**A MINI PROJECT REPORT**

**On**

**Sentiment Analysis on Text Using BERT Neural Network**

**Submitted to Jawaharlal Nehru Technological university for the partial**

**Fulfillment of the Requirement for the Award of the Degree of**

**BACHELOR OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE AND ENGINEERING**

**By**

|  |  |  |
| --- | --- | --- |
| **J ESHWAR** | **-** | **20Q91A05J5** |
| **P DIVYASRI** | **-** | **20Q91A05I6** |
| **S PRASAD** | **-** | **20Q91A05I1** |
| **K ANAND** | **-** | **20Q91A05J8** |

Under the guidance of

**Mr. Shivarao Yannam**

**ASSISTANT PROFESSOR**

****

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

**MALLA REDDY COLLEGE OF ENGINEERING**

**(Approved by AICTE-Permanently Affiliated to JNTU-Hyderabad)**

Accredited by NBA & NAAC, Recognized section 2(f) & 12(B) of UGC New

Delhi ISO 9001:2015 certified Institution

Maisammaguda, Dhulapally (Post via Kompally ), Secunderabad – 500100

**2023 – 2024**

**MALLA REDDY COLLEGE OF ENGINEERING**

**(Approved by AICTE-Permanently Affiliated to JNTU-Hyderabad)**

Accredited by NBA & NAAC, Recognized section 2(f) & 12(B) of UGC New

Delhi ISO 9001:2015 certified Institution

Maisammaguda, Dhulapally (Post via Kompally ), Secunderabad – 500100

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

****

**CERTIFICATE**

This is to certify that the Major Project report on “sentiment Analysis Using BERT Neural Network” Is successfully done by the following students of Department of Computer Science and Engineering of our college in partial fulfilment of the requriment for the award of B.Tech degree in the year 2023-2024. The results embodied in this report have not been submited to any other university for the award of any diploma or degree.

|  |  |
| --- | --- |
| JELLA ESHWAR : | 20Q91A05J5 |
| SANKATI PRASAD : | 20Q91A05I1 |
| PENCHALA DIVYASRI : | 20Q91A05I6 |
| KAMMARI ANAND : | 20Q91A05J8 |

Submitted for the viva voce examination held on:

|  |  |  |
| --- | --- | --- |
| **INTERNAL GUIDE**  **Mr.Shivarao Yannam**  **Asst.Professor** | **H O D**  **Dr.G.Radha Devi**  **Asst.Professor** | **PRINCIPAL**  **Dr.Ashok Reddy**  **Professor** |

|  |  |
| --- | --- |
| **Internal Examiner** | **External Examiner** |

**DECLARATION**

We,the final year students are hereby declaring the the minor project report entitled "Sentiment Analysis Using BERT Neural Network" has done by us under the guidance of Mr.Shivarao Yannam Assistant Professor, Department of CSE is submitted in the partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE OF ENGINEERING.**

The Results embedded in this project report have not been submitted to anyother University or institute for the award of any degree or diploma.

Signature of the Candidate

|  |  |
| --- | --- |
| **JELLA ESHWAR** | **20Q91A05J5** |
| **SANKATI PRASAD** | **20Q91A05I1** |
| **PENCHALA DIVYASRI** | **20Q91A05I6** |
| **KAMMARI ANAND** | **20Q91A05J8** |

**DATE:**

**PLACE : Maisammaguda**

**ACKNOWLEDGEMENT**

First and foremost, we would like to express our immense gratitude towards our institution Malla Reddy College of Engineering, which helped us to attain profound technical skills in the field of Computer Science & Engineering, there by fulilling our most cherished goal.

We are pleased to thank Sri Ch.Malla Reddy,our Founder,Chairman **MRGI**,Sri Ch.Mahender Reddy,Secretary,**MRGI** for providing this opportunity and support throughout the course.

It gives us immense pleasure to acknowledge the perennial inspiration of Dr. Ashok Reddy our beloved principal for his kind co-operation and encouragement in bringing out this task.

We would like to thank Dr.T.V. Reddy our vice principal, Dr.G.Radha Devi HOD,CSE Department for their inspiration adroit guidance and constructive criticism for successful completion of our degree.

We would like to thank Mr. Shivarao Yannam Assistant Professor our internal guide, for his valuable suggestions and the guidence during the exhibition and completion of the project.

Finally, we avail this opportunity to express our deep gratitude to all staff who have contribute their valuable assistance and support making our project success.

|  |  |
| --- | --- |
| **JELLA ESHWAR** | **20Q91A05J5** |
| **SANKATI PRASAD** | **20Q91A05I1** |
| **PENCHALA DIVYASRI** | **20Q91A05I6** |
| **KAMMARI ANAND** | **20Q91A05J8** |

**ABSTRACT**

We introduce a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre- trained BERT model can be fine tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task specific architecture modifications. In this paper, we improve the fine-tuning based approaches by proposing BERT: Bidirectional Encoder Representations from Transformers.

BERT alleviates the previously mentioned unidirectionality constraint by using a “masked language model” (MLM) pre-training objective, inspired by the Cloze task (Taylor, 1953). The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. Unlike left-to-right language model pre training, the MLM objective enables the representation to fuse the left and the right context, which allows us to pre train a deep bidirectional Transformer. In addition to the masked language model, we also use a “next sentence prediction” task that jointly pretrains text-pair representations.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **CERTIFICATE** | **i** |
| **DECLARATION** | **ii** |
| **ACKNOWLEDGEMENT** | **iii** |
| **ABSTRACT** | **iv**  **v** |
| **TABLE OF CONTENTS** | **vi**  **vii** |
| **LIST OF FIGURES** | **viii** |
| **LIST OF SCREENSHOTS** | **ix** |
| **LIST OF ABBREVIATIONS** | **x** |
| **CHAPTER 1 : INTRODUCTION**   |  | | --- | | 1.1 Introduction | | **1** |
| **CHAPTER 2 : LITERATURE SURVEY**   |  | | --- | | 2.1 Literature survey | | **2** |
| **CHAPTER 3 : SYSTEM ANALYSIS**   |  | | --- | | 3.1 Existing System | | 3.2 Drawbacks | | 3.3 Proposed System | | 3.4 Advantages | | 3.5 System Requirements | | **5**  **6**  **7**  **7**  **8**  **9** |
| **CHAPTER 4 : SYSTEM DESIGN**   |  | | --- | | 4.1 System Architecture | | 4.2 Modules | | 4.3 UML Diagrams | | **10**  **11**  **14**  **16** |
| **CHAPTER 5 : SYSTEM IMPLEMENTATION**   |  | | --- | | 5.1 Deep Learning  5.2 Natural Language Processing |   5.3 Python  5.4 Fundamental of Python  5.5 Google Colab  5.6 Visual Studio Code  5.7 Algorithm  5.8 Packages  5.9 Source Code | **23**  **24**  **27**  **28**  **33**  **34**  **35**  **37**  **38**  **40** |
| **CHAPTER 6 : TESTING**   |  | | --- | | 6.1 Testing  6.2 Testing Methods  6.3 Validation | | **51**  **52**  **52**  **70** |
| **CHAPTER 7 : RESULTS**   |  | | --- | | 7.1 Screenshots | | **73**  **73** |

|  |  |  |
| --- | --- | --- |
| **CHAPTER 8 : CONCLUSION**   |  | | --- | | 8.1 Conclusion | | **80**  **81** |
| **CHAPTER 9 : FUTURE ENHANCEMENT**   |  | | --- | | 9.1 Future Enhancement | | **82**  **83** |
| **REFERENCES** | **85** |

**LIST OF FIGURES**

**Figure No Name of the figure Page No**

4.1.1 System Architecture 11

4.2.1 Modules 14

4.3.1 Class Diagram 17

4.3.2 Use case Diagram 18

4.3.3 Sequence Diagram 19

4.3.4 Activity Diagram 20

4.3.5 Component Diagram 21

4.3.6 Deployment Diagram 22

5.1.1 Deep learning image 33

5.2.1 NLP Image 34

7.2.1 Training history 77

7.2.2 Heat Map 78

7.2.3 Bar plot 79

**LIST OF SCREENSHORTS**

**Figure No Name of the Screenshort Page No**

7.1.1 Sentiment analysis on movie review 73

7.1.2 Prediction on positive text 74

7.1.3 Prediction on negative text 75

7.1.4 Prediction on neutral text 75

7.1.5 Sentiment analysis on tweet 76

**LIST OF ABBREVIATIONS**

**S.NO SHORT FORM FULL FORM**

1. BERT Bidirectional Encoder Representation using

Transformers

2 GPU Graphical Processing Unit

3 TF-IDF Term Frequency-Inverse Document Frequency

4 LSTM Long Short-Term Memory

5 CNN Convolutional Neural Network

# 

# 

# CHAPTER-1

# INTRODUCTION

# 1.1 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 endeavours 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 endeavours 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.

**CHAPTER-2**

**LITERATURE SURVEY**

**2.1 LITERATURE SURVEY**

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.

**CHAPTER-3**

**SYSTEM ANALYSIS**

**3.1 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**.**

**3.2 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.

# 3.3 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.

# 3.4 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.

# 3.5 SYSTEM REQUIREMENTS

**SOFTWARE REQUIREMENTS :**

IDE  **:** Google colab

LANGUAGE  **:** Python

LIBRARIES  **:** BERT, SCI-KIT learn, Pandas,Numpy,Matplotlib

**HARDWARE REQUIREMENTS:**

RAM : 16GB Minimum

CPU : I5 Processor

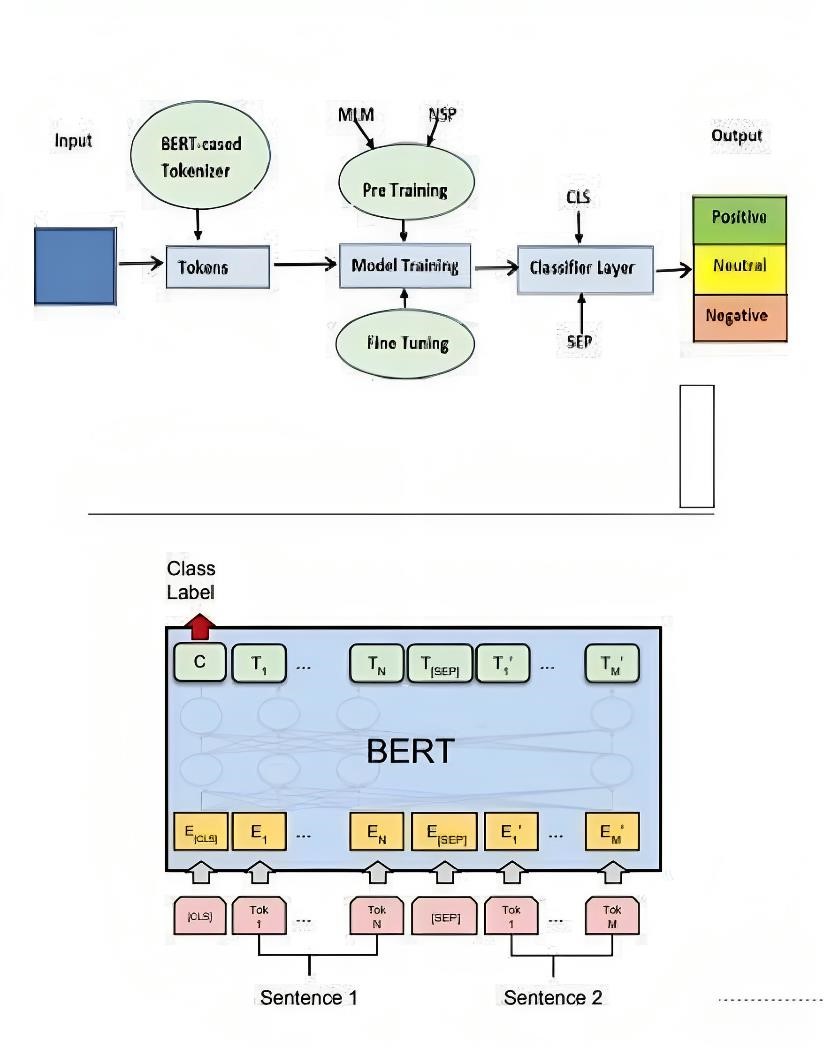
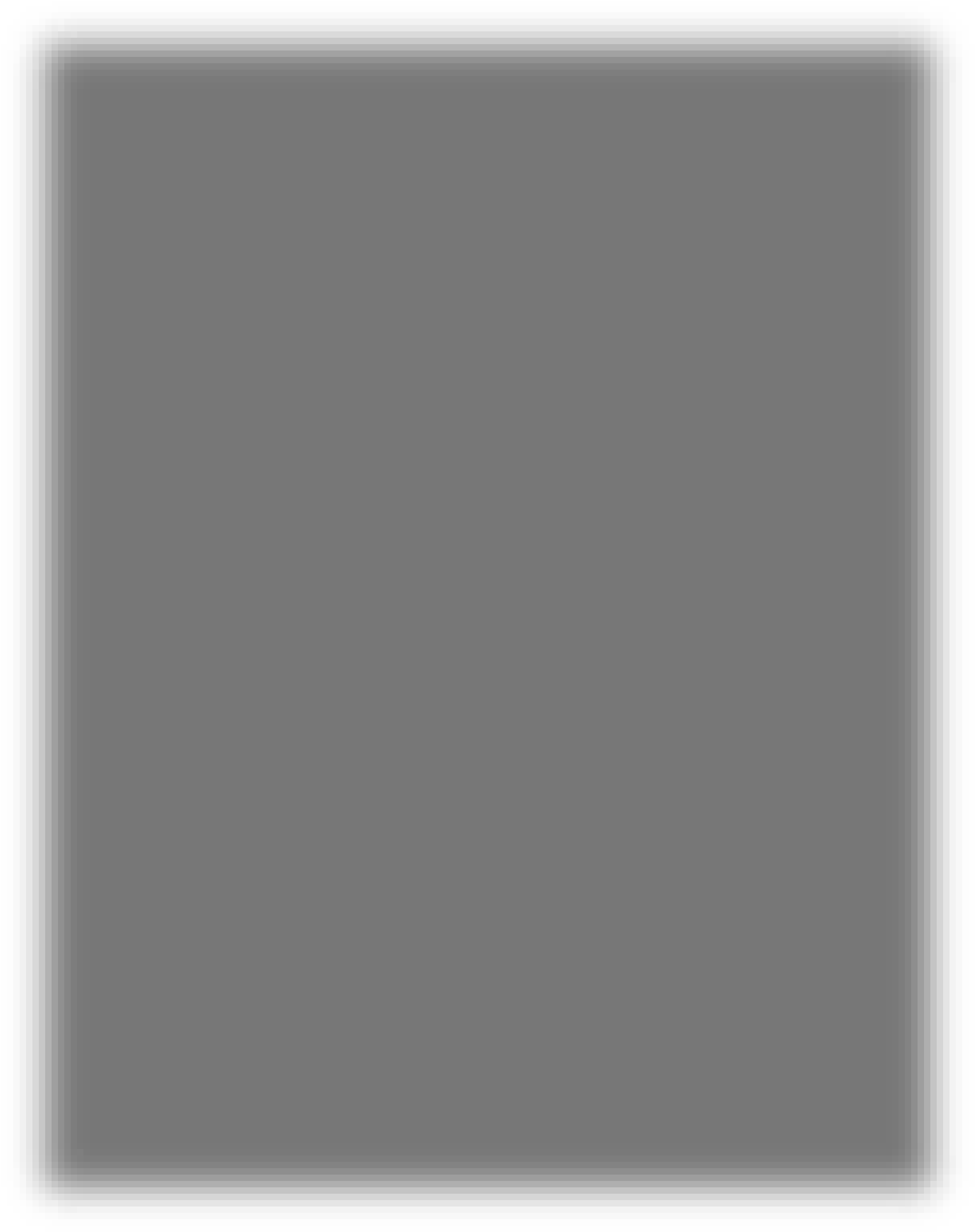
SSD : 256 GB

GPU

**CHAPTER-4**

**SYSTEM DESIGN**

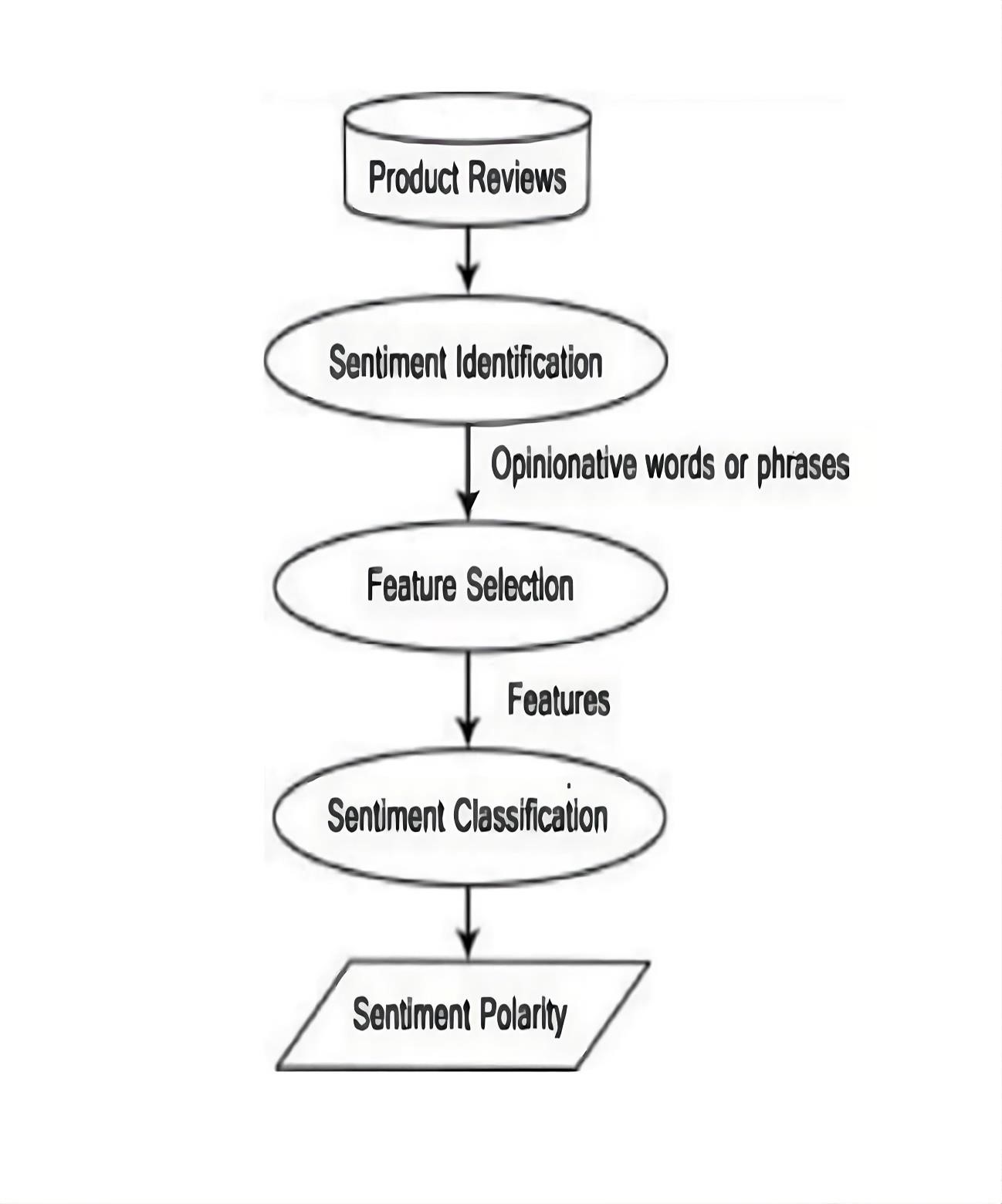
**4.1 SYSTEM ARCHITECTURE**



**Fig 4.1.1**

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 tothe input text, such as "positive," "negative," or "neutral," based on the training it received.

**4.2 Modules**



**Fig 4.2.1**

**1. Product Review:** 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.

**2. Sentiment Identification**: This step involves identifying and extracting the sentiment from the product reviews. Sentiment identification can be thought of as the preliminary analysis, where you determine whether the sentiment is positive, negative, or neutral. BERT, as mentioned earlier, plays a crucial role in this step.

**3. Features Selection:** Feature selection 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.

**4. 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.

**5. 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 reviews.

**4.3 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.

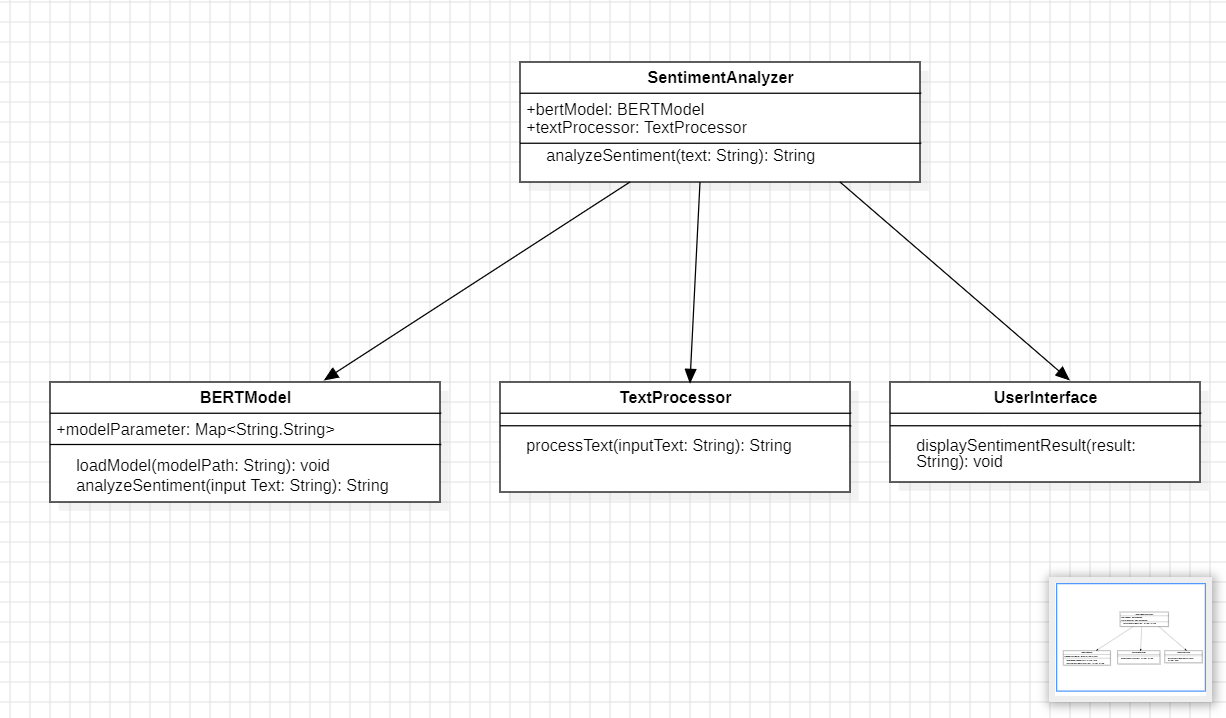
****In practical terms, class diagrams aid software developers in the design phase by providing a clear overview of the system's building blocks. Stakeholders can grasp the interconnections between classes, fostering better collaboration and understanding of the overall system structure. As an integral 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.

Fig 4.3.1

**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.

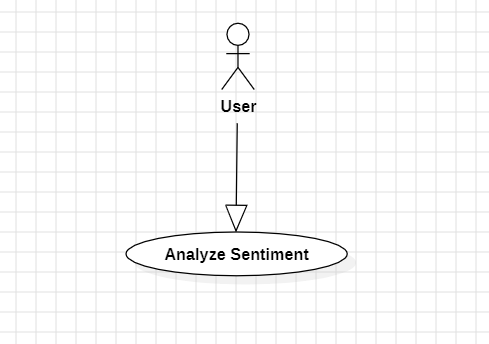


Fig 4.3.2

**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.

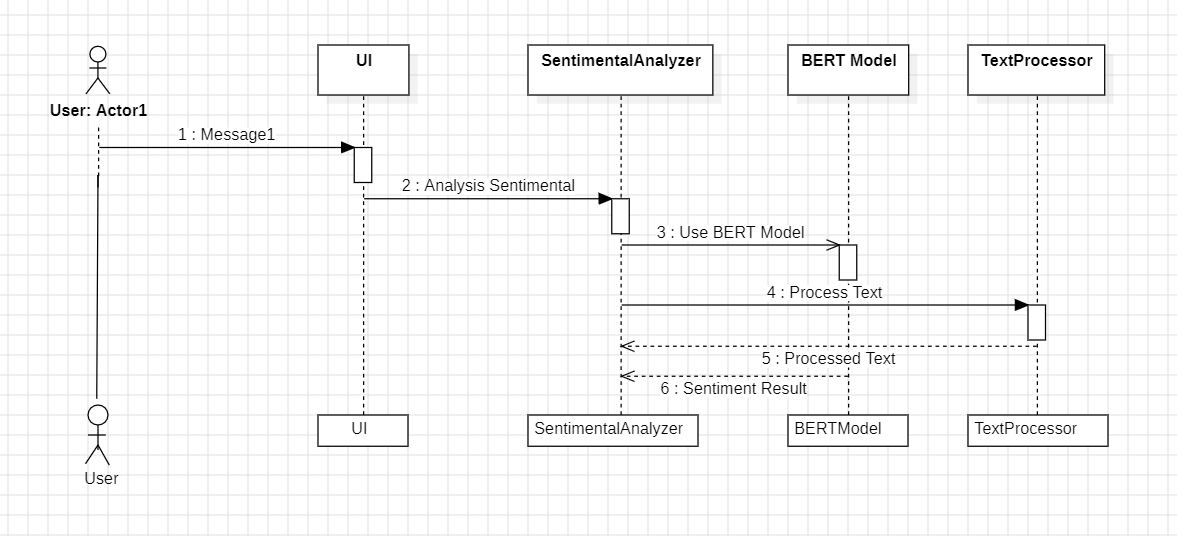
****

Fig 4.3.3

**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.

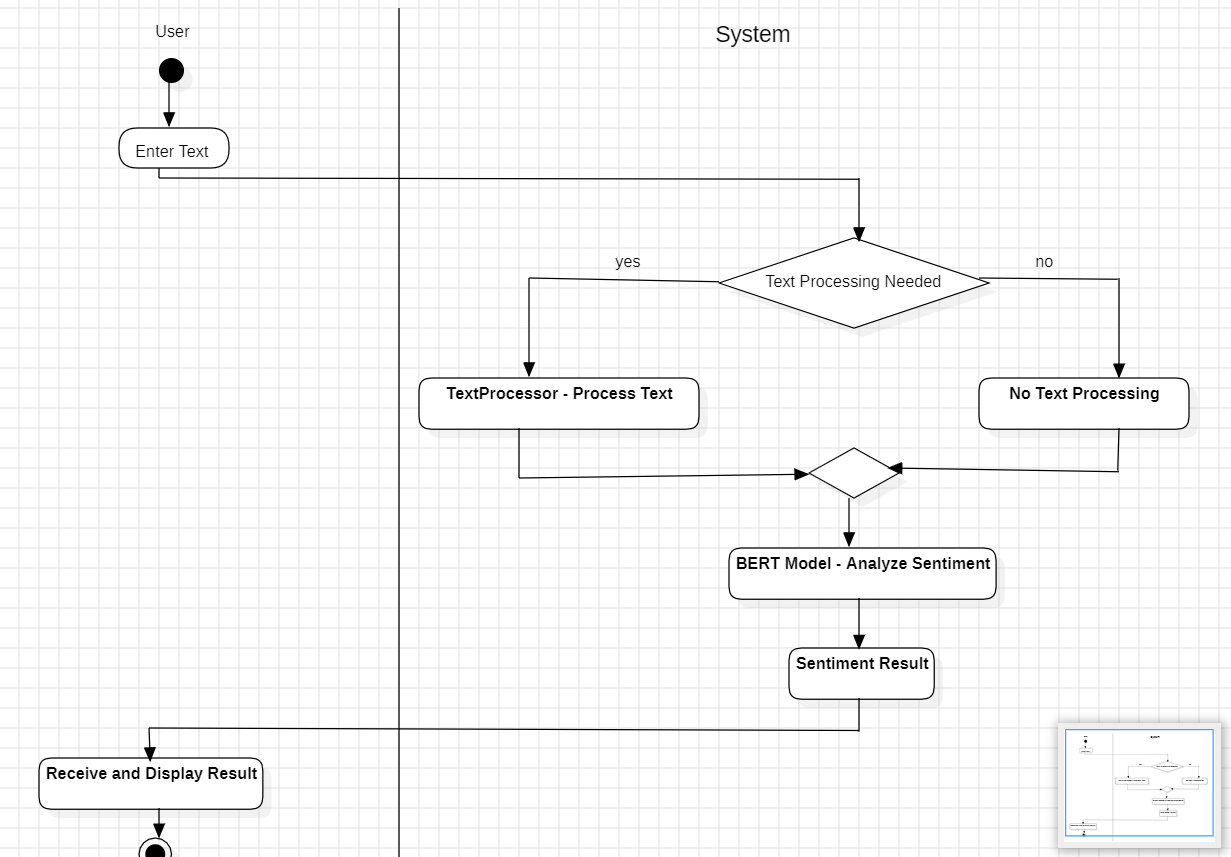
Activity diagrams are valuable for visualizing and understanding complex processes, identifying potential bottlenecks, and improving the efficiency of workflows. They are commonly used during the analysis and design phases of software development, offering a clear and intuitive way to communicate the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

Fig 4.3.4

**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.

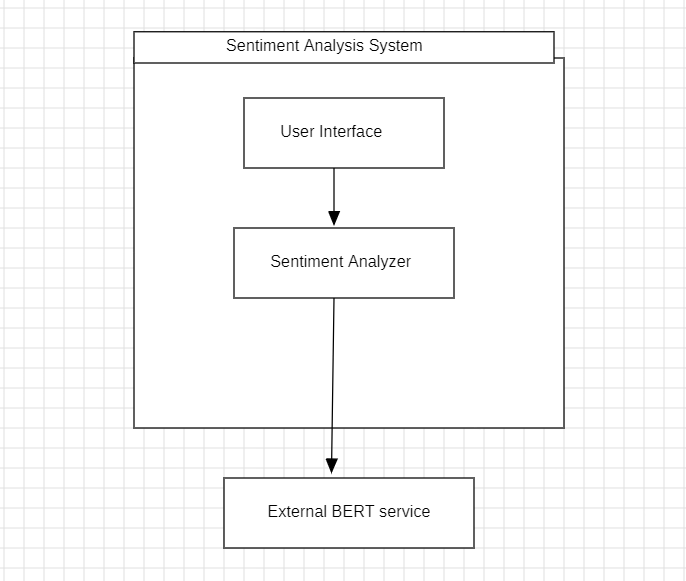
****

Fig 4.3.5

**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.

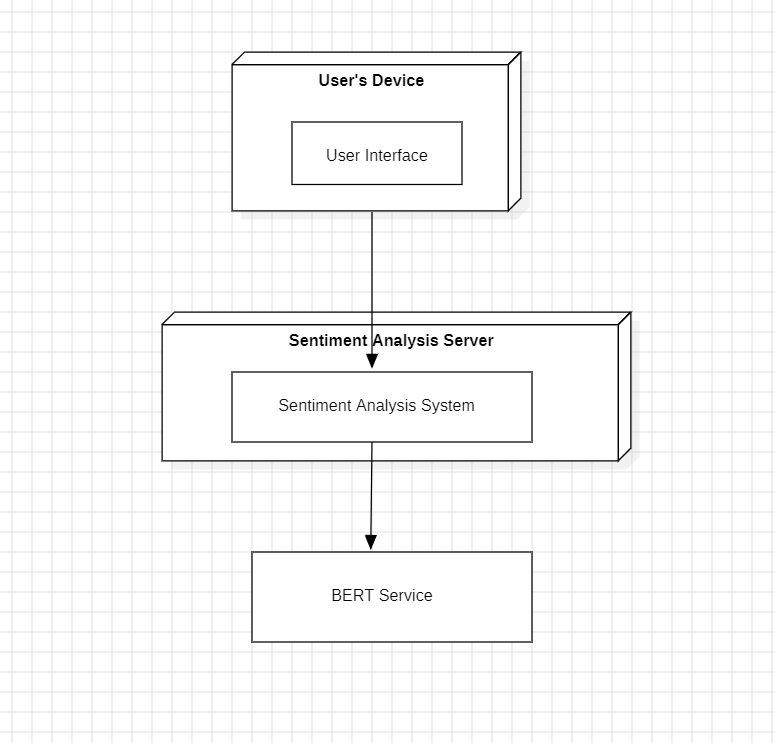
Deployment diagrams are useful for understanding the physical aspects of a system's architecture, including server configurations, network topology, and the placement of software components on hardware. They are commonly employed in the later stages of system design and development.

Fig 4.3.6

**CHAPTER-5**

**SYSTEM IMPLEMENTATION**

* 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.

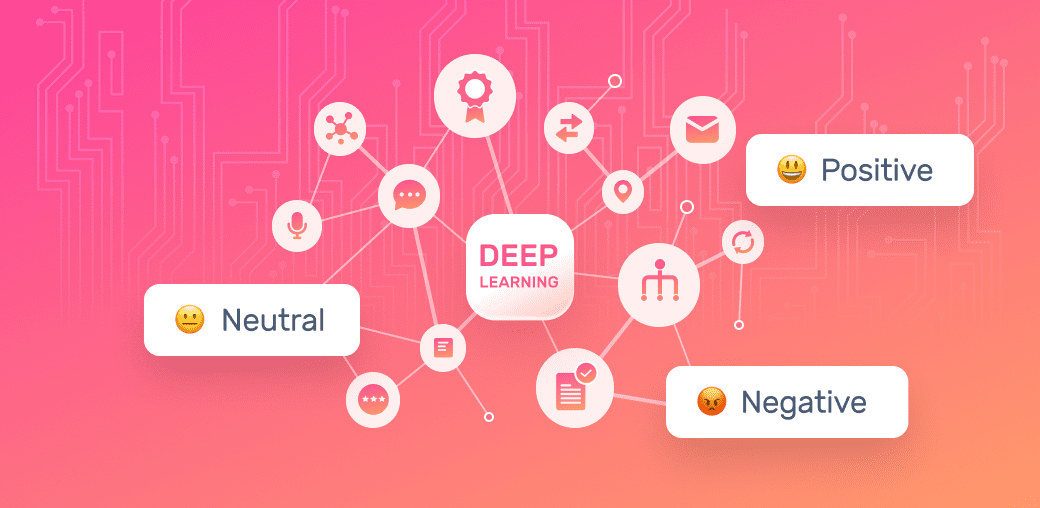
****

Fig 5.1.1

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.

* 1. **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.

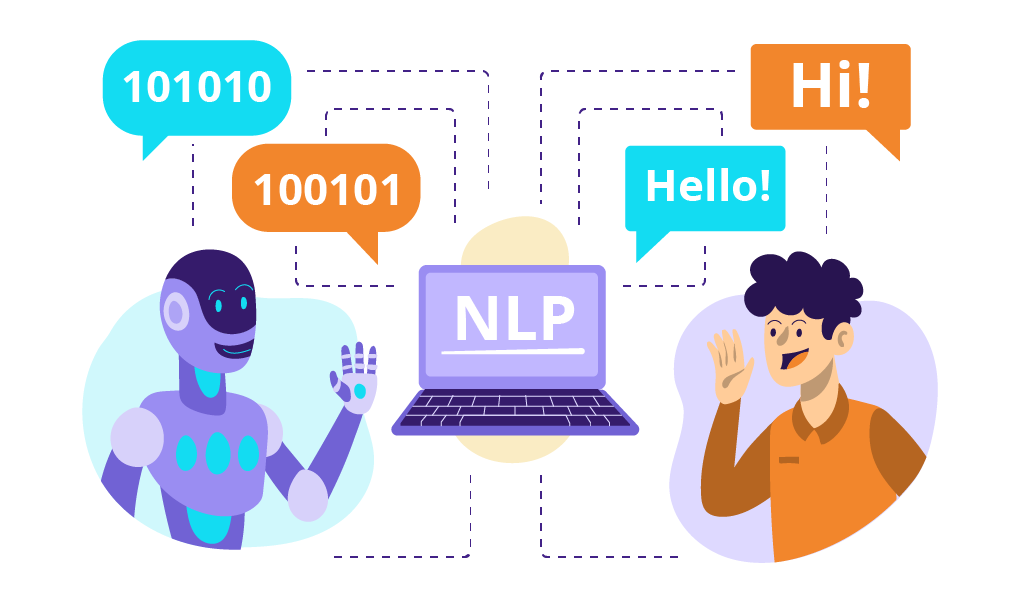


Fig 5.2.1

**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.

* 1. **Python**

Python programming language is used for building the machine learning model

* + 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.

**5.3.2 Python Features**

1. Easy to Use

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

2) 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.

3) Cross-platform language

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

4) Free and Open Source

Python language is freely available.

5) Object-Oriented language

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

6) 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.

7) Large Standard Library

Python has a large and broad library.

8) GUI Programming

Graphical user interfaces can be developed using Python.

9) Integrated

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

**5.3.3. Python History**

Python laid its foundation in the late 1980s.The implementation of Python was started in the December 1989 by Guido Van Rossum at CWI in Netherland. ABC programming language is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.

**5.3.4. Python Version**

Python programming language is being updated regularly with new features and support. There are a lot of updates in python versions, started from 1994 to current date. A list of python versions with its released date is given below

|  |  |
| --- | --- |
| **Python Version** | **Released Date** |
| Python 1.0 | January 1994 |
| Python 1.5 | December 31, 1997 |
| Python 1.6 | September 5, 2000 |
| Python 2.0 | October 16, 2000 |
| Python 2.1 | April 17, 2001 |
| Python 2.2 | December 21, 2001 |
| Python 2.3 | July 29, 2003 |
| Python 2.4 | November 30, 2004 |
| Python 2.5 | September 19, 2006 |
| Python 2.6 | October 1, 2008 |
| Python 2.7 | July 3, 2010 |
| Python 3.0 | December 3, 2008 |
| Python 3.1 | June 27, 2009 |
| Python 3.2 | February 20, 2011 Python |
| Python 3.3 | 3 September 29, 2012 |

Table 5.3.1 : Python version table

**5.3.5. 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

**5.3.6 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:

1. Click on Start button -> All Programs -> Python -> IDLE(Python GUI)
2. 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

* 1. **FUNDAMENTALS OF PYTHON**

This section contains the basic fundamentals of Python.

* + 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.

**5.4.2 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')

**5.4.3 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'}

* 1. **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.

* 1. **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.

**5.7 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.

* 1. **Packages**

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

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

5.8.2. 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.

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

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

5.8.5. 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.

5.8.6. 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.

5.8.7. 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.

5.8.8. 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.

* 1. **Source Code**

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 warnings

warnings.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")

# 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)

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)

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

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

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]}')

raw\_text = "this app is not that good"

prediction\_on\_raw\_data(raw\_text)

**CHAPTER-6**

**TESTING**

**6.1 TESTING**

**The various levels of testing are**

1. White Box Testing
2. Black Box Testing
3. Unit Testing
4. Functional Testing
5. Performance Testing
6. Integration Testing
7. Objective
8. Integration Testing
9. Validation Testing
10. System Testing
11. Structure Testing
12. Output Testing
13. User Acceptance Testing
14. **White Box Testing**

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).

While white-box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

White-box test design techniques include:

* Control flow testing
* Data flow testing
* Branch testing
* Path testing
* Statement coverage
* Decision coverage

White-box testing is a method of testing the application at the level of the source code. The test cases are derived through the use of the design techniques mentioned above: control flow testing, data flow testing, branch testing, path testing, statement coverage and decision coverage as well as modified condition/decision coverage. White-box testing is the use of these techniques as guidelines to create an error free environment by examining any fragile code.

These White-box testing techniques are the building blocks of white-box testing, whose essence is the careful testing of the application at the source code level to prevent any hidden errors later on. These different techniques exercise every visible path of the source code to minimize errors and create an error-free environment. The whole point of white-box testing is the ability to know which line of the code is being executed and being able to identify what the correct output should be.

Levels

1. Unit testing. White-box testing is done during unit testing to ensure that the code is working as intended, before any integration happens with previously tested code. White-box testing during unit testing catches any defects early on and aids in any defects that happen later on after the code is integrated with the rest of the application and therefore prevents any type of errors later on.
2. Integration testing. White-box testing at this level are written to test the interactions of each interface with each other. The Unit level testing made sure that each code was tested and working accordingly in an isolated environment and integration examines the correctness of the behaviour in an open environment through the use of white-box testing for any interactions of interfaces that are known to the programmer.
3. Regression testing. White-box testing during regression testing is the use of recycled white-box test cases at the unit and integration testing levels.

White-box testing's basic procedures involve the understanding of the source code that you are testing at a deep level to be able to test them. The programmer must have a deep understanding of the application to know what kinds of test cases to create so that every visible path is exercised for testing. Once the source code is understood then the source code can be analysed for test cases to be created. These are the three basic steps that white-box testing takes in order to create test cases:

1. Input, involves different types of requirements, functional specifications, detailed designing of documents, proper source code, security specifications. This is the preparation stage of white-box testing to layout all of the basic information.
2. Processing Unit, involves performing risk analysis to guide whole testing process, proper test plan, execute test cases and communicate results. This is the phase of building test cases to make sure they thoroughly test the application the given results are recorded accordingly.
3. Output, prepare final report that encompasses all of the above preparations and results.
4. **Black Box Testing**

Black-box testing is a method of software testing that examines the functionality of an application (e.g. what the software does) without peering into its internal structures or workings (see white-box testing). This method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher-level testing, but can also dominate unit testing as well

Test procedures

Specific knowledge of the application's code/internal structure and programming knowledge in general is not required. The tester is aware of *what* the software is supposed to do but is not aware of *how* it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of *how* the software produces the output in the first place.

### Test cases

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily *functional* in nature, *non-functional* tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output without any knowledge of the test object's internal structure.

### Test design techniques

Typical black-box test design techniques include:

* Decision table testing
* All-pairs testing
* State transition tables
* Equivalence partitioning
* Boundary value analysis

1. **Unit Testing**

In computer programming, unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are created by programmers or occasionally by white box testers during the development process.

Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended. Its implementation can vary from being very manual (pencil and paper)to being formalized as part of build automation.

Testing will not catch every error in the program, since it cannot evaluate every execution path in any but the most trivial programs. The same is true for unit testing. Additionally, unit testing by definition only tests the functionality of the units themselves. Therefore, it will not catch integration errors or broader system-level errors (such as functions performed across multiple units, or non-functional test areas such as performance).

Unit testing should be done in conjunction with other software testing activities, as they can only show the presence or absence of particular errors; they cannot prove a complete absence of errors. In order to guarantee correct behaviour for every execution path and every possible input, and ensure the absence of errors, other techniques are required, namely the application of formal methods to proving that a software component has no unexpected behaviour.

Software testing is a combinatorial problem. For example, every Boolean decision statement requires at least two tests: one with an outcome of "true" and one with an outcome of "false". As a result, for every line of code written, programmers often need 3 to 5 lines of test code.

 This obviously takes time and its investment may not be worth the effort. There are also many problems that cannot easily be tested at all – for example those that are nondeterministic or involve multiple threads. In addition, code for a unit test is likely to be at least as buggy as the code it is testing. Fred Brooks in The Mythical Man-Month quotes: *never take two chronometers to sea. Always take one or three.* Meaning, if two chronometers contradict, how do you know which one is correct?

Another challenge related to writing the unit tests is the difficulty of setting up realistic and useful tests. It is necessary to create relevant initial conditions so the part of the application being tested behaves like part of the complete system. If these initial conditions are not set correctly, the test will not be exercising the code in a realistic context, which diminishes the value and accuracy of unit test results.

To obtain the intended benefits from unit testing, rigorous discipline is needed throughout the software development process. It is essential to keep careful records not only of the tests that have been performed, but also of all changes that have been made to the source code of this or any other unit in the software. Use of a version control system is essential. If a later version of the unit fails a particular test that it had previously passed, the version-control software can provide a list of the source code changes (if any) that have been applied to the unit since that time.

It is also essential to implement a sustainable process for ensuring that test case failures are reviewed daily and addressed immediately if such a process is not implemented and ingrained into the team's workflow, the application will evolve out of sync with the unit test suite, increasing false positives and reducing the effectiveness of the test suite.

Unit testing embedded system software presents a unique challenge: Since the software is being developed on a different platform than the one it will eventually run on, you cannot readily run a test program in the actual deployment environment, as is possible with desktop programs

1. **Functional Testing**

Functional testing is a quality assurance (QA) process and a type of black box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (not like in white-box testing). Functional Testing usually describes *what* the system does.

Functional testing differs from system testing in that functional testing "*verifies* a program by checking it against ... design document(s) or specification(s)", while system testing "*validate* a program by checking it against the published user or system requirements" (Kane, Falk, Nguyen 1999, p. 52).

Functional testing typically involves five steps .The identification of functions that the software is expected to perform

1. The creation of input data based on the function's specifications
2. The determination of output based on the function's specifications
3. The execution of the test case
4. The comparison of actual and expected outputs
5. **Performance Testing**

In software engineering, performance testing is in general testing performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Performance testing is a subset of performance engineering, an emerging computer science practice which strives to build performance into the implementation, design and architecture of a system.

**Testing types**

### Load testing

Load testing is the simplest form of performance testing. A load test is usually conducted to understand the behaviour of the system under a specific expected load. This load can be the expected concurrent number of users on the application performing a specific number of transactions within the set duration. This test will give out the response times of all the important business critical transactions. If the database, application server, etc. are also monitored, then this simple test can itself point towards bottlenecks in the application software.

### Stress testing

Stress testing is normally used to understand the upper limits of capacity within the system. This kind of test is done to determine the system's robustness in terms of extreme load and helps application administrators to determine if the system will perform sufficiently if the current load goes well above the expected maximum.

### Soak testing

Soak testing, also known as endurance testing, is usually done to determine if the system can sustain the continuous expected load. During soak tests, memory utilization is monitored to detect potential leaks. Also important, but often overlooked is performance degradation. That is, to ensure that the throughput and/or response times after some long period of sustained activity are as good as or better than at the beginning of the test. It essentially involves applying a significant load to a system for an extended, significant period of time. The goal is to discover how the system behaves under sustained use.

### Spike testing

Spike testing is done by suddenly increasing the number of or load generated by, users by a very large amount and observing the behaviour of the system. The goal is to determine whether performance will suffer, the system will fail, or it will be able to handle dramatic changes in load.

### Configuration testing

Rather than testing for performance from the perspective of load, tests are created to determine the effects of configuration changes to the system's components on the system's performance and behaviour. A common example would be experimenting with different methods of load-balancing.

**Isolation testing**

Isolation testing is not unique to performance testing but involves repeating a test execution that resulted in a system problem. Often used to isolate and confirm the fault domain.

1. **Integration Testing**

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

Purpose

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface.

Test cases are constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.

Some different types of integration testing are big bang, top-down, and bottom-up. Other Integration Patterns are: Collaboration Integration, Backbone Integration, Layer Integration, Client/Server Integration, Distributed Services Integration and High-frequency Integration.

### Big Bang

In this approach, all or most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. The Big Bang method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing.

A type of Big Bang Integration testing is called Usage Model testing. Usage Model Testing can be used in both software and hardware integration testing. The basis behind this type of integration testing is to run user-like workloads in integrated user-like environments. In doing the testing in this manner, the environment is proofed, while the individual components are proofed indirectly through their use.

Usage Model testing takes an optimistic approach to testing, because it expects to have few problems with the individual components. The strategy relies heavily on the component developers to do the isolated unit testing for their product. The goal of the strategy is to avoid redoing the testing done by the developers, and instead flesh-out problems caused by the interaction of the components in the environment.

For integration testing, Usage Model testing can be more efficient and provides better test coverage than traditional focused functional integration testing. To be more efficient and accurate, care must be used in defining the user-like workloads for creating realistic scenarios in exercising the environment. This gives confidence that the integrated environment will work as expected for the target customers.

### Top-down and Bottom-up

Bottom-Up Testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher-level components. The process is repeated until the component at the top of the hierarchy is tested.

All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower-level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

Top-Down Testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.

Sandwich Testing is an approach to combine top-down testing with bottom up testing.

The main advantage of the Bottom-Up approach is that bugs are more easily found. With Top-Down, it is easier to find a missing branch link

1. **Validation Testing**

Verification and Validation are independent procedures that are used together for checking that a product, service, or system meets requirements and specifications and that it full fills its intended purpose.  These are critical components of a quality management system such as ISO 9000. The words "verification" and "validation" are sometimes preceded with "Independent" (or IV&V), indicating that the verification and validation is to be performed by a disinterested third party.

It is sometimes said that validation can be expressed by the query "Are you building the right thing?" and verification by "Are you building it right?"In practice, the usage of these terms varies. Sometimes they are even used interchangeably.

The PMBOK guide, an IEEE standard, defines them as follows in its 4th edition

* "Validation. The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with *verification*."
* "Verification. The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with *validation*."
* Verification is intended to check that a product, service, or system (or portion thereof, or set thereof) meets a set of initial design specifications. In the development phase, verification procedures involve performing special tests to model or simulate a portion, or the entirety, of a product, service or system, then performing a review or analysis of the modelling results. In the post-development phase, verification procedures involve regularly repeating tests devised specifically to ensure that the product, service, or system continues to meet the initial design requirements, specifications, and regulations as time progresses. It is a process that is used to evaluate whether a product, service, or system complies with regulations, specifications, or conditions imposed at the start of a development phase. Verification can be in development, scale-up, or production. This is often an internal process.
* Validation is intended to check that development and verification procedures for a product, service, or system (or portion thereof, or set thereof) result in a product, service, or system (or portion thereof, or set thereof) that meets initial requirements. For a new development flow or verification flow, validation procedures may involve modelling either flow and using simulations to predict faults or gaps that might lead to invalid or incomplete verification or development of a product, service, or system (or portion thereof, or set thereof). A set of validation requirements, specifications, and regulations may then be used as a basis for qualifying a development flow or verification flow for a product, service, or system (or portion thereof, or set thereof). Additional validation procedures also include those that are designed specifically to ensure that modifications made to an existing qualified development flow or verification flow will have the effect of producing a product, service, or system (or portion thereof, or set thereof) that meets the initial design requirements, specifications, and regulations; these validations help to keep the flow qualified. It is a process of establishing evidence that provides a high degree of assurance that a product, service, or system accomplishes its intended requirements. This often involves acceptance of fitness for purpose with end users and other product stakeholders. This is often an external process.
* It is sometimes said that validation can be expressed by the query "Are you building the right thing?" and verification by "Are you building it right?". "Building the right thing" refers back to the user's needs, while "building it right" checks that the specifications are correctly implemented by the system. In some contexts, it is required to have written requirements for both as well as formal procedures or protocols for determining compliance.
* It is entirely possible that a product passes when verified but fails when validated. This can happen when, say, a product is built as per the specifications but the specifications themselves fail to address the user’s needs.

Activities

Verification of machinery and equipment usually consists of design qualification (DQ), installation qualification (IQ), operational qualification (OQ), and performance qualification (PQ). DQ is usually a vendor's job. However, DQ can also be performed by the user, by confirming through review and testing that the equipment meets the written acquisition specification. If the relevant document or manuals of machinery/equipment are provided by vendors, the later 3Q needs to be thoroughly performed by the users who work in an industrial regulatory environment. Otherwise, the process of IQ, OQ and PQ is the task of validation. The typical example of such a case could be the loss or absence of vendor's documentation for legacy equipment or do-it-yourself (DIY) assemblies (e.g., cars, computers etc.) and, therefore, users should endeavour to acquire DQ document beforehand. Each template of DQ, IQ, OQ and PQ usually can be found on the internet respectively, whereas the DIY qualifications of machinery/equipment can be assisted either by the vendor's training course materials and tutorials, or by the published guidance books, such as *step-by-step* series if the acquisition of machinery/equipment is not bundled with on- site qualification services. This kind of the DIY approach is also applicable to the qualifications of software, computer operating systems and a manufacturing process. The most important and critical task as the last step of the activity is to generating and archiving machinery/equipment qualification reports for auditing purposes, if regulatory compliances are mandatory.

Qualification of machinery/equipment is venue dependent, in particular items that are shock sensitive and require balancing or calibration, and re-qualification needs to be conducted once the objects are relocated. The full scales of some equipment qualifications are even time dependent as consumables are used up (i.e. filters) or springs stretch out, requiring recalibration, and hence re-certification is necessary when a specified due time lapse Re-qualification of machinery/equipment should also be conducted when replacement of parts, or coupling with another device, or installing a new application software and restructuring of the computer which affects especially the pre-settings, such as on BIOS, registry, disk drive partition table, dynamically-linked (shared) libraries, or an ini file etc., have been necessary. In such a situation, the specifications of the parts/devices/software and restructuring proposals should be appended to the qualification document whether the parts/devices/software are genuine or not.

Torres and Hyman have discussed the suitability of non-genuine parts for clinical use and provided guidelines for equipment users to select appropriate substitutes which are capable to avoid adverse effects. In the case when genuine parts/devices/software are demanded by some of regulatory requirements, then re-qualification does not need to be conducted on the non-genuine assemblies. Instead, the asset has to be recycled for non-regulatory purposes.

When machinery/equipment qualification is conducted by a standard endorsed third party such as by an ISO standard accredited company for a particular division, the process is called certification. Currently, the coverage of ISO/IEC 15408 certification by an ISO/IEC 27001 accredited organization is limited; the scheme requires a fair amount of efforts to get popularized.

1. **System Testing**

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called *assemblages*) or between any of the *assemblages* and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing tests not only the design, but also the behavior and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification

Types of tests to include in system testing

The following examples are different types of testing that should be considered during System testing:

* Graphical user interface testing
* Usability testing
* Software performance testing
* Compatibility testing
* Exception handling
* Load testing
* Volume testing
* Stress testing
* Security testing
* Scalability testing
* Sanity testing
* Smoke testing
* Exploratory testing
* Ad hoc testing
* Regression testing
* Installation testing
* Maintenance testing Recovery testing and failover testing.
* Accessibility testing, including compliance with:
  + Americans with Disabilities Act of 1990
  + Section 508 Amendment to the Rehabilitation Act of 1973
  + Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C)

Although different testing organizations may prescribe different tests as part of System testing, this list serves as a general framework or foundation to begin with.

1. **Structure Testing**

It is concerned with exercising the internal logic of a program and traversing particular execution paths.

1. **Output Testing**

* Output of test cases compared with the expected results created during design of test cases.
* Asking the user about the format required by them tests the output generated or displayed by the system under consideration.
* Here, the output format is considered into two was, one is on screen and another one is printed format.
* The output on the screen is found to be correct as the format was designed in the system design phase according to user needs.
* The output comes out as the specified requirements as the user’s hard copy.

1. **User Acceptance Testing**

* Final Stage, before handling over to the customer which is usually carried out by the customer where the test cases are executed with actual data.
* The system under consideration is tested for user acceptance and constantly keeping touch with the prospective system user at the time of developing and making changes whenever required.
* It involves planning and execution of various types of test in order to demonstrate that the implemented software system satisfies the requirements stated in the requirement document.

1. **Validation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TEST  CASE  NAME | DESCRIPTION | steps | ACTION TO BE TAKEN (DESIGN STEPS) | EXPECTED  (DESIGN  STEP) | TEST EXECUTION  RESULTS |
| 1.Ensuring the correctness of the Excel workbook | Objective:  An Excel sheet should be available, with the flexibility to add any number of rows. | Step 1 | Excel sheet should be available | The excel sheet is available | Pass |
|  |  | Step 2 | Excel sheet is created based on the template | The excel sheet should be based on the template | Pass |
|  |  | Step 3 | Changed the name of the excel sheet | Should not change any modifications of the name of excel sheet | Fail |
|  |  | Step 4 | Added 1000 or above records | Can add any number of records | Fail |
| 2.Comprehensive Sentiment Analysis Tests | Objective:  Conduct thorough testing of the BERT model for sentiment analysis. | Step 1 | Input a strongly positive sentiment text. | Model should confidently predict a strong positive sentiment. | pass |
|  |  | Step 2 | Input a mix of positive and negative sentiments | Model should accurately identify and handle mixed sentiments | pass |
|  |  | Step 3 | Conduct stress test with a large dataset. | Model should handle a large dataset efficiently without significant performance degradation | pass |
|  |  | Step 4 | Test the model's ability to generalize across different domains | Model should perform consistently across various types of content | fail |
| 3. Diverse Sentiment Analysis Scenarios | Objective: Assess BERT model across various challenging scenarios | Step 1 | Input text with highly subjective language. | Model should discern and analyze sentiment despite subjectivity | pass |
|  |  | Step 2 | Test the model's response to colloquial expressions | Model should accurately interpret sentiment in colloquial language | pass |
|  |  | Step 3 | Input text with cultural or regional sentiment nuances | Model should be sensitive to cultural variations in sentiment expression | pass |
|  |  | Step 4 | Test the model's performance on mixed-language sentiment analysis | Model should handle texts with mixed languages while accurately predicting sentiment | pass |

**CHAPTER-7**

**RESULTS**

**7.1 SCREENSHORTS**

**sentiment analysis on movie reviews**

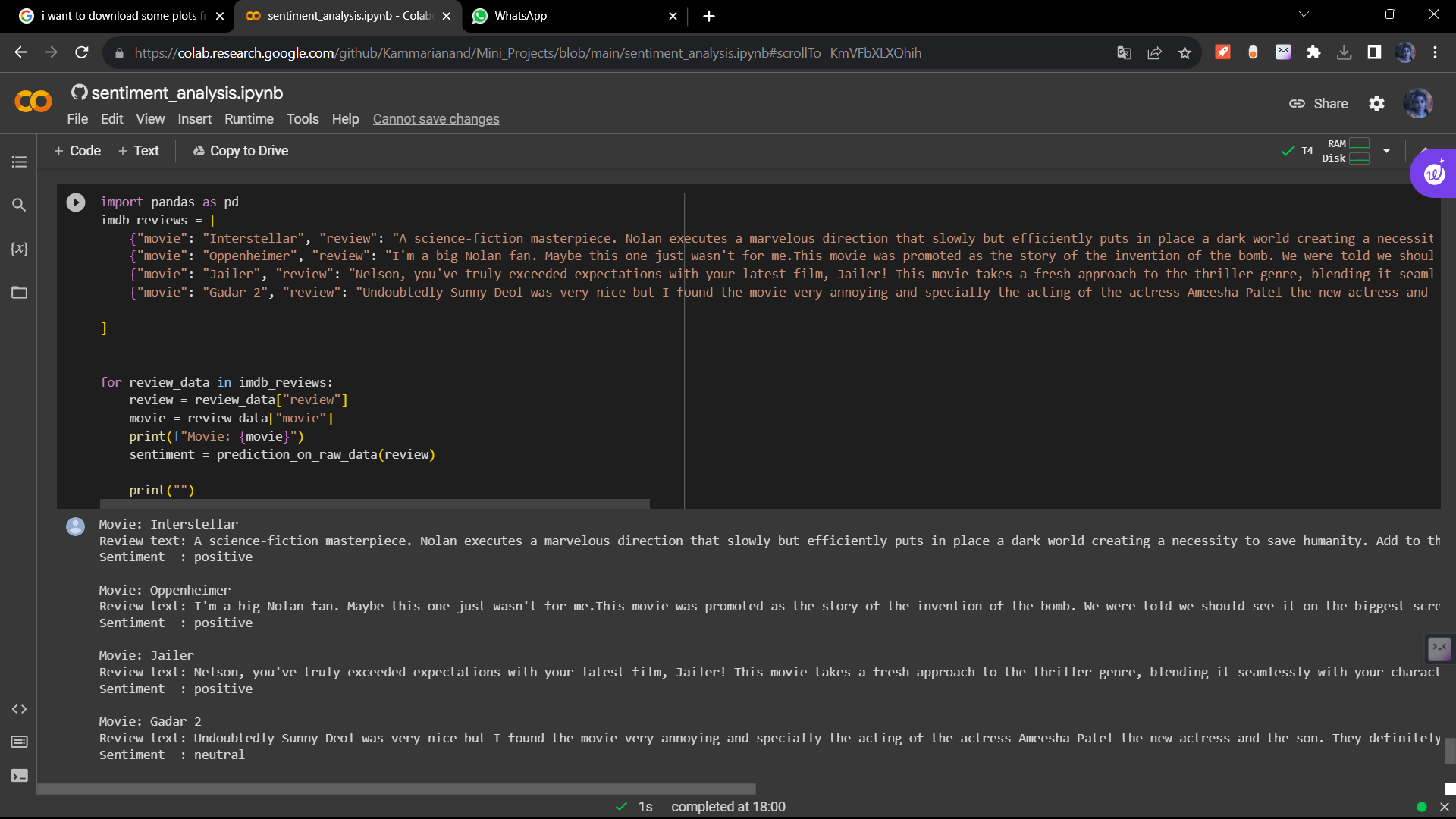
****

Fig 7.1.1 prediction on movie reviews

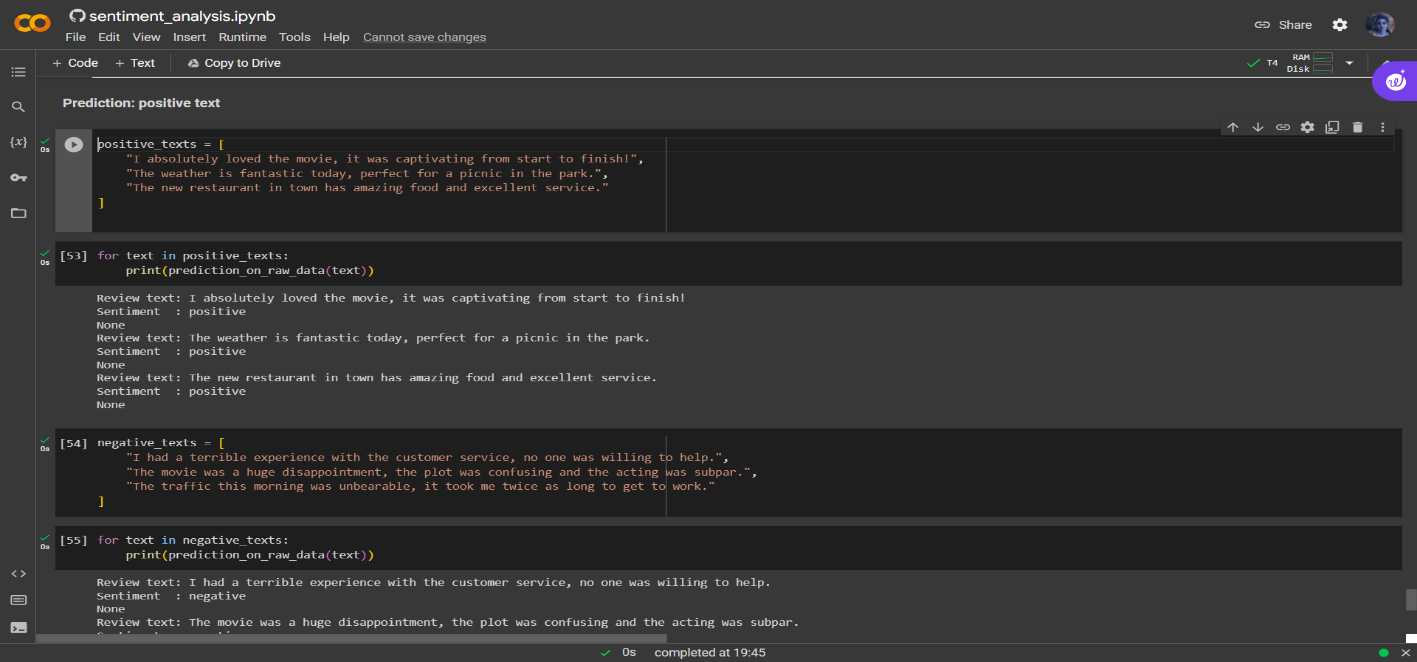
**2.Prediction on positive text**

Fig 7.1.2 prediction on positive sentence

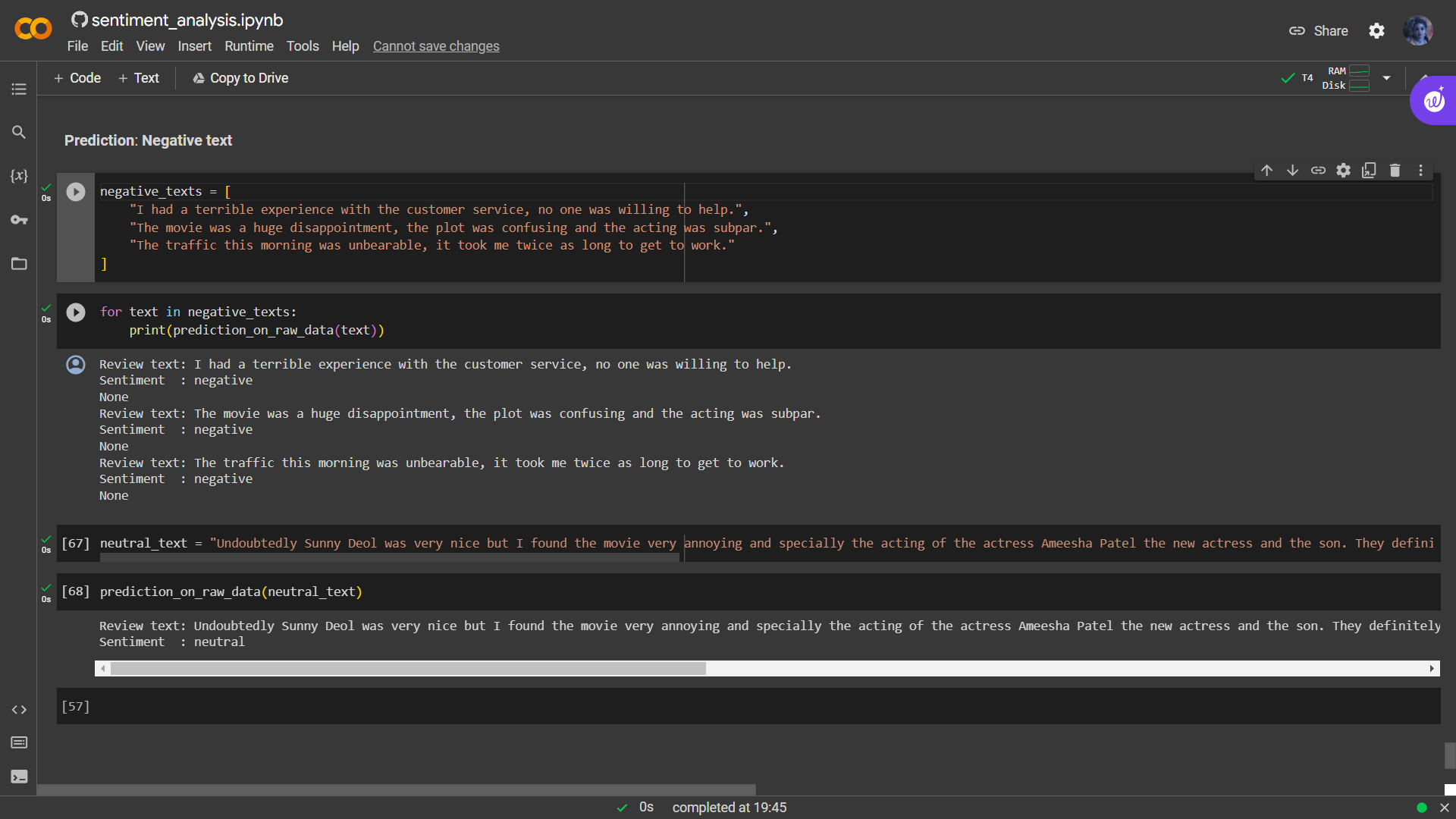
**3.Prediction on negative text **

Fig 7.1.3 prediction on negative sentence

**4.Prediction on neutral text**

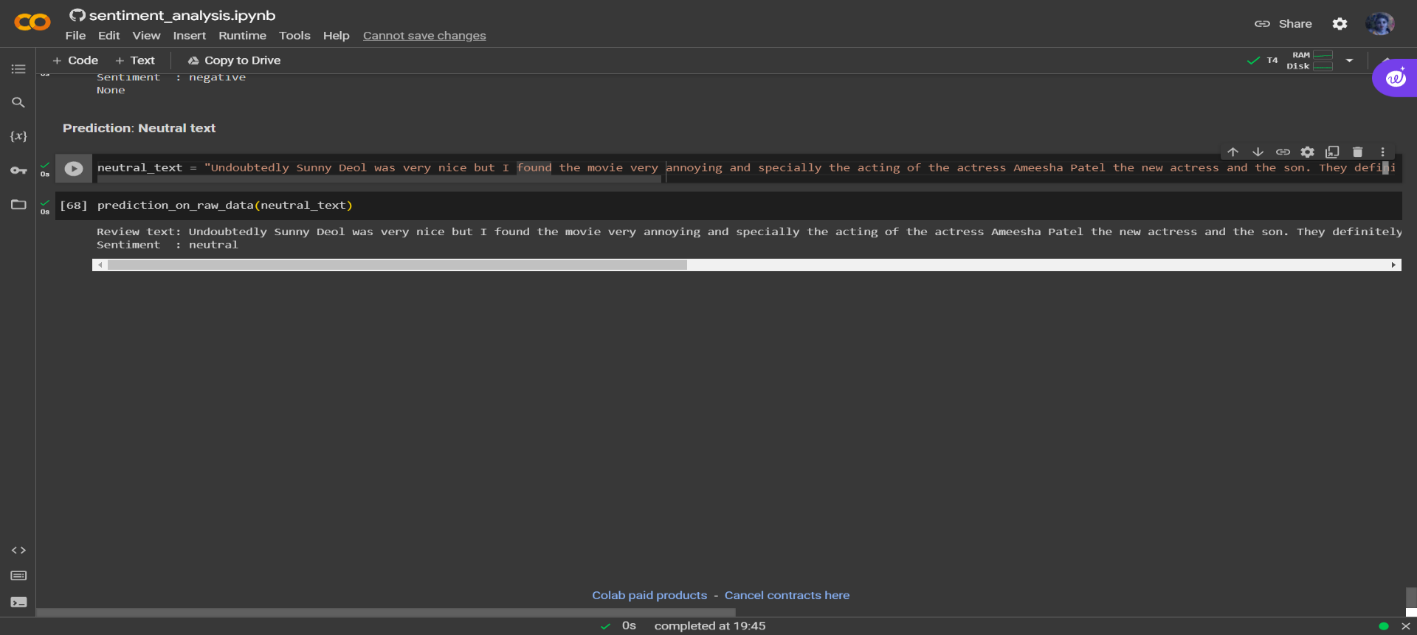


Fig 7.1.4 prediction on neutral sentence

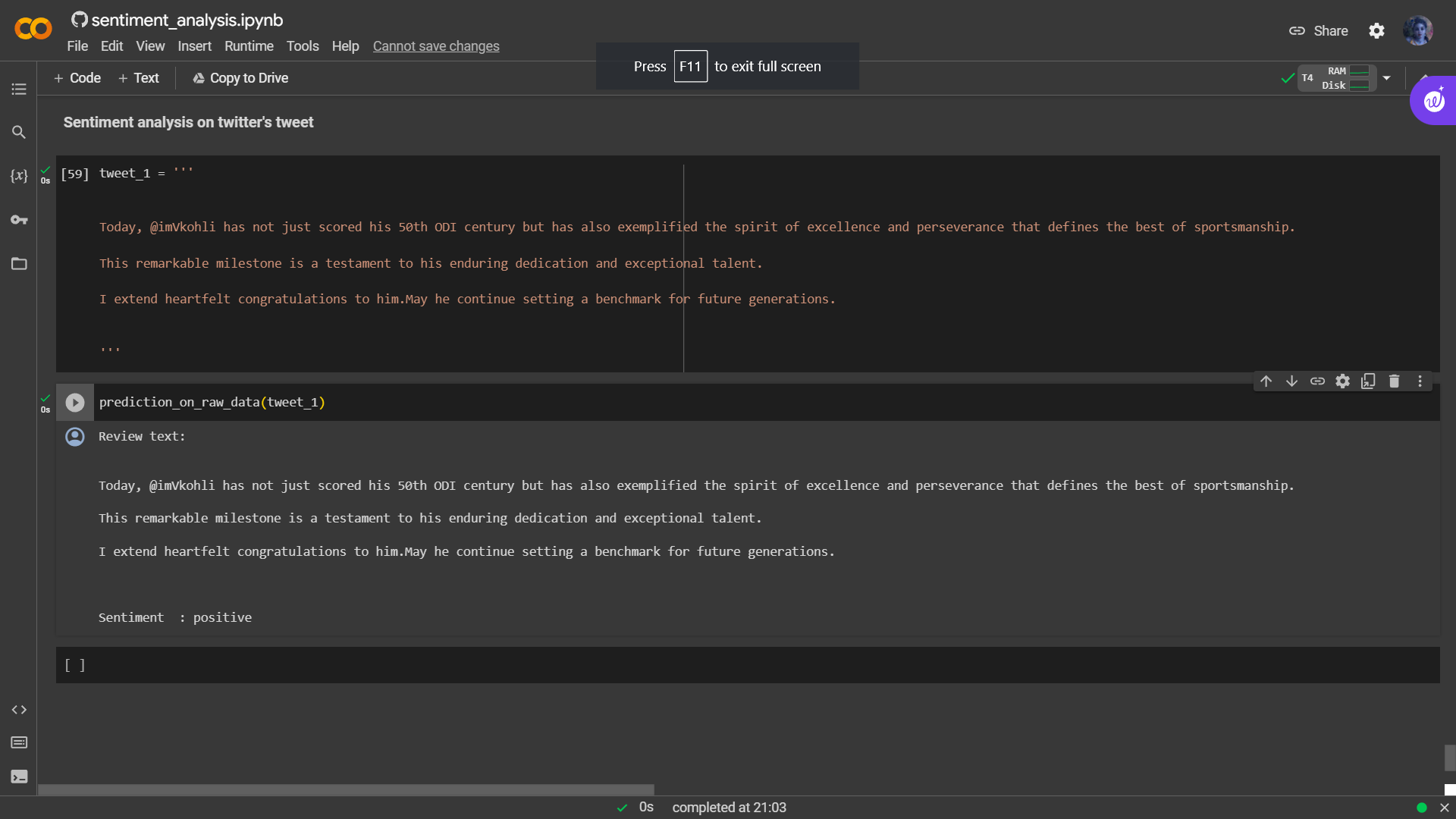
**5.Sentiment analysis on tweet**

Fig 7.1.5 analysis on tweets

**7.2 Plots**

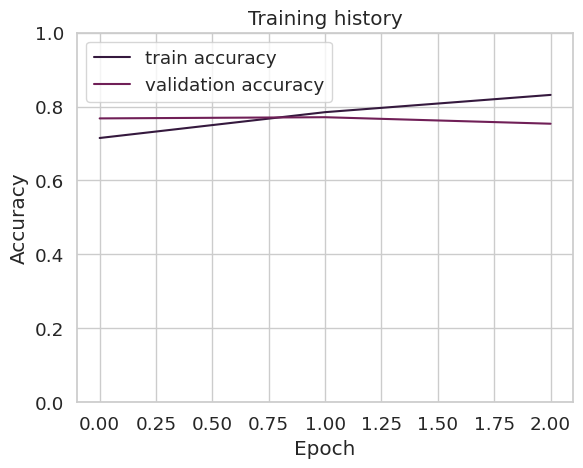
****

Fig 7.2.1 Accuracy

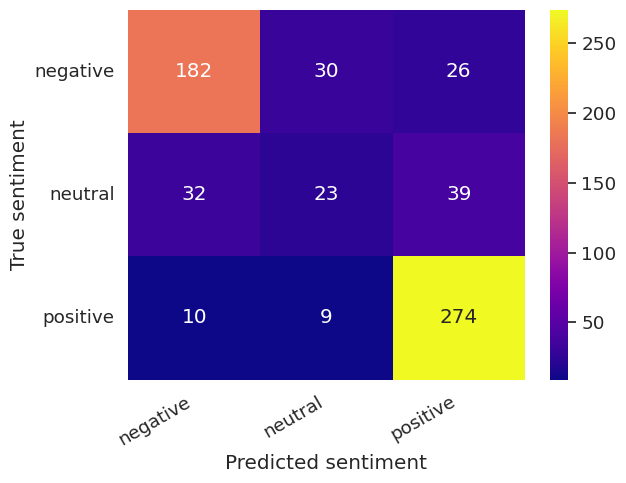


Fig 7.2.2 Heat Map

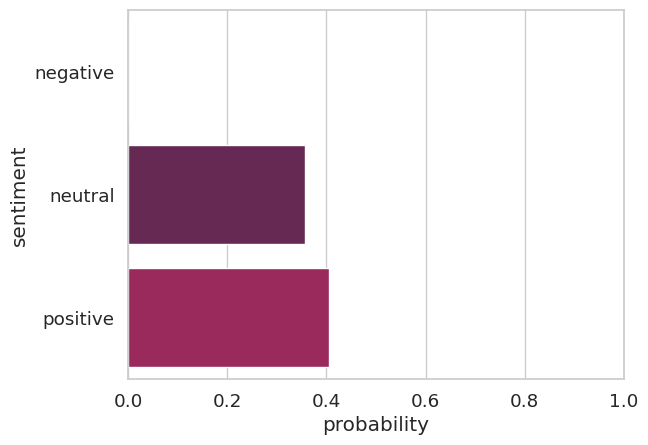


Fig 7.2.3 Bar plot

**CHAPTER-8**

**CONCLUSION**

# 8.1 CONCLUSION

In summary, our project, 'Sentiment Analysis on Text Using BERT Neural Network,' 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.

**CHAPTER-9**

**FUTURE ENCHANTMENT**

# 9.1FUTURE ENHANCEMENT

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

1. **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**.**

1. **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**.**

1. **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.

1. **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 :**

* Rishi Meridian, “Analysis of Google Play Store Data set and predict the popularity of an app on Google Play Store”, Texas a&M University College Station, Texas, June-2020. Jong, J. (2011). Predicting rating with sentiment analysis
* William B Dolan and Chris Brockett. 2005. Automati cally constructing a corpus of sentential paraphrases. In Proceedings of the Third International Workshop on Paraphrasing (IWP2005).
* Harman, M., Jiao, Y., and Zhang, Y. (2012). App store mining and analysis: Mrs for app Stores. In 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), Pages 108– 111.
* [Https://developer.mozilla.org/en-US/docs/Learn/Serverside/Django/Introduction](https://developer.mozilla.org/en-US/docs/Learn/Serverside/Django/Introduction)
* In the Google play store. Empirical Software Engineering, 21(3), 1346-1370. Islam, M. R. (2014). Numeric rating of apps on Google play store by sentiment Analysis on user reviews. In 2014 International Conference on Electrical Engineering And Information Communication Technology, pages 1–4.
* John Blitzer, Ryan McDonald, and Fernando Pereira. 2006. Domain adaptation with structural correspon dence learning. In Proceedings of the 2006 confer ence on empirical methods in natural language pro cessing, pages 120–128. Association for Computa tional Linguistic.
* Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large anno tated corpus for learning natural language inferencE.
* J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09.
* William Fedus, Ian Goodfellow, and Andrew M Dai. 2018. Maskgan: Better text generation via filling in the . arXiv preprint arXiv:1801.07736.
* Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning.