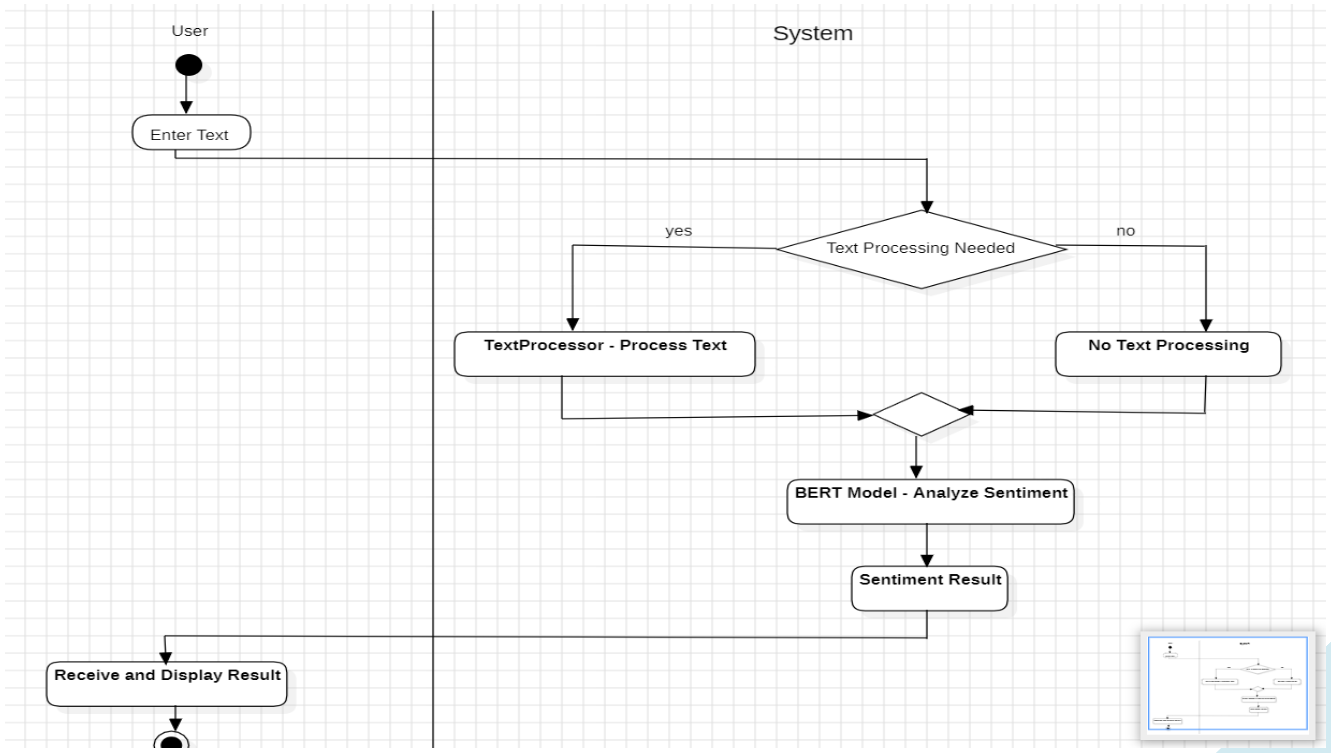

User
System
Enter Text
yes
Text Processing Needed
no
TextProcessor - Process Text
No Text Processing
BERT Model - Analyze Sentiment
Sentiment Result
Receive and Display Result

Fig. 5.3.4

5. COMPONENT DIAGRAM:

A component diagram is a type of Unified Modeling Language (UML) diagram that illustrates the organization and dependencies between software components in a system. It provides a high-level view of the physical structure of a system, focusing on the modular components and their relationships. Components in a system could be individual software modules, classes, packages, or larger subsystems.

In a component diagram, components are represented as rectangles, and the relationships between them are depicted by connecting lines. Common relationships include associations, dependencies, and interfaces. The diagram helps to visualize how different components interact with each other and how they contribute to the overall functionality of the system. Additionally, component diagrams may include deployment-related information, indicating the distribution of components across hardware nodes.

Component diagrams are valuable for system architects and developers to understand the structure of a complex system, facilitating the design and maintenance of modular and scalable software architectures. They are particularly useful in large-scale software systems where breaking down the functionality into modular components enhances maintainability, reusability, and collaboration among development teams.

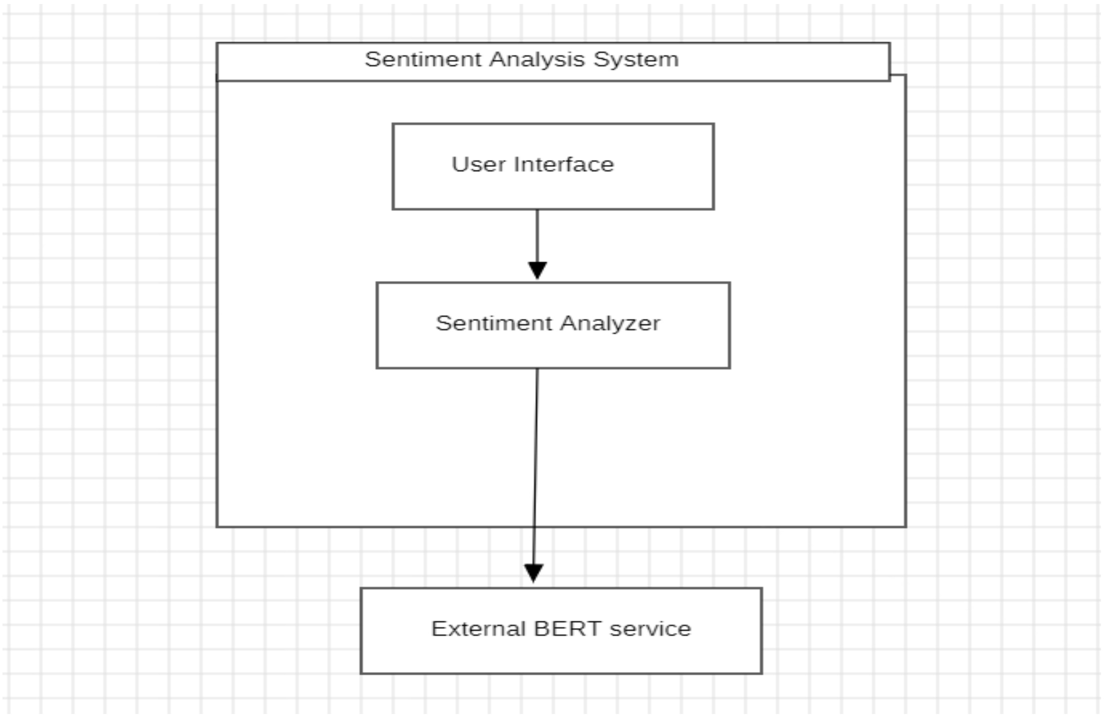


Fig. 5.3.5

6. DEPLOYMENT DIAGRAM:

A deployment diagram is a type of Unified Modeling Language (UML) diagram that models the physical deployment of software components and the relationships between them in a hardware environment. It provides a visual representation of how a software system is distributed across different nodes (hardware devices), illustrating the allocation of software components to specific hardware nodes.

In a deployment diagram, nodes (representing hardware devices) are depicted as boxes, and the software components are shown as artifacts, which are typically rectangles with the component's name inside. Deployment relationships, such as associations and dependencies, are represented by lines connecting the nodes and artifacts. Additionally, deployment diagrams may include deployment specifications, which detail configuration and deployment parameters.

Deployment diagrams are useful for understanding the physical aspects of a system's architecture, including server configurations, network topology, and the placement of software components on hardware. They are commonly employed in the later stages of system design and development

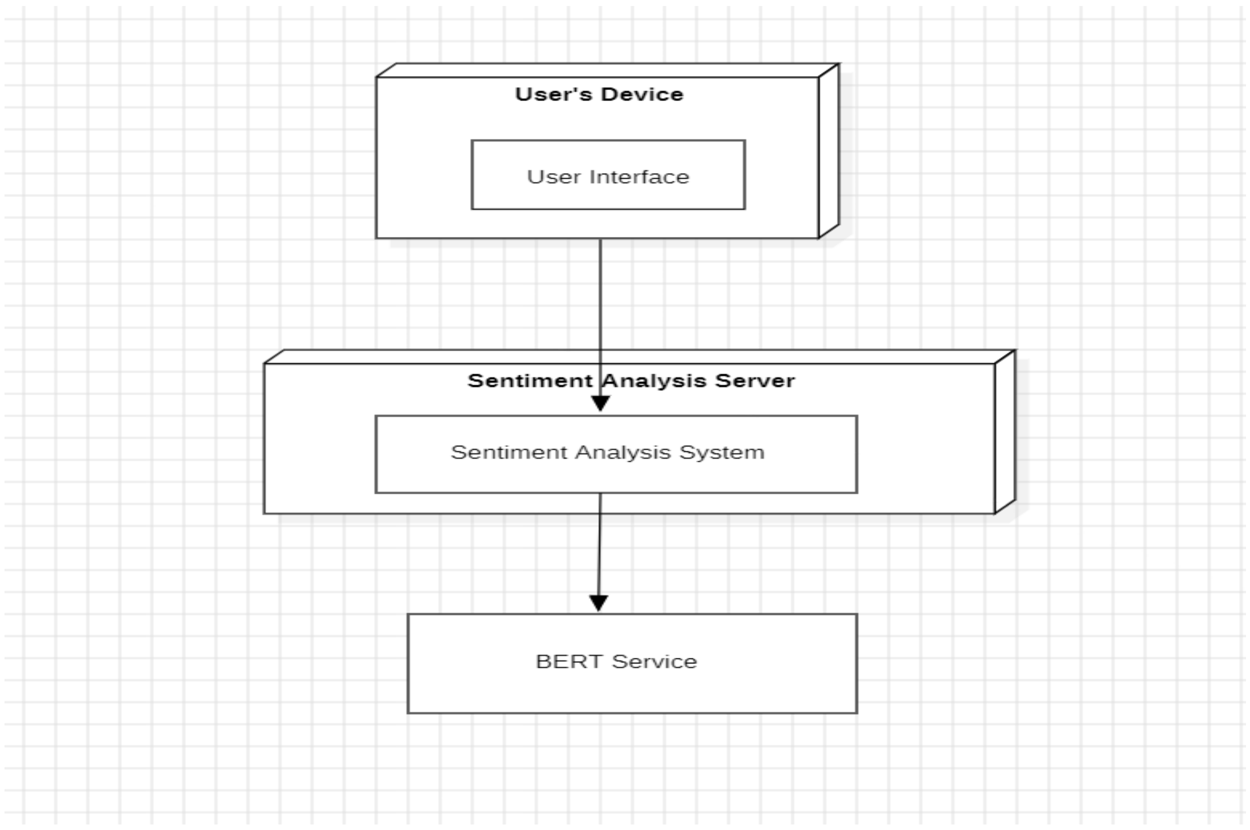


Fig. 5.3.6

6. IMPLEMENTATION & ALGORITHM

6.1. DEEP LEARNING

Deep learning is a subset of machine learning that involves neural networks with three or more layers. These neural networks attempt to simulate the behaviour of the human brain to "learn" from large amounts of data. Deep learning algorithms strive to automatically learn hierarchical representations of data, leading to better feature learning and abstraction.

These networks are designed to recognize complex patterns in data, making deep learning useful for tasks such as computer vision, natural language processing, and speech recognition. Deep learning enables machines to learn from experience and improve their performance without being explicitly programmed to do so, and it has been successful in a wide range of applications.

Deep learning has proven to be highly effective in sentiment analysis tasks. The ability of deep neural networks to automatically learn intricate features from raw data makes them well-suited for processing and understanding the complexity of natural language. In sentiment analysis, deep learning models can capture nuanced relationships and patterns in text data, allowing for more accurate sentiment predictions.

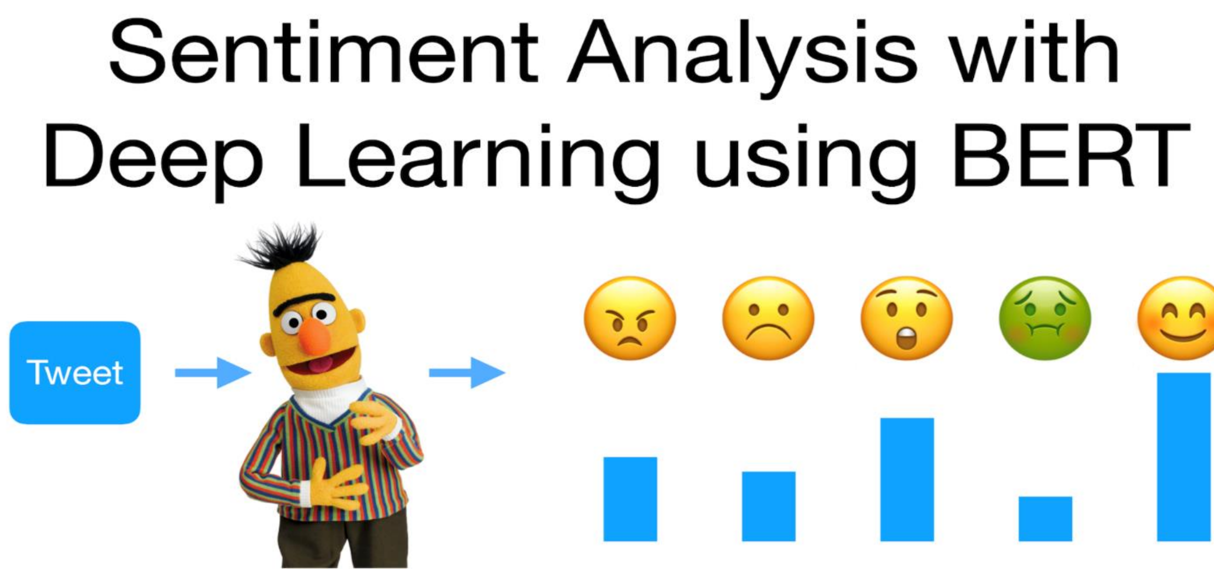


Fig 6.1.1

The life of deep learning programs is straightforward and can be summarized in the following points

- Problem Definition
- Data Collection
- Data Preprocessing
- Data visualization
- Model Selection and Architecture Design

- Model Training
- Model Evaluation
- Model Fine-Tuning
- Testing and Feedback Collection
- Model Deployment
- Monitoring and Maintenance

Types of Deep Learning Algorithms:

1. Feedforward Neural Networks (FNN):

Description: The basic building block of deep learning, consisting of an input layer, hidden layers, and an output layer. Information travels in one direction, from input to output.

Usefulness for Sentiment Analysis: FNNs can be used for basic sentiment classification tasks, especially when dealing with simple structures in text data.

2. Recurrent Neural Networks (RNN):

Description: Designed to work with sequential data, RNNs have connections that form directed cycles. This architecture allows them to capture dependencies in sequences.

Usefulness for Sentiment Analysis: RNNs are suitable for sentiment analysis tasks involving sequences of text, where the order of words matters. However, they may struggle with long-term dependencies.

3. Long Short-Term Memory (LSTM) Networks:

Description: A type of RNN that addresses the vanishing gradient problem, enabling the model to capture long-term dependencies in sequential data.

Usefulness for Sentiment Analysis: LSTMs are effective for sentiment analysis tasks where understanding the context of words over long distances is crucial.

4. Gated Recurrent Unit (GRU) Networks:

Description: Similar to LSTMs, GRUs are designed to handle sequential data and address the vanishing gradient problem. They have a simpler architecture than LSTMs.

Usefulness for Sentiment Analysis: GRUs are suitable for sentiment analysis tasks, offering a balance between performance and computational efficiency.

5. Convolutional Neural Networks (CNN):

Description: Originally designed for image processing, CNNs use convolutional layers to detect hierarchical features. They can also be applied to text data by treating it as an image.

Usefulness for Sentiment Analysis: CNNs are effective for extracting local patterns and features from text data, making them suitable for sentiment analysis tasks.

6. Transformer Networks:

Description: Introduced with the Attention Is All You Need paper, transformers use self-attention mechanisms to capture contextual relationships in input sequences.

Usefulness for Sentiment Analysis: Transformer architectures, like BERT, have excelled in various

NLP tasks, including sentiment analysis. They can capture long-range dependencies and context effectively.

6.2 NATURAL LANGUAGE PROCESSING (NLP)

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) that focuses on the interaction between computers and humans through natural language. The primary goal of NLP is to enable machines to understand, interpret, and generate human-like language.

NLP techniques used in sentiment analysis encompass various processes, including tokenization (breaking down text into smaller units), part-of-speech tagging (identifying grammatical parts of speech in a sentence), named entity recognition (identifying named entities within text), sentiment lexicons (using pre-assigned sentiment scores for words or phrases), and machine learning models (such as neural networks) to predict sentiment based on learned patterns and features in text data.

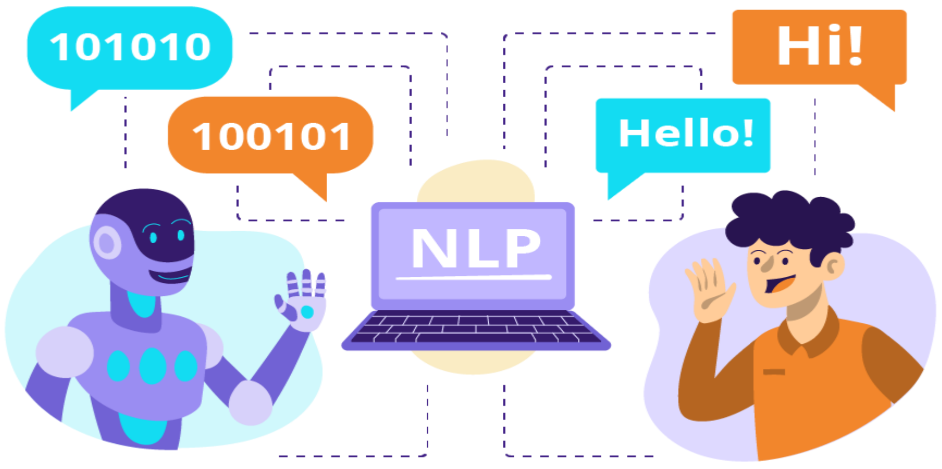


Fig 6.2.1

NLP Techniques for Sentiment Analysis:

1.Bag-of-Words (BoW):

Description: Representing a document as an unordered set of words, disregarding grammar and word order.

Usefulness for Sentiment Analysis: BoW is a simple and effective technique for representing text data, especially for traditional machine learning models.

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

Description: Assigning weights to words based on their frequency in a document and across multiple documents.

Usefulness for Sentiment Analysis: TF-IDF helps in highlighting words that are important in a specific document but not common across all documents, providing a more contextually relevant representation.

3.Word2Vec:

Description: Word embedding technique that represents words as vectors based on their co-occurrence patterns.

Usefulness for Sentiment Analysis: Word2Vec captures semantic relationships between words, enhancing the understanding of word meanings in sentiment analysis tasks.

4.GloVe (Global Vectors for Word Representation):

Description: Word embedding technique based on global word-word co-occurrence statistics.

Usefulness for Sentiment Analysis: GloVe provides vector representations for words, considering their global relationships, which can improve the model's contextual understanding.

6.3 PYTHON

Python programming language is used for building the machine learning model.

6.3.1 Introduction

Python is an object-oriented, high level language, interpreted, dynamic and multipurpose programming language. Python is easy to learn yet powerful and versatile scripting language which makes it attractive for Application Development. Python's syntax and dynamic typing with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas. Python supports multiple programming pattern, including object oriented programming, imperative and functional programming or procedural styles. Python is not intended to work on special area such as web programming. That is why it is known as multipurpose because it can be used with web, enterprise, 3D CAD etc.

We don't need to use data types to declare variable because it is dynamically typed so we can write a=10 to declare an integer value in a variable. Python makes the development and debugging fast because there is no compilation step included in python development and edit-test-debug cycle is very fast.

6.3.2 Python Features

- **Easy to Use**

  Python is easy to very easy to use and high-level language. Thus it is a programmer-friendly language.

- **Interpreted Language**

  Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

- **Cross-platform language**

  Python can run equally on different platforms such as Windows, Linux, Unix, Macintosh etc. Thus, Python is a portable language.

- **Free and Open Source**
  Python language is freely available.

- **Object-Oriented language**
  Python supports object-oriented language. The concept of classes and objects comes into existence.

- **Extensible**
  It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in your Python code.

- **Large Standard Library**
  Python has a large and broad library.

- **GUI Programming**
  Graphical user interfaces can be developed using Python.

- **Integrated**
  It can be easily integrated with languages like C, C++, JAVA etc.

6.3.3. Python Applications

Python as a whole can be used in any sphere of development. Let us see what are the major regions where Python proves to be handy.

1) Console Based Application Python can be used to develop console based applications.

2) Audio or Video based Applications Python proves handy in multimedia section.

3) 3D CAD Applications Fandango is a real application which provides full features of CAD.

4) Web Applications Python can also be used to develop web based application. Some important developments are PythonWikiEngines, Pocoo, PythonBlogSoftware etc.

5) Enterprise Applications Python can be used to create applications which can be used within an Enterprise or an Organization.

6) Applications for Images Using Python several application can be developed for image. Applications developed are VPython, Gogh, imgSeek etc. There are several such applications which can be developed using Python

6.3.4. Python Execution

1) Interactive Mode: You can enter "python" in the command prompt and start working with Python by executing Python commands.

2) Script Mode: Using Script Mode , Python code is written in a separate file using any editor of the Operating System. It is then saved using .py extension. In order to open use the command " python

file_name.py” after setting the path of the file in command prompt. NOTE: Path in the command prompt should be where you have saved your file. In the above case file should be saved at desktop.

3) Using IDE: (Integrated Development Environment) Python code can be executed using a Graphical User Interface (GUI).It is done by following the below steps:

Click on Start button -> All Programs -> Python -> IDLE(Python GUI) In IDE both interactive and script mode can be used.

• Using Interactive mode: Execute your Python code on the Python prompt and it will display result simultaneously.

• Using Script Mode:

i) Click on Start button -> All Programs -> Python -> IDLE(Python GUI)

ii) Python Shell will be opened. Now click on File -> New Window. A new Editor will be opened .

Write your Python code here.

Click on file -> save as Run then code by clicking on Run in the Menu bar.

Run -> Run Module

Result will be displayed on a new Python shell

6.4. FUNDAMENTALS OF PYTHON

This section contains the basic fundamentals of Python.

6.4.1. Tokens

Tokens can be defined as a punctuator mark, reserved words and each individual word in a statement. Token is the smallest unit inside the given program. Tokens include Keywords, Identifiers, Literals, Operators.

6.4.2. Tuples

Tuple is another form of collection where different type of data can be stored. It is similar to list where data is separated by commas. Only the difference is that list uses square bracket and tuple uses parenthesis. Tuples are enclosed in parenthesis and cannot be changed. Eg: tuple=('rahul',100,60.4,'deepak')

6.4.3. Dictionary

Dictionary is a collection which works on a key-value pair. It works like an associated array where no two keys can be same. Dictionaries are enclosed by curly braces ({}) and values can be retrieved by square bracket([]) Eg: dictionary={'name':'charlie','id':100,'dept':'it'}

6.5. GOOGLE COLAB

Google Colab, short for Google Colaboratory, is a cloud-based platform provided by Google that allows users to write, execute, and share Python code in a collaborative environment. It provides an integrated development environment (IDE) within a web browser, offering a cost-effective and accessible way for individuals to work on machine learning and data analysis projects without requiring powerful local hardware.

Google Colab's ease of use, free access to accelerators, collaboration features, and integration with widely used tools make it a popular choice among data scientists, machine learning engineers, researchers, and students for developing and sharing code and projects in a cloud-based environment.

Some key features of Google Colab include:

1. Free Access to GPU and TPU: Google Colab provides free access to graphics processing units (GPUs) and tensor processing units (TPUs), which can significantly accelerate computation for machine learning tasks like training deep learning models.

2. Easy Collaboration: Google Colab allows users to share their work with others, making it an excellent tool for collaborative coding and data science projects.

3. Seamless Integration with Google: It seamlessly integrates with Google Drive, enabling users to save and access their notebooks and datasets directly from their Google Drive storage.

4. Jupyter Notebook Support: Users can create and execute Jupyter notebooks within Google Colab, leveraging its versatile interface and support for rich text, images, and code execution.

5. Libraries and Dependencies: Google Colab supports popular Python libraries such as TensorFlow, Keras, PyTorch, OpenCV, and others, making it a suitable platform for machine learning and deep learning tasks.

6. Version Control: Google Colab integrates with Git, allowing users to manage and track changes in their notebooks using version control systems.

7. Educational and Research Use: It is commonly used in education and research settings due to its ease of access and the provision of powerful hardware resources for computationally intensive tasks.

6.6. VISUAL STUDIO CODE

Visual Studio Code (VS Code) is a free and lightweight source code editor developed by Microsoft. It is widely used by developers for various programming languages and platforms. Visual Studio Code (VS Code) can be used in several ways:

1. Coding and Development: VS Code provides a powerful code editor with features like syntax highlighting, code completion, and IntelliSense, which facilitate writing ML algorithms for project, You can write and debug your ML code directly in VS Code, making it convenient for implementing and testing different ML models.

2. Python and ML Libraries: Python is a popular programming language for ML tasks, including traffic prediction. VS Code has excellent support for Python development, including a built-in Python interpreter and integration with popular ML libraries such as scikit-learn, TensorFlow, and PyTorch. You can leverage these libraries to build and train ML models for traffic prediction within the VS Code environment.

3. Jupyter Notebooks: VS Code supports Jupyter Notebooks, which are interactive documents that allow you to combine code, visualizations, and explanatory text. Jupyter Notebooks are commonly used in ML tasks for data exploration, model development, and result analysis. You can create and work with Jupyter Notebooks in VS Code, making it easier to iterate on and document your traffic prediction experiments.

4. Data Visualization: VS Code has extensions and integrations with data visualization libraries such as Matplotlib and Plotly, enabling you to create insightful visualizations of your traffic data. Visualizing the data can help you understand patterns, trends, and anomalies, which are crucial for developing accurate traffic prediction models.

5. Git Integration and Collaboration: Traffic prediction projects often involve collaboration and version control. VS Code's built-in Git integration allows you to manage your code repository directly within the editor. You can easily commit, push, and pull changes, collaborate with team members, and track project history, ensuring smooth collaboration and code management.

6. Terminal and Command-Line Tools: Traffic prediction projects may require running command-line tools or scripts for data preprocessing, model training, or evaluation. VS Code provides an integrated terminal, allowing you to execute command-line operations without leaving the editor. You can run scripts, manage dependencies, and interact with the commandline tools required for your traffic prediction ML workflow.

7. Extension Ecosystem: VS Code has a vast extension ecosystem, including ML-specific extensions, that can enhance your traffic prediction workflow. These extensions provide additional functionality, such as data exploration tools, model evaluation metrics, automated hyperparameter tuning, and deployment options. You can explore and install relevant ML extensions from the VS Code Marketplace to augment your traffic prediction ML capabilities.

In summary, VS Code offers a flexible and feature-rich environment for developing and implementing related resources to the project. It provides coding support, integration with ML libraries, Jupyter Notebook capabilities, data visualization tools, collaboration features, command-line access, and a wide range of extensions to enhance your traffic prediction ML workflow.

6.7 ALGORITHM

The BERT (Bidirectional Encoder Representations from Transformers) model is being used as part of the algorithm for sentiment analysis in the project. BERT is a powerful natural language processing model developed by Google, capable of understanding context and meaning in language. It has been widely adopted for various NLP tasks, including sentiment analysis, due to its ability to capture complex language patterns.

This algorithm includes steps for importing packages, loading datasets, and preparing the data for model training.

Algorithm for Sentiment Analysis using BERT:

Step 1: Importing Packages

- Import required packages such as pandas, numpy, matplotlib, seaborn, transformers, torch, and sklearn.

Step 2: Set Key Variables

- Define key variables including random seed, batch size, and the device (GPU or CPU) for running the model.

Step 3: Load Dataset

- Read the dataset for sentiment analysis, for example, a CSV file containing movie reviews or user feedback.

Step 4: Data Preprocessing

- Prepare the data for training, validation, and testing sets.
- Tokenize the text data using the BERT tokenizer and apply padding to ensure consistent input sequence length.

Step 5: Create Data Loaders

- Define functions to create data loaders for the training, validation, and testing sets with tokenized and padded input.

Step 6: Define BERT Model

- Instantiate the BERT model and tokenizer using the transformers library.

Step 7: Model Training

- Define the training loop, including loss function, optimizer, and the number of training epochs.

Step 8: Model Evaluation

- Evaluate the trained model on the validation and testing data using appropriate metrics such as accuracy, precision, recall, and F1 score.

Step 9: Deployment

- Deploy the trained sentiment analysis model for inference on new text data.

This provides an outline of the algorithm flow for integrating the BERT model into sentiment analysis.

6.8. PACKAGES

Here's a list of the Python modules used along with brief descriptions:

6.8.1. pandas (pd): This module is used for data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series.

6.8.2. numpy (np): NumPy is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

6.8.3. matplotlib.pyplot (plt): Matplotlib is a plotting library for creating static, animated, and interactive visualizations in Python. The module pyplot provides a MATLAB-like interface for creating plots and visualizations.

6.8.4. seaborn (sns): Seaborn is a data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

6.8.5. sklearn.model_selection: This module provides various utilities for model selection, evaluation, and tuning. It includes tools for dividing datasets into training and testing sets, cross-validation, and parameter tuning.

6.8.6. sklearn.metrics: The metrics module in scikit-learn contains various supervised learning evaluation metrics, including confusion matrix, classification report, accuracy score, precision, recall, and F1 score.

6.8.7. transformers: This module provides state-of-the-art general-purpose architectures for natural language understanding and natural language generation. It includes pre-trained models, tokenizers, and various utilities for working with Transformers-based models.

6.8.8. torch: Torch is a scientific computing framework with wide support for machine learning algorithms. It provides tensor computing with strong GPU acceleration and is used for building and training neural networks.

These modules collectively enable functions such as data handling, visualization, model training and evaluation, and natural language processing, making the code versatile and powerful for sentiment analysis.

6.9. SOURCE CODE

```
! pip install transformers

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='whitegrid',font_scale=1.2)
sns.set_palette(sns.color_palette("rocket"))
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import transformers
from transformers import BertModel, BertTokenizer, AdamW,
get_linear_schedule_with_warmup
import torch
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
from collections import defaultdict
# ignore the warnings
import warningswarnings.filterwarnings('ignore')
# Let's start by defining some key variables that will be used later on in the
training/evaluation process

RANDOM_SEED = 50
BATCH_SIZE = 16 # Note that increasing the batch size reduces the training time
significantly, but gives you lower accuracy.# Set seed for reproducibility.
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

df_reviews = pd.read_csv("bert_sentiment.csv")
df_reviews.head(3)

# drop review id
df_reviews.drop('reviewId',axis=1,inplace=True)
# lets check the shape of reviews dataframe
df_reviews.shape
(12495, 11)
df_reviews.isnull().sum()
def draw_percentage(ax,total=float(len(df_reviews))):
for p in ax.patches:
percentage = '{:.1f}%'.format(100 * p.get_height()/total)
x = p.get_x() + p.get_width() / 2.
```

```
y = p.get_height()
ax.annotate(percentage, (x, y),ha='center',va='bottom')
plt.figure(figsize = (10,6))
total = float(len(df_reviews))
ax = sns.countplot(x = 'score',data=df_reviews)
plt.title('Count Plot of Review Score', fontsize=20)
plt.xlabel('review score')draw_percentage(ax)
plt.show()

def to_sentiment(rating):
rating = int(rating)
if rating <= 2:
return 0
elif rating == 3:
return 1
else:
return 2
df_reviews['sentiment'] = df_reviews.score.apply(to_sentiment)
plt.figure(figsize = (8,6))
total = float(len(df_reviews))
ax = sns.countplot(x = 'sentiment',data=df_reviews)
class_names = ['negative', 'neutral', 'positive']
ax.set_xticklabels(class_names)
plt.title('Count Plot of Review Score', fontsize=20)
plt.xlabel('review sentiment')
draw_percentage(ax)
plt.show()
ax.set_xticklabels(class_names)plt.title('Count Plot of Review Score', fontsize=20)
plt.xlabel('review sentiment')
draw_percentage(ax)
plt.show()
# We can use a cased and uncased version of BERT and tokenizer. I am using cased
version.
PRE_TRAINED_MODEL_NAME = 'bert-base-cased'
# Let's load a pre-trained BertTokenizer
tokenizer = BertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)
{"model_id":"1e656deb3cef4536b155f058873e7827","version_major":2,"version_mino
r":0}
{"model_id":"8b446bae6c9f4b1dae280da4a29369c7","version_major":2,"version_mino
r":0}
{"model_id":"5a4eeccdc41c4d969cdad7813a8ce5f0","version_major":2,"version_minor
":0}
sample_txt = "we love you"
tokens = tokenizer.tokenize(sample_txt)
```

```
token_ids = tokenizer.convert_tokens_to_ids(tokens)
print(f' Sentence: {sample_txt}')
print(f' Tokens: {tokens}')
print(f'Token IDs: {token_ids}')

encoding = tokenizer.encode_plus(
sample_txt, padding='max_length', # Pad sentence to max length
truncation=True, #Truncate sentence to max length
max_length=32,
add_special_tokens=True, # Add '[CLS]' and '[SEP]'
return_token_type_ids=False,
return_attention_mask=True, # Return attention mask
return_tensors='pt', # Return torch objects
)
encoding.keys()
dict_keys(['input_ids', 'attention_mask'])
print(len(encoding['input_ids'][0]))
print(encoding['input_ids'][0])


# The attention mask has the same length
print(len(encoding['attention_mask'][0]))
print(encoding['attention_mask'])
# We can inverse the tokenization to have a look at the special tokens
tokenizer.convert_ids_to_tokens(encoding['input_ids'][0])
# BERT works with fixed-length sequences. We'll use a simple strategy to choose the
max length.
# Let's store the token length of each review.
token_lens = []
for text in df_reviews.content:
tokens = tokenizer.encode(text, truncation=True,max_length=512)
token_lens.append(len(tokens))
plt.figure(figsize = (8,6))
sns.histplot(token_lens,kde=True)
plt.xlim([0, 150])
plt.xlabel('Token count')
plt.show()
MAX_LEN = 150
# We have all building blocks required to create a torch dataset. Let's do it...
class dataset(Dataset):
def __init__(
self, reviews, targets, tokenizer, max_len):
self.reviews = reviews
self.targets = targets
self.tokenizer = tokenizer
self.max_len = max_len
def __len__(self):
return len(self.reviews)


def __getitem__(self, item): # step 1: get the reviews and targets
review = str(self.reviews[item])
target = self.targets[item]
# step 2: use tokenizer to encode sentence (includes padding/truncation up to max
length)
encoding = self.tokenizer.encode_plus(
review,
add_special_tokens=True, # Add '[CLS]' and '[SEP]'
padding='max_length', # Pad sentence to max length
truncation=True, # Truncate sentence to max length
max_length=self.max_len,
return_token_type_ids=False,
return_attention_mask=True, # Return attention mask
return_tensors='pt', # return torch objects/tensor
)
return {
'review_text': review,
'input_ids': encoding['input_ids'].flatten(), # Tensor of token ids to be fed to a model
'attention_mask': encoding['attention_mask'].flatten(), #Tensor of indices specifying
which tokens should be attended to by the model
 'targets': torch.tensor(target, dtype=torch.long)
}
df_train, df_test = train_test_split(df_reviews, test_size=0.1,
random_state=RANDOM_SEED)
df_val, df_test = train_test_split(df_test, test_size=0.5,
random_state=RANDOM_SEED)
print('Train Data Size', df_train.shape)
print('Validation Data Size', df_val.shape)
print('Test Data Size', df_test.shape)
def create_data_loader(df, tokenizer, max_len, batch_size):
ds = dataset(
 reviews=df.content.to_numpy(),
targets=df.sentiment.to_numpy(),
tokenizer=tokenizer,
max_len=max_len
)
return DataLoader(
ds,
batch_size=batch_size
)
train_data_loader = create_data_loader(df_train, tokenizer, MAX_LEN, BATCH_SIZE)
val_data_loader = create_data_loader(df_val, tokenizer, MAX_LEN, BATCH_SIZE)
test_data_loader = create_data_loader(df_test, tokenizer, MAX_LEN, BATCH_SIZE)


data = next
data.keys()
bert_model =
BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME,return_dict=False)
last_hidden_state, pooled_output = bert_model(
  input_ids=encoding['input_ids'],
  attention_mask=encoding['attention_mask']
)
# The last_hidden_state is a sequence of hidden states of the last layer of the model.
# Obtaining the pooled_output is done by applying the BertPooler on last_hidden_state.
last_hidden_state.shape
class SentimentClassifier(nn.Module):
    def __init__(self, n_classes):
        super(SentimentClassifier, self).__init__()
        self.bert =
BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME,return_dict=False)
        # dropout layer for some regularization
        self.drop = nn.Dropout(p=0.3)
        # A fully-connected layer for our output
        self.out = nn.Linear(self.bert.config.hidden_size, n_classes)
    def forward(self, input_ids, attention_mask):
        """
        Feed input to BERT and the classifier to compute logits.
        @param    input_ids (torch.Tensor): an input tensor with shape (batch_size,
                      max_length)
        @param    attention_mask (torch.Tensor): a tensor that hold attention mask
                      information with shape (batch_size, max_length)
        @return   logits (torch.Tensor): an output tensor with shape (batch_size,
                      num_labels)
        """
        # Feed input to BERT
        last_hidden_state,pooled_output = self.bert(
        input_ids=input_ids,
        attention_mask=attention_mask
        )
        output = self.drop(pooled_output)
```

```
 return self.out(output)
# Let's create an instance and move it to the GPU.
model = SentimentClassifier(len(class_names))
model = model.to(device)
# We'll move the example batch of our training data to the GPU
input_ids = data['input_ids'].to(device)
attention_mask = data['attention_mask'].to(device)
print(input_ids.shape) # batch size x seq length

print(attention_mask.shape) # batch size x seq length
EPOCHS = 3
#  AdamW optimizer to correct weight decay
optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)
total_steps = len(train_data_loader) * EPOCHS
#  We'll also use a linear scheduler with no warmup steps
scheduler = get_linear_schedule_with_warmup(
  optimizer,
  num_warmup_steps=0,
  num_training_steps=total_steps
)
# cross-entropy loss function
loss_fn = nn.CrossEntropyLoss().to(device)
def train_epoch(model,data_loader,loss_fn,optimizer,device,scheduler,n_examples):
    # put model in training mode
    model = model.train()
    # Create empty lists to store outputs
    losses = []
    correct_predictions = 0
    for batch in data_loader:
        # We'll move the example batch of our training data to the GPU
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        targets = batch["targets"].to(device)

        # Perform a forward pass. This will return logits.
        outputs = model(
              input_ids=input_ids,
              attention_mask=attention_mask
            )
        # Get the predictions
        _, preds = torch.max(outputs, dim=1)

        # Compute loss and accumulate the loss values
        loss = loss_fn(outputs, targets)
```

```
losses.append(loss.item())
        # Calculate the accuracy rate
        correct_predictions += torch.sum(preds == targets)
        # backward pass - Perform a backward pass to calculate gradients
        loss.backward()
        # gradient clipping - Clip the norm of the gradients to 1.0 to prevent "exploding
gradients"
        nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        # Update parameters and the learning rate

optimizer.step()
        scheduler.step()
        # Zero out any previously calculated gradients
        optimizer.zero_grad()

    return correct_predictions.double() / n_examples, np.mean(losses)
def eval_model(model, data_loader, loss_fn, device, n_examples):
    # put model in evaluation mode
    model = model.eval()
    # Create empty lists to store outputs
    losses = []
    correct_predictions = 0
    with torch.no_grad():
        for batch in data_loader:
            # We'll move the example batch of our validation data to the GPU
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            targets = batch["targets"].to(device)

            # Perform a forward pass. This will return logits.
            outputs = model(input_ids=input_ids,attention_mask=attention_mask)
            # Get the predictions
            _, preds = torch.max(outputs, dim=1)

            # Compute loss and accumulate the loss values
            loss = loss_fn(outputs, targets)
            losses.append(loss.item())
            # Calculate the accuracy rate
            correct_predictions += torch.sum(preds == targets)

    return correct_predictions.double() / n_examples, np.mean(losses)
%%time
history = defaultdict(list)
```

```
best_accuracy = 0
# Start training loop
for epoch in range(EPOCHS):
    print(f'Epoch {epoch + 1}/{EPOCHS}')
    print('-' * 100)
    # model training
    train_acc, train_loss = train_epoch(model, train_data_loader, loss_fn, optimizer,
device, scheduler, len(df_train))
    print(f'Train loss {train_loss} accuracy {train_acc}')
    # After the completion of each training epoch, measure the model's performance on
our validation set.

val_acc, val_loss = eval_model(model, val_data_loader, loss_fn, device, len(df_val))
    print(f'Val loss   {val_loss} accuracy {val_acc}')
    print()
    # append training accuracy,loss and validation accuracy and loss to the history
variable
    history['train_acc'].append(train_acc)
    history['train_loss'].append(train_loss)
    history['val_acc'].append(val_acc)
    history['val_loss'].append(val_loss)
    # save the best model based on below condition
    if val_acc > best_accuracy:
        torch.save(model.state_dict(), 'best_model_state.bin')
        best_accuracy = val_acc
history_cpu_train_acc = [i.cpu() for i in history['train_acc']]
history_cpu_val_acc = [i.cpu() for i in history['val_acc']]
plt.plot(history_cpu_train_acc, label='train accuracy')
plt.plot(history_cpu_val_acc, label='validation accuracy')
plt.title('Training history')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.ylim([0, 1])
plt.show()
model = SentimentClassifier(len(class_names))
model.load_state_dict(torch.load('best_model_state.bin'))
model = model.to(device)
test_acc, _ = eval_model(
  model,
  test_data_loader,
  loss_fn,
  device,
len(df_test)
)
test_acc.item()
def get_predictions(model, data_loader):
  # put model in evaluation mode
  model = model.eval()
  # Create empty lists to store outputs
  review_texts = []
  predictions = []
  prediction_probs = []
  real_values = []

  with torch.no_grad():

for batch in data_loader:
      # We'll move the example batch of our test data to the GPU
      texts = batch["review_text"]
      input_ids = batch["input_ids"].to(device)
      attention_mask = batch["attention_mask"].to(device)
      targets = batch["targets"].to(device)
      # Perform a forward pass. This will return logits.
      outputs = model(
        input_ids=input_ids,
        attention_mask=attention_mask
      )
      # Get the predictions
      _, preds = torch.max(outputs, dim=1)
      review_texts.extend(texts)
      predictions.extend(preds)
      prediction_probs.extend(outputs)
      real_values.extend(targets)
  predictions = torch.stack(predictions).cpu()
  prediction_probs = torch.stack(prediction_probs).cpu()
  real_values = torch.stack(real_values).cpu()
  return review_texts, predictions, prediction_probs, real_values
y_review_texts, y_pred, y_pred_probs, y_test = get_predictions(
  model,
  test_data_loader
)
print(classification_report(y_test, y_pred, target_names=class_names))
def show_confusion_matrix(confusion_matrix):
  hmap = sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="plasma")
  hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(), rotation=0, ha='right')
  hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(), rotation=30, ha='right')
```

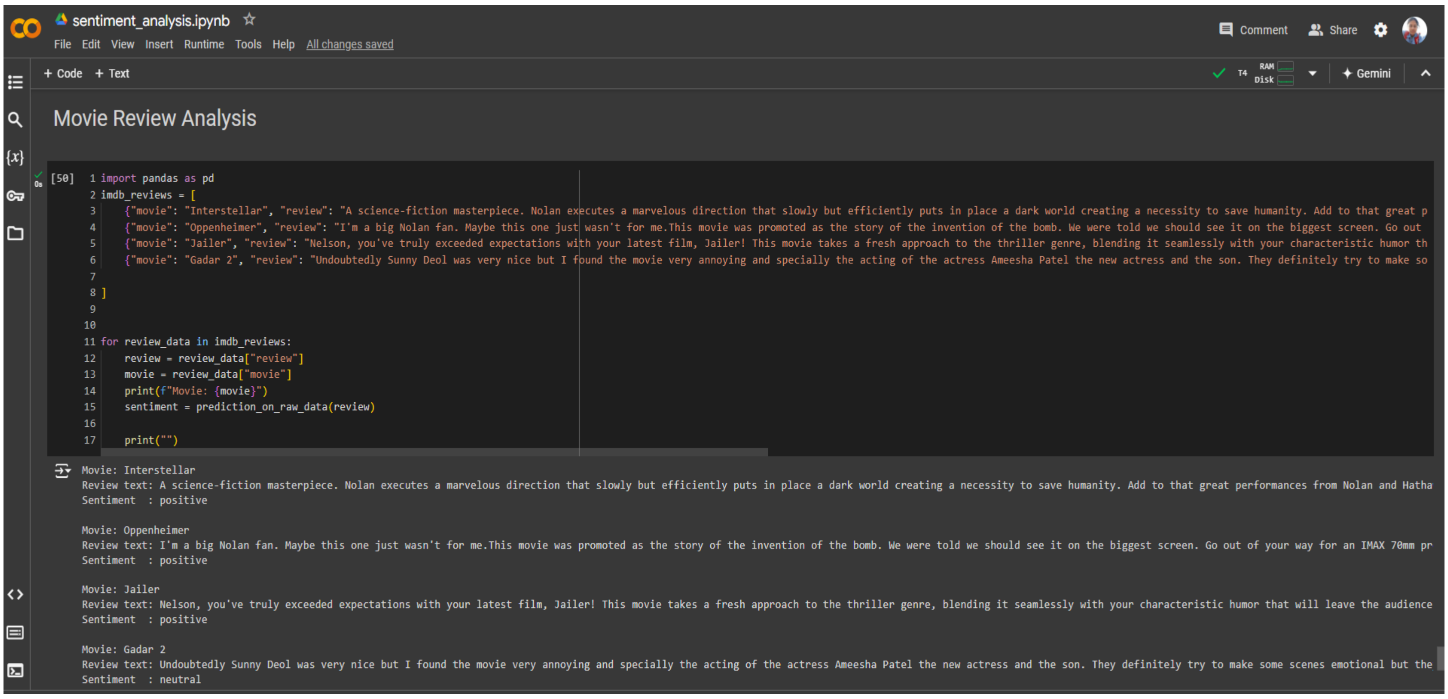
```
plt.ylabel('True sentiment')
  plt.xlabel('Predicted sentiment');
cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)
show_confusion_matrix(df_cm)
idx = 50
review_text = y_review_texts[idx]
true_sentiment = y_test[idx]
pred_df = pd.DataFrame({
  'class_names': class_names,
  'values': y_pred_probs[idx]
})

from textwrap import wrap

print("\n".join(wrap(review_text)))
print()
print(f'True sentiment: {class_names[true_sentiment]}')
sns.barplot(x='values', y='class_names', data=pred_df, orient='h')
plt.ylabel('sentiment')
plt.xlabel('probability')
plt.xlim([0, 1])
plt.show()

# Let's use our model to predict the sentiment of some raw text
raw_text = "It is very useful for seening plans for recharging jio. We can recharge plans
easily by using this my jio app."
def prediction_on_raw_data(raw_text):
  encoded_review = tokenizer.encode_plus(
  raw_text,
  padding='max_length', # Pad sentence to max length
  truncation=True,  #Truncate sentence to max length
  max_length=32,
  add_special_tokens=True, # Add '[CLS]' and '[SEP]'
  return_token_type_ids=False,
  return_attention_mask=True, # Return attention mask
  return_tensors='pt',  # Return torch objects
  )

  input_ids = encoded_review['input_ids'].to(device)
  attention_mask = encoded_review['attention_mask'].to(device)
  output = model(input_ids, attention_mask)
  _, prediction = torch.max(output, dim=1)
```

```
print(f'Review text: {raw_text}')
  print(f'Sentiment  : {class_names[prediction]}')
prediction_on_raw_data(raw_text)
raw_text = "this app is not that good"
prediction_on_raw_data(raw_text)
imdb_google_review = "Logan (2017) delivers a gritty and emotional portrayal of
Wolverine's journey, offering a bittersweet farewell to the beloved character. Hugh
Jackman's performance shines, evoking a mix of nostalgia and poignancy. The film's
somber tone and intense action sequences contribute to its impact, though some might
find its darkness a bit overwhelming"
prediction_on_raw_data(imdb_google_review)

import pandas as pd
imdb_reviews = [
    {"movie": "Interstellar", "review": "A science-fiction masterpiece. Nolan executes a
marvelous direction that slowly but efficiently puts in place a dark world creating a
necessity to save humanity. Add to that great performances from Nolan and Hathaway
plus a great score from Hans Zimmer. The result is on the best science-fiction movies of
all time"},
    {"movie": "Oppenheimer", "review": "I'm a big Nolan fan. Maybe this one just wasn't
for me.This movie was promoted as the story of the invention of the bomb. We were
told we should see it on the biggest screen. Go out of your way for an IMAX 70mm
projection if you can, or at least get a regular 70mm or Laser IMAX."},
    {"movie": "Jailer", "review": "Nelson, you've truly exceeded expectations with your
latest film, Jailer! This movie takes a fresh approach to the thriller genre, blending it
seamlessly with your characteristic humor that will leave the audience in stitches. It's an
exhilarating dark comedy rollercoaster that keeps you hooked throughout."},
    {"movie": "Gadar 2", "review": "Undoubtedly Sunny Deol was very nice but I found
the movie very annoying and specially the acting of the actress Ameesha Patel the new
actress and the son. They definitely try to make some scenes emotional but they fail to
do so and I really hate the writing of the writers. I mean this movie is coming after
decades and a movie for only like stupid people, Pakistan was shown like North
Korea"},

]

for review_data in imdb_reviews:
    review = review_data["review"]
    movie = review_data["movie"]
    print(f"Movie: {movie}")
    sentiment = prediction_on_raw_data(review)

    print("")
 (review) print("")
```

7. RESULT

7.1 SCREENSHORTS

Sentiment analysis on movie reviews



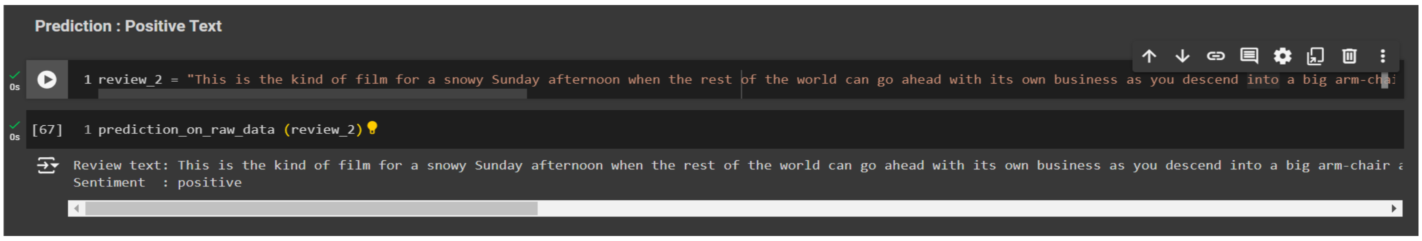
Movie Review Analysis

```
[50]
import pandas as pd
imdb_reviews = [
    {"movie": "Interstellar", "review": "A science-fiction masterpiece. Nolan executes a marvelous direction that slowly but efficiently puts in place a dark world creating a necessity to save humanity. Add to that great p
    {"movie": "Oppenheimer", "review": "I'm a big Nolan fan. Maybe this one just wasn't for me.This movie was promoted as the story of the invention of the bomb. We were told we should see it on the biggest screen. Go out
    {"movie": "Jailer", "review": "Nelson, you've truly exceeded expectations with your latest film, Jailer! This movie takes a fresh approach to the thriller genre, blending it seamlessly with your characteristic humor th
    {"movie": "Gadar 2", "review": "Undoubtedly Sunny Deol was very nice but I found the movie very annoying and specially the acting of the actress Ameesha Patel the new actress and the son. They definitely try to make so

]


for review_data in imdb_reviews:
    review = review_data["review"]
    movie = review_data["movie"]
    print(f"Movie: {movie}")
    sentiment = prediction_on_raw_data(review)

    print("")
```

```
Movie: Interstellar
Review text: A science-fiction masterpiece. Nolan executes a marvelous direction that slowly but efficiently puts in place a dark world creating a necessity to save humanity. Add to that great performances from Nolan and Hatha
Sentiment  : positive

Movie: Oppenheimer
Review text: I'm a big Nolan fan. Maybe this one just wasn't for me.This movie was promoted as the story of the invention of the bomb. We were told we should see it on the biggest screen. Go out of your way for an IMAX 70mm pr
Sentiment  : positive

Movie: Jailer
Review text: Nelson, you've truly exceeded expectations with your latest film, Jailer! This movie takes a fresh approach to the thriller genre, blending it seamlessly with your characteristic humor that will leave the audience
Sentiment  : positive

Movie: Gadar 2
Review text: Undoubtedly Sunny Deol was very nice but I found the movie very annoying and specially the acting of the actress Ameesha Patel the new actress and the son. They definitely try to make some scenes emotional but the
Sentiment  : neutral
```

Fig. Sentiment analysis on movie review

Prediction on positive text



Prediction : Positive Text

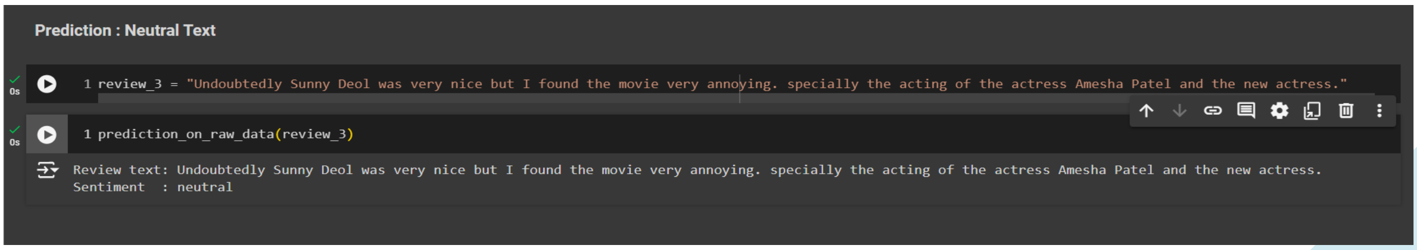
```
review_2 = "This is the kind of film for a snowy Sunday afternoon when the rest of the world can go ahead with its own business as you descend into a big arm-ch
```

```
[67]
prediction_on_raw_data (review_2)
```

```
Review text: This is the kind of film for a snowy Sunday afternoon when the rest of the world can go ahead with its own business as you descend into a big arm-chair a
Sentiment  : positive
```

Fig. Prediction on positive review

Prediction on neutral text



Prediction : Neutral Text

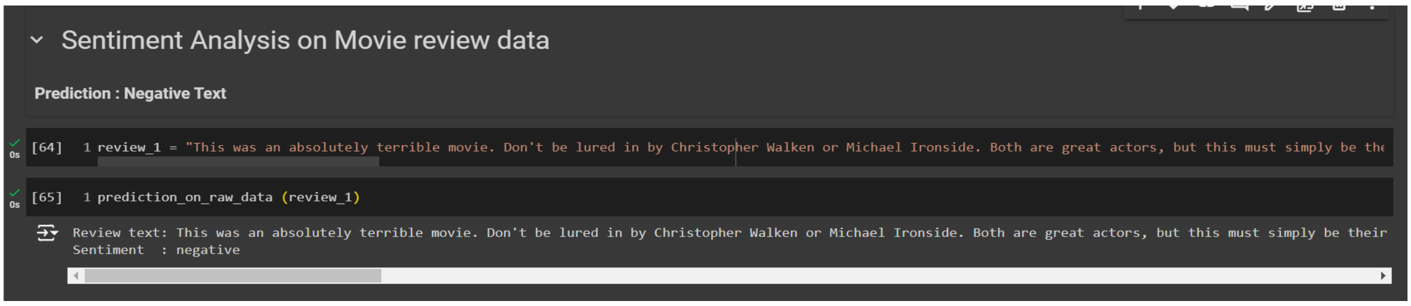
```
review_3 = "Undoubtedly Sunny Deol was very nice but I found the movie very annoying. specially the acting of the actress Amesha Patel and the new actress."
```

```
prediction_on_raw_data(review_3)
```

```
Review text: Undoubtedly Sunny Deol was very nice but I found the movie very annoying. specially the acting of the actress Amesha Patel and the new actress.
Sentiment  : neutral
```

Fig. Prediction on neutral review

Prediction on negative text



Sentiment Analysis on Movie review data

Prediction : Negative Text

```
[64] review_1 = "This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this must simply be the
```

```
[65] prediction_on_raw_data (review_1)
```

```
Review text: This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this must simply be their
Sentiment  : negative
```

Fig. Prediction on negative review

7.2. PLOTS

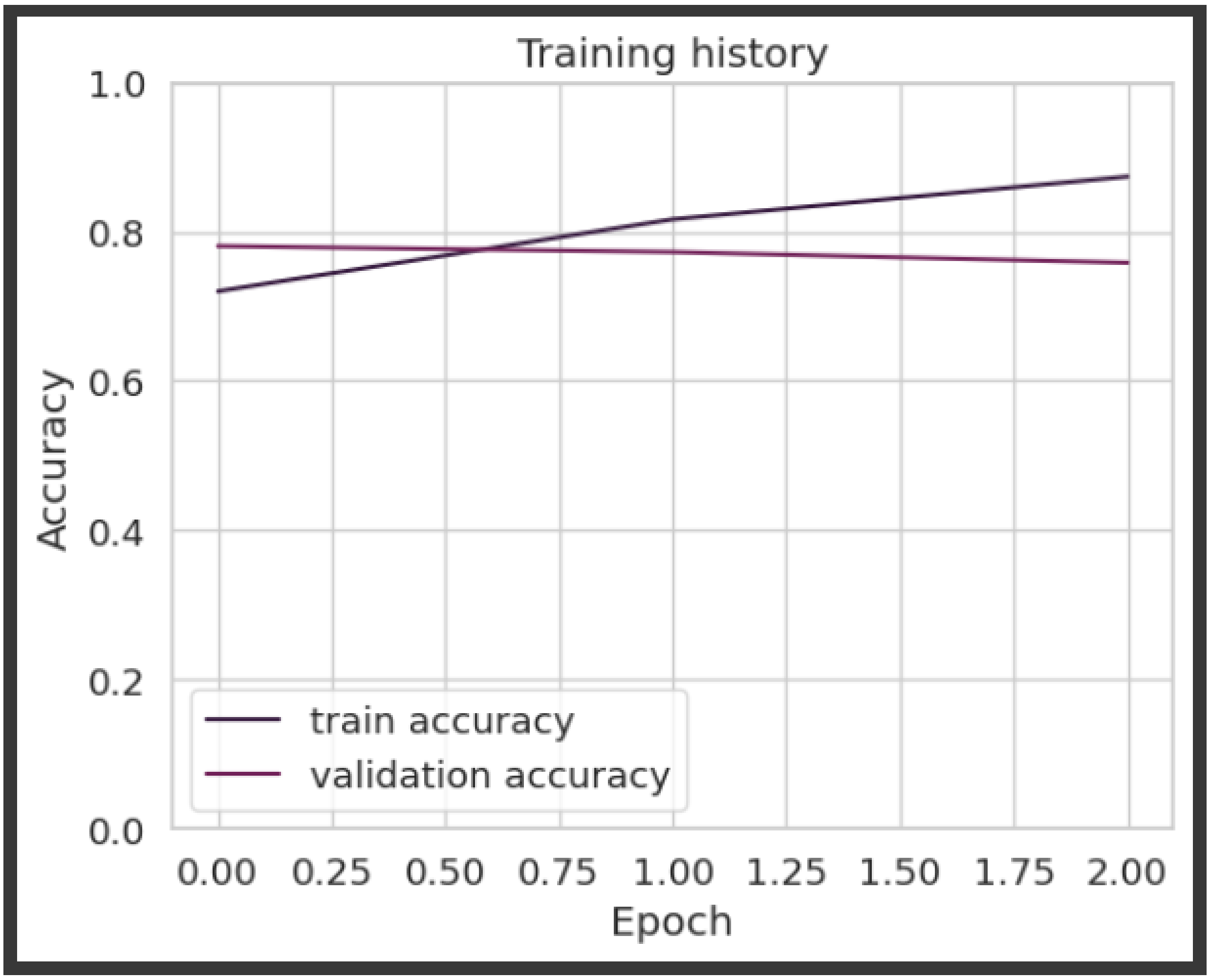


Fig. Accuracy

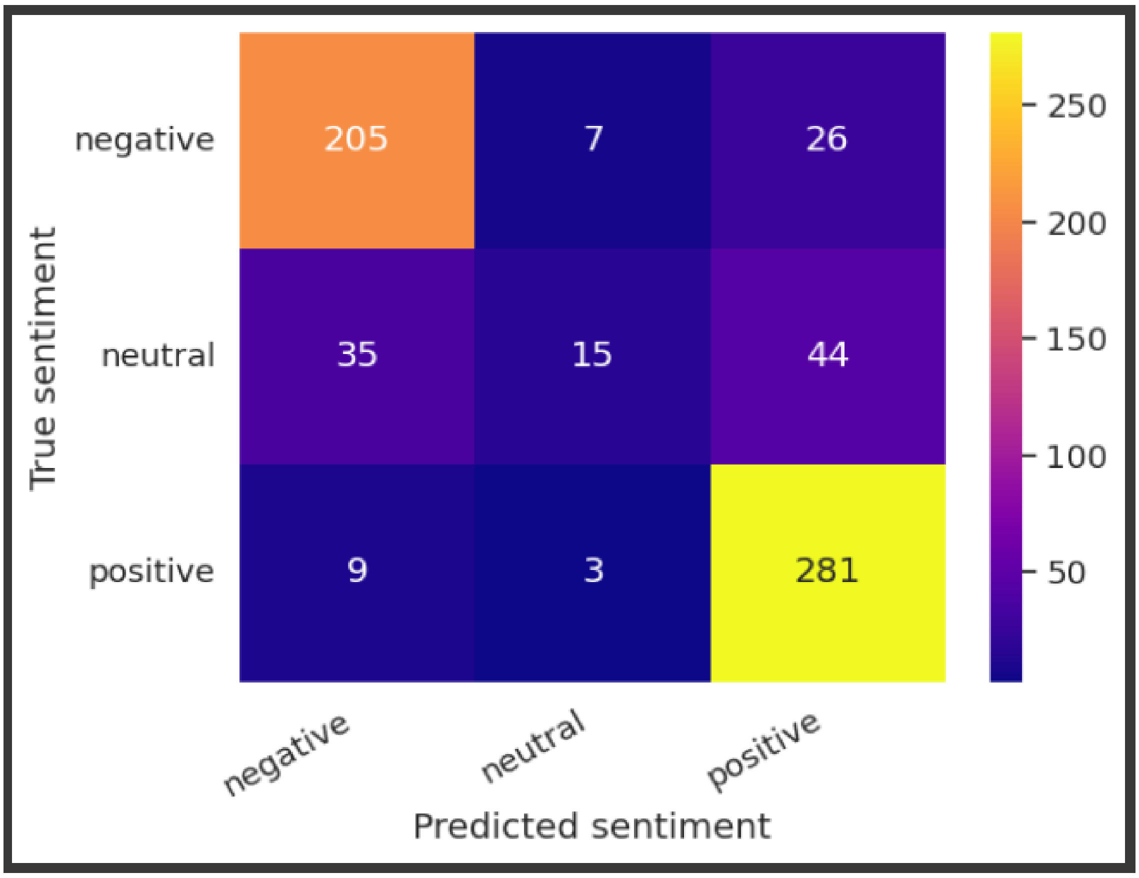


Fig. Heat Map

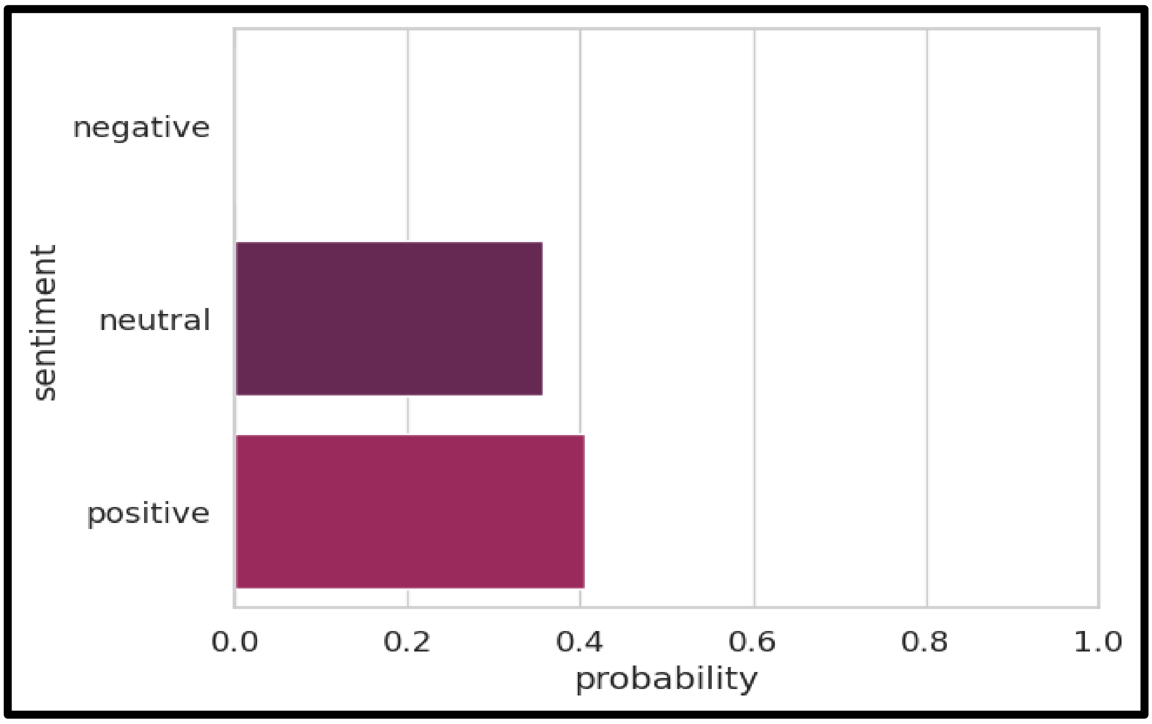


Fig. Bar Plot

8. CONCLUSION

In summary, our project, 'Sentiment Detection from Text Data' marks a significant leap forward in the realm of sentiment analysis. By embracing the power of BERT, we've addressed the limitations of traditional sentiment analysis methods. Our approach offers unparalleled accuracy, making it a valuable tool for assessing and understanding sentiment in text. Importantly, this is not just an academic endeavour; it has real-world implications. Our solution is designed for practical applications, offering insights and decision support to individuals and businesses. BERT's adaptability across languages and domains, combined with its reduced data dependency, positions it as a versatile and scalable solution. Moreover, we've considered ethical considerations, ensuring responsible AI use. As we look ahead, our project stands as a testament to the potential of sentiment analysis, underpinned by the transformative capabilities of BERT, and we are poised for future endeavours that will continue to redefine the landscape of text data analysis.

The project has showcased the efficiency of BERT in capturing contextual information and discerning sentiment, thereby substantiating its relevance in sentiment analysis. By leveraging the bidirectional processing and attention mechanism of BERT, we have been able to achieve notable accuracy in identifying and categorizing sentiment, thus contributing to the growing body of knowledge in the realm of sentiment analysis.

The project has demonstrated the potential of deep learning techniques particularly BERT, in enhancing the accuracy and effectiveness of sentiment analysis processes. By highlighting the capabilities of BERT in capturing subtle contextual cues, this project has underscored the importance of leveraging advanced neural network architectures for sentiment analysis, thereby paving the way for more and robust sentiment interpretation in textual data.

9. FUTURE ENHANCEMENT

- **Multi-lingual Sentiment Analysis**: Extending the BERT-based sentiment analysis model to support multiple languages, thereby broadening its applicability to diverse linguistic contexts.

- **Fine-grained Sentiment Classification**: Refining the model to categorize sentiments into more nuanced categories, such as sentiment intensity levels or specific emotions, to provide deeper insights into textual sentiment**.**

- **Domain-specific Sentiment Analysis:** Adapting the BERT model to specialize in sentiment analysis within specific domains, such as finance, healthcare, entertainment, to cater to industry-specific sentiment interpretation needs**.**

- **Aspect-based Sentiment Analysis:** Enhancing the model's capability to identify and analyse sentiment towards specific aspects or entities within the text, enabling more targeted sentiment assessment.

- **Real-time Sentiment Analysis:** Developing a framework for real-time sentiment analysis utilizing BERT, enabling and continuous sentiment monitoring in dynamic textual data sources, such as social media streams and news articles**.**

REFERENCES

1. Wase, Krutika, Pranali Ramteke, Rushabh Bandewar, Nadim Badole and Bhuvneshwar Kumar. “Sentiment analysis of product review.” (2018).
2. Jadeja, Avani and Indrajeet Rajput. “Feature-Based Sentiment Analysis On Customer Feedback: A Survey.” International journal of engineering research and technology 2 (2013).
3. Jotheeswaran, Jeevanandam. (2015). Sentiment Analysis: A Survey of Current Research and Techniques. International Journal of Innovative Research in Computer and Communication Engineering. 03. 3749-3757. 10.15680/ijircce.2015.0305002.
4. Karim, Mirsa & Das, Smija. (2018). Sentiment Analysis on Textual Reviews. IOP Conference Series: Materials Science and Engineering. 396. 012020. 10.1088/1757-899X/396/1/012020.
5. Kim S-M, Hovy E (2004) Determining the sentiment of opinions In: Proceedings of the 20th international conference on Computational Linguistics, page 1367.. Association for Computational Linguistics, Stroudsburg, PA, USA.
6. Liu B (2010) Sentiment analysis and subjectivity In: Handbook of Natural Language Processing, Second Edition.. Taylor and Francis Group, Boca.
7. Liu B, Hu M, Cheng J (2005) Opinion observer: Analysing and comparing opinions on the web In: Proceedings of the 14th International Conference on World Wide Web, WWW ’05, 342–351.. ACM, New York, NY, USA
8. Pak A, Paroubek P (2010) Twitter as a corpus for sentiment analysis and opinion mining In: Proceedings of the Seventh conference on International Language Resources and Evaluation.. European Languages Resources Association, Valletta, Malta.
9. Pang B, Lee L (2004) A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts In: Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics, ACL ’04.. Association for Computational Linguistics, Stroudsburg, PA, USA.
10. Pang B, Lee L (2008) Opinion mining and sentiment analysis. Found Trends Inf Retr2(1-2): 1–135