**SENTIMENTAL ANALYSIS FROM TEXT**

**USING DEEP LEARNING**

**This project report is submitted in partial fulfilment of the requirement**

**for the award of the degree**

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**



Submitted by

**POLIREDDY SUSHMA**

**Reg.no: 208297601042**

Under the esteemed guidance of

**Mrs. JAYANTHI HARINI**

**Assistant Professor**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ADIKAVI NANNAYA UNIVERSITY

RAJAMAHENDRAVARAM-533296

2020-2024

**ADIKAVI NANNAYA UNIVERSITY, RAJAMAHENDRAVARAM**

**UNIVERSITY COLLEGE OF ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**



This is to certify that this project report entitled, “**SENTIMENTAL ANALYSIS FROM TEXT USING DEEP LEARNING**” is a bonafide work of **POLIREDDY SUSHMA**, Reg.No: **208297601042** submitted in a partial fulfilment of the requirements for the award of Degree of BTech (CSE) during the period 2020-2024. This work carried out by her under my supervision and guidance and submitted to Department of Computer Science and Engineering, Adikavi Nannaya University.

**INTERNAL GUIDE HEAD OF THE DEPARTMENT**

**EXTERNAL EXAMINER**

**DECLARATION**

I **POLIREDDY SUSHMA,** reg.no:**208297601042**, hereby declare that the project report entitled Sentimental Analysis from Text using Deep Learning done by me under the guidance of **Mrs. JAYANTHI HARINI, Assistant Professor, Adikavi Nannaya University**, is submitted for the partial fulfilment of requirement of the award of the degree, Bachelor of Technology in Computer Science and Engineering in the academic year 2020-2024.

Signature of the student

**ACKNOWLEDGEMENT**

I feel fortunate to pursue my **Bachelor of Technologies** degree in the **Department of Computer Science and Engineering**, **University College of Engineering, Adikavi Nannaya University**. It provided all the facilities in the area of Computer Science and Engineering. It’s a genuine pleasure to express my deep sense of thanks and gratitude to my mentor and project guide **Mrs. Jayanthi Harini, Assistant Professor, Department of Computer Science and Engineering, Adikavi Nannaya University** for her excellent guidance right from the selection of the project and her valuable suggestions throughout the project work. Her constant and timely counsel has caused me to succeed in completing this project in college.

I profusely thank **Dr.P. Venkateswara Rao**, Principal, University College of Engineering for all the encouragement and support.

I also thank **Dr.B. Kezia Rani**, Head of the Department of Computer Science and Engineering for the guidance.

I also thank **Dr.D. Latha**, Project Coordinator, Department of Computer Science and Engineering for the guidance.

A great deal of thanks goes to Review Committee Members and the entire Faculty of the Department of Computer Science and Engineering, University College of Engineering, Adikavi Nannaya University, Rajahmahendravaram for their excellent supervision and valuable suggestions.

POLIREDDY SUSHMA

208297601042

**ABSTRACT**

Sentimental analysis is also known as opinion mining is the computational task of determining the sentiment whether it is positive, negative or neutral conveyed within a piece of text. As the internet and social media platforms continue to serve as large number of mediums for communication, the analysis of user-generated content has gained supreme importance. Individuals, businesses and organizations seek to extract valuable insights from vast datasets comprising customer reviews, social media posts, and textual feedback. Traditional sentiment analysis approaches often fall short in capturing the complexity of human language and context. Deep learning, a subset of machine learning provides a powerful and intense tool for natural language understanding and sentiment classification.

Sentimental analysis is a crucial component of natural language processing which plays a pivotal role in understanding the emotional tone expressed in textual data. In this project, an approach is proposed for sentiment analysis by using a deep learning model of Long Short-Term Memory (LSTM) networks and word embeddings like Bidirectional Encoder Representations from Transformers (BERT). LSTM models are effective in capturing sequential patterns in text, while BERT provides contextualized word embeddings. The project utilizes a dataset of text samples labelled with sentiment categories to train and evaluate deep learning model.

**INDEX**

|  |  |
| --- | --- |
| **CONTENTS** | **PAGE NO** |
| **1. INTRODUCTION**  1.1 PROBLEM STATEMENT  1.2 LITERATURE SURVEY  1.3 EXISTING SYSTEM  1.4 PROPOSED SYSTEM  1.5 SCOPE OF THE PROJECT | 1  3  3  5  6  6 |
| **2. SYSTEM ANALYSIS**  2.1 HARDWARE REQUIREMENTS  2.2 SOFTWARE REQUIREMENTS  2.3 FUNCTIONAL REQUIREMENTS  2.4 NON-FUNCTIONAL REQUIREMENTS | 8  8  8  11  11 |
| **3. SYSTEM DESIGN**  3.1 SYSTEM ARCHITECTURE  3.2 METHODOLOGY  3.3 UML DIAGRAMS | 13  13  14  18 |
| **4. IMPLEMENTATION**  4.1 DATASET  4.2 CODING  4.3 MODEL DEPLOYMENT | 22  26  26  36 |
| **5. OUTPUT SCREENS** | 44 |
| **6. CONCLUSION AND FUTURE SCOPE** | 52 |
| **7. REFERENCES** | 53 |

|  |  |  |
| --- | --- | --- |
|  | **FIGURE NAME** | **PAGE NO** |
|  | 1.Showing system architecture | 13 |
|  | 2.The repeating module in a standard RNN | 15 |
|  | 3.The repeating module in an LSTM | 15 |
|  | 4.The working of Bert | 17 |
|  | 5.The Architecture of Bert and Lstm | 17 |
|  | 6.Use case diagram | 19 |
|  | 7.Class diagram | 20 |
|  | 8.Sequence diagram | 20 |
|  | 9.Activity diagram | 21 |
|  | 10.The Decision Tree diagram | 23 |
|  | 11.The output for user interface | 44 |
|  | 12.Home page | 44 |
|  | 13.About page | 45 |
|  | 14.Load page for uploading file | 45 |
|  | 15.Load page displaying Data Loaded Successfully | 46 |
|  | 16.Preprocess page | 46 |
|  | 17.Preprocess page displaying Data pre-processed successfully | 47 |
|  | 18.Model page for selecting the model | 47 |
|  | 19.Model page displaying accuracy of the selected model | 48 |
|  | 20.Prediction page for entering text data of example1 | 48 |
|  | 21.Prediction page displaying output for example1 | 49 |
|  |  |  |

**LIST OF FIGURES**

|  |  |
| --- | --- |
| 22.Prediction page for entering text data of example2 | 49 |
| 23. Prediction page displaying output for example2 | 50 |
| 24.Prediction page for entering text data of example3 | 50 |
| 25.Prediction page displaying output for example3 | 51 |

# CHAPTER-1

**INTRODUCTION**

Sentimental analysis is also known as opinion mining, is a natural language processing (NLP) technique used to determine the sentiment expressed in a piece of text. It involves analysing and categorizing the subjective information conveyed in the text as positive, negative, or neutral. Sentiment analysis algorithms can be applied to various types of textual data, including social media posts, product reviews, customer feedback, news articles, and more.

The primary goal of sentiment analysis is to extract valuable insights from textual data, allowing businesses, organizations, and individuals to understand public opinion, customer sentiment, and trends. It has numerous applications across different industries, including market research, brand monitoring, customer service, reputation management, and social media analytics. Sentimental analysis enables stakeholders to make data-driven decisions, improve products and services, enhance customer satisfaction, and respond effectively to public opinion.

Sentimental analysis that involves determining the sentiment or emotional tone expressed in textual data. With the exponential growth of textual data on the internet, sentimental analysis has become increasingly important for understanding public opinion, customer feedback, and social media sentiment. Deep learning techniques have emerged as powerful tools for sentiment analysis due to their ability to learn complex patterns and representations directly from raw data. In this project, we aim to explore the application of deep learning, specifically Long Short-Term Memory (LSTM) networks and Bidirectional Encoder Representations from Transformers (BERT), for sentiment analysis tasks.

Sentimental analysis using LSTM (Long Short-Term Memory) and BERT (Bidirectional Encoder Representations from Transformers) holds significant importance due to their capabilities in capturing complex patterns and nuances in text data, leading to more accurate sentiment classification.

Here are some key reasons why sentimental analysis using LSTM and BERT is important:

**1. Capturing Long-Term Dependencies:** LSTM networks are well-suited for capturing long-term dependencies in sequential data, making them effective for sentiment analysis tasks where the context of words and phrases over extended sequences is crucial for understanding sentiment.

**2. Handling Sequential Data:** Text data is inherently sequential, and LSTM networks excel in processing sequences, making them suitable for sentiment analysis tasks where the order of words matters in determining sentiment.

**3. Contextual Understanding:** BERT, a state-of-the-art pre-trained language model, provides deep contextual understanding of text by considering bidirectional contexts. This allows BERT to capture complex linguistic nuances and semantic relationships, enhancing the accuracy of sentiment analysis.

**4. Transfer Learning:** BERT can be fine-tuned on domain-specific sentiment analysis tasks with relatively small amounts of labelled data, leveraging its pre-trained knowledge from large corpora. This facilitates faster model training and deployment in real-world applications.

**5. Handling Ambiguity and sarcasm:** Sentiment analysis often faces challenges with ambiguous language and sarcasm. LSTM and BERT models, with their ability to capture contextual information and semantic nuances, can effectively handle such complexities, leading to more accurate sentiment predictions.

**6. Improved Performance:** By leveraging the strengths of LSTM and BERT, sentiment analysis models can achieve higher accuracy, precision, recall, and F1-score compared to traditional methods. This translates to better insights and decision-making for businesses relying on sentiment analysis for customer feedback, brand monitoring, and market research.

Sentimental analysis using LSTM and BERT enables organizations to extract deeper insights from text data, leading to enhanced understanding of customer sentiment, market trends, and brand perception, ultimately driving better business outcomes.

Top of Form

**1.1** **PROBLEM STATEMENT**

In the era of massive textual data generated across digital platforms, existing sentiment analysis approaches fall short in providing nuanced and real-time insights into the emotional tone of language. The challenge lies in creating a system that not only automates sentiment classification but also comprehends the complex contextual nuances, varying writing styles, and linguistic expressions present in diverse datasets. Despite the advancements in sentiment analysis techniques, challenges persist in accurately capturing the nuanced sentiments expressed in diverse types of text data, including social media posts, customer reviews, news articles, and online discussions. The project aims to address this by developing a deep learning model that uses recurrent architecture to achieve an efficient sentimental analysis solution. The problem addressed by this project is to develop effective sentiment analysis models using deep learning techniques, specifically Long Short-Term Memory (LSTM) networks and Bidirectional Encoder Representations from Transformers (BERT), to overcome the limitations of existing sentiment analysis approaches. The goal is to create models capable of accurately analysing and categorizing sentiments expressed in various types of textual data, including short informal texts from social media platforms and longer, more structured texts from news articles or customer reviews.

**1.2 LITERATURE SURVEY**

**[1] P. S. Sreeja and G. S. Mahalakshmi:** Poem is a type of literature designed to convey, ideas, emotions, and experiences in a brilliant way. In this article, we discuss the automatic emotion recognition of poems written in English. This is a pioneering approach in emotion recognition from poems. Emotions from the poems, classified into nine emotions, based on 'Navarasa' under 'Rasa Theory' which is described in 'Natyashastra' written by 'Bharatha Muni'. The nine basic emotions such as Love, Sadness, Anger, Hatred, Fear, Surprise, Courage and Peace, classified as “Navarasa ". As to our knowledge, we are not familiar about a text corpus of poems based on nine emotions, we have manually created an emotion tagged corpus from poems in English. The corpus created is from an exhaustive collection of poems of Indian poets from the period 1850-1950. The poems are mined from the web, and we applied ten cross fold Naïve Bayes classifier to recognize the emotion of a poem by maximum likelihood probability.

**Summary:** The nine basic emotions such as Love, Sadness, Anger, Hatred, Fear, Surprise, Courage, Joy, and Peace, classified as “Navarasa ". As to our knowledge, we are not familiar about a text corpus of poems based on nine emotions, we have manually created an emotion tagged corpus from poems in English. The corpus created is from an exhaustive collection of poems of Indian poets from the period 1850-1950. The poems are mined from the web, and we applied ten cross fold Naïve Bayes classifier to recognize the emotion of a poem by maximum likelihood probability.

**[2] J. Kaur and J. R. Saini:** Literature of country represents the prosperity of that country. India, being a multilingual country, is having a rich heritage and literature. In order to retrieve literature pieces easily, it must be classified. In this research work, vocabulary-content based classification of Punjabi poetry is done. 4 different poetry categories are populated with 240 poetries (with 60 poems in each category). These 240 poetry documents are passed through typical NLP text classification phases like Sentence Splitting, Tokenization and Bag-of-Words (BOW) representation, finally yielding to their Vector Space Model (VSM) representation. Total 9867 unique words extracted from last step are used for building the different machine learning models. For the first time in research community, 10 different machine learning algorithms are trained and tested for any Indian language, using weka, with an aim to find the most suitable algorithm.

**Summary:** For the first time in research community, 10 different machine learning algorithms are trained and tested for any Indian language, using weka, with an aim to find the most suitable algorithm. Results for Punjabi poetry classification revealed that 4 machine learning algorithms namely, Hyperpipes (HP), K- nearest neighbor (KNN), Naive Bayes (NB) and Support Vector Machine (SVM) with an accuracy of 50.63 %, 52.92 %, 52.75 % and 58.79 % respectively, outperformed all other machine learning algorithms under the test.

**[3] G. Mohanty and P. Mishra:** Resource poor languages, like Odia, inherently lack the necessary resources and tools for the task of sentiment analysis to give promising results. With more user-generated raw data readily available today, it is of prime importance to have annotated corpora from various domains. This paper is a first attempt towards building an annotated corpus of Odia poetry with sentiment labels. The annotated corpus is further used for sentiment classification using machine learning techniques in order to establish a baseline. Stylistic variations and structural differences between poetic and non-poetic texts make the task of sentiment classification challenging for the former. Using the annotated corpus of poems, we obtained comparable accuracies across various classification models. Linear-SVM outperformed other classifiers with a macro F1- Score of 0.68. The annotated corpus contains a total of 730 Odia Poems of various genres with a vocabulary of more than 23k words. Fleiss Kappa score of 0.83 was obtained which corresponds to near perfect agreement among the annotators.

**Summary:** This paper is a first attempt towards building an annotated corpus of Odia poetry with sentiment labels. The annotated corpus is further used for sentiment classification using machine learning techniques in order to establish a baseline. Stylistic variations and structural differences between poetic and non-poetic texts make the task of sentiment classification challenging for the former. Using the annotated corpus of poems, we obtained comparable accuracies across various classification models. Linear-SVM outperformed other classifiers with a macro F1- Score of 0.68. The annotated corpus contains a total of 730 Odia Poems of various genres with a vocabulary of more than 23k words.

* 1. **EXISTING SYSTEM**

Emotions from the text are detected manually by humans by analyzing the content present in the text. This consumes time for analyzing the data.

**Traditional Machine Learning Approaches:**

Machine learning algorithms such as Decision Tree, Random Forest and Logistic regression. These models often use derived features and may struggle with capturing complex linguistic patterns.

Challenges in the existing system include:

1. Difficulty in capturing contextual information and understanding the semantics of the text.
2. Difficulty in Handling Long-Range Dependencies
3. Fixed-Length Feature Vectors
4. Difficulty in Capturing Subtle Sentiments
5. Limited Generalization Across Writing Styles
6. Low accuracy
   1. **PROPOSED SYSTEM**

The proposed system introduces a deep learning architecture that uses Long Short-Term Memory (LSTM) networks and word embeddings BERT (BERT stands for Bidirectional Encoder Representations from Transformers). This architecture aims to capture both local patterns and long-range dependencies in textual data, providing a more nuanced understanding of sentiment. The proposed system is designed with scalability in mind, allowing the system to perform well across different linguistic styles and domains.

Benefits of the Proposed System:

1. Enhanced Accuracy: Deep learning models like LSTM and BERT have shown superior performance in capturing complex linguistic patterns and semantic relationships, leading to higher accuracy in sentiment analysis.
2. Improved Robustness: By leveraging deep learning techniques, the proposed system can handle noisy and informal text data more effectively, leading to improved robustness and generalization capabilities.
3. Contextual Understanding: BERT-based models can capture contextual information and semantics more accurately compared to traditional methods, resulting in better sentiment analysis results, especially for ambiguous or sarcastic text.
4. Scalability: Deep learning models can scale well with large volumes of text data, making them suitable for processing and analyzing massive datasets commonly found on social media platforms and online forums.

The proposed system aims to address the limitations of existing sentiment analysis approaches by leveraging the power of deep learning techniques like LSTM and BERT, ultimately leading to more accurate, robust, and context-aware sentiment analysis from text.

**1.5 SCOPE OF THE PROJECT**

The project on sentimental analysis using LSTM and BERT has a wide scope and potential applications across various domains. Here are some aspects that highlight the scope of the project:

**1. Industry Applications:** Sentiment analysis is valuable for industries such as marketing, customer service, finance, and healthcare. Implementing LSTM and BERT models for sentiment analysis can provide actionable insights for businesses to improve customer satisfaction, identify market trends, manage brand reputation, and make informed decisions.

**2. Social Media Monitoring:** With the proliferation of social media platforms, there's a growing need to analyze user-generated content to understand public opinion, sentiment trends, and emerging issues. LSTM and BERT-based sentiment analysis can help monitor social media conversations in real-time, detect sentiment shifts, and identify influential topics.

**3. Customer Feedback Analysis:** Many businesses collect feedback from customers through surveys, reviews, and comments. LSTM and BERT models can automate the analysis of large volumes of textual feedback, categorize sentiments, and extract valuable insights to enhance product quality, service offerings, and overall customer experience.

**4. Brand Reputation Management:** Maintaining a positive brand image is crucial for

businesses. Sentiment analysis using LSTM and BERT can monitor online mentions, news articles, and reviews to assess brand sentiment, detect potential PR crises, and formulate proactive strategies to manage reputation effectively.

**5. Market Research:** LSTM and BERT-based sentiment analysis can aid market researchers in analyzing consumer sentiment towards products, competitors, and industry trends. By understanding market sentiment, businesses can identify new opportunities, assess market demand, and refine marketing strategies accordingly.

**6. Customization and Personalization**: Sentiment analysis models can be customized and fine-tuned for specific domains, languages, or target audiences. This allows organizations to tailor sentiment analysis solutions to their unique requirements and achieve higher accuracy in sentiment classification.

# CHAPTER-2

**SYSTEM ANALYSIS**

**2.1 HARDWARE REQUIREMENTS**

The Hardware consists of the physical components of the computer that input storage processing control, output devices.

The kind of hardware used in the project is

**1. GPU**: It is recommended to use a GPU with at least 8 GB of VRAM for training deep learning model. This will provide sufficient memory to handle many parameters in the model and the large amount of data used for training.

**2. CPU:** An Intel Core i5 processor

**3. OS:** Windows 11

**4. RAM:** 8 GB RAM

**5. Hard Disk:** 256 GB

**2.2 SOFTWARE REQUIREMENTS**

Software is a set of programs to do a particular task. Software is an essential requirement of computer systems. The system requirements or software requirements is a listing of what software programs are required to operate the program properly.

The kind of software used in the project is

1. Python: version 3.10
2. TensorFlow: an open-source machine learning framework used for building and training machine learning models
3. Keras: a high-level neural networks API, written in Python and capable of running on top of TensorFlow
4. Jupyter Notebook: an open-source application that allows you to create and share documents that contain live code, equations, visualizations and narrative text
5. Visual Studio Code: to execute the code
6. NumPy: a Python library used for working with arrays
7. Pandas: a Python library used for data manipulation and analysis
8. Matplotlib: a Python library used for creating static, animated, and interactive visualizations in Python
9. Sci-kit learn: to implement statistical modelling

### **SOFTWARE REQUIREMENTS DESCRIPTION**

**Jupyter Notebook**

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results.

The Jupyter notebook combines two components:

1. **A web application**: A browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.
2. **Notebook documents**: A representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects.

**Visual Studio Code**

VS Code provides a powerful IDE with features like code highlighting, IntelliSense, and debugging tools, which enhance productivity and make development easier.

**Python**

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. In July 2018, Van Rossum stepped down as the leader in the language community.

**Features**

1. Python uses dynamic typing, and a combination of reference counting and a cycle- detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

2. Python's design offers some support for functional programming. It has filter(), map(), and reduce() functions; list comprehensions, dictionaries, and sets; and generator expressions.

**Libraries**

1. NumPy: NumPy is a Python library used for working with arrays. It provides support for mathematical operations on large, multi-dimensional arrays and matrices.
2. Pandas: Pandas is a Python library used for data manipulation and analysis. It provides tools for working with structured data such as tabular data, time-series data, and matrix data.
3. Matplotlib: Matplotlib is a Python library used for creating static, animated, and interactive visualizations in Python. It provides tools for creating plots, charts, and other types of visualizations.
4. TensorFlow or Keras: TensorFlow is a popular deep learning framework, and Keras is a high-level API that can run on top of TensorFlow. These libraries provide tools for building and training deep learning models, including LSTM networks.
5. Sci-kit learn**:** Scikit-learn is a machine learning library in Python that provides simple and efficient tools for data mining and data analysis. It can be used for tasks like data preprocessing, feature extraction, and evaluation metrics for classification tasks
6. Natural Language Tool Kit: NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources, such as WordNet, along with a suite of text processing libraries for tasks like tokenization, stemming, and part-of-speech tagging.
7. Transformers**:** Transformers is a Python library for natural language processing (NLP) using deep learning techniques. It provides state-of-the-art pre-trained models for various NLP tasks, including sentiment analysis, named entity recognition, and text classification.
8. Ktrain**:** Ktrain is a lightweight wrapper for TensorFlow Keras that simplifies the process of training and deploying deep learning models, especially for tasks like text classification and sentiment analysis.

**2.3 FUNCTIONAL REQUIREMENTS**

**1. Text Input Handling**: The system should accept textual input for sentiment analysis from various sources, including user inputs, social media posts, reviews.

**2. Preprocessing:** Tokenization of text data. Removal of stop words, punctuation, and special characters.

**3. Word Embeddings:** Contextual embeddings BERT (Bidirectional Encoder Representations from Transformers).

**4. Deep Learning Model:** Long Short-Term Memory (LSTM) neural network.

**5. User Interface:** Develop a user interface that allows users to input text for sentiment analysis and receive the corresponding sentiment classification.

**6. Prediction Module:** Implement a prediction module that applies the trained model to new data and outputs the predicted sentiment class (positive, negative and neutral).

* 1. **NON-FUNCTIONAL REQUIREMENTS**

**1. Performance**: The system should be able to process large volumes of text data efficiently, including both training and inference phases. It should provide fast response times for analysing sentiment in real-time or near real-time.

**2. Scalability:** The system should be able to scale horizontally to handle increasing workloads and growing datasets. It should support distributed training and inference across multiple machines or devices.

**3. Accuracy:** The sentiment analysis model should achieve high accuracy in classifying the sentiment of text inputs. It should be able to generalize well to unseen data and handle different types of text inputs, including variations in language, style, and context.

**4. Robustness:** The system should be robust to noise, errors, and variations in input data. It should handle misspellings, grammatical errors, slang, and other linguistic phenomena commonly found in natural language text.

**5. Interpretability:** The sentiment analysis model should be interpretable, meaning that it should provide insights into how decisions are made and what features contribute to the classification of sentiment. This helps users understand and trust the model's predictions.

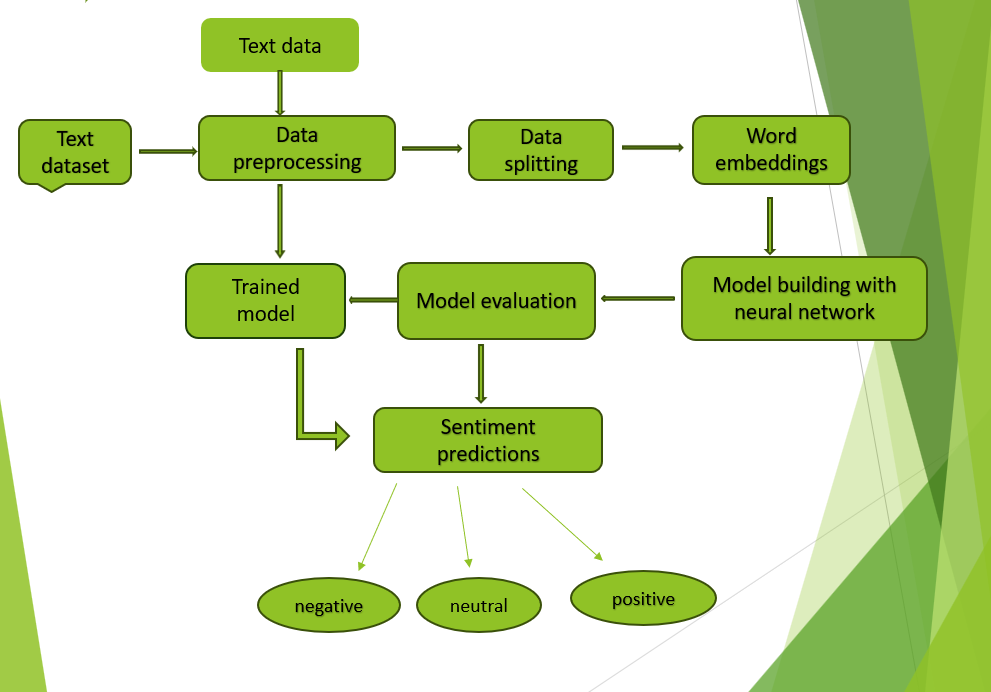
**6. Resource Efficiency:**  The system should use computational resources (e.g., memory, CPU, GPU) efficiently to minimize costs and maximize performance. It should optimize model size, memory usage, and energy consumption for deployment on different platforms and devices.

**7. Usability:** The system should be easy to use and understand for both developers and end-users. It should provide clear documentation, intuitive interfaces, and helpful error messages to support effective interaction and troubleshooting.

**CHAPTER-3**

**SYSTEM DESIGN**

**3.1 SYSTEM ARCHITECTURE**

****

**Figure1 Showing System Architecture**

The system architecture for a sentiment analysis project using LSTM and BERT may involve several components working together to process text data, train models, and perform sentiment analysis.

**1.** **Data Collection**: Text data is collected from various sources such as social media platforms, customer reviews, or text corpora. This data is ingested into the system for preprocessing and analysis.

**2.** **Data Preprocessing**: The raw text data undergoes preprocessing steps such as text cleaning, tokenization and removing stop words. This prepares the text data for feature extraction and model training.

**3. Data Splitting:** After completion of data preprocessing, the data is divided into train and test with 80% and 20% respectively**.**

**4. Word Embeddings:** BERT (Bidirectional Encoder Representations from Transformers) capture contextual information bidirectionally in a given text. It compares every single word with other words in the sentence.

**5. Model Training**: Two types of models are trained in parallel:

LSTM Model: A Long Short-Term Memory (LSTM) neural network is trained on the preprocessed text data to learn patterns and relationships in the text that correlate with sentiment.

BERT Model: A Bidirectional Encoder Representations from Transformers (BERT) model is fine-tuned on the sentiment analysis task using transfer learning, leveraging a pre-trained BERT model's knowledge.

**6. Evaluation**: The trained model is evaluated on a separate validation dataset to assess their performance in classifying sentiment accurately. Metrics such as accuracy, precision, recall, and F1-score are computed to evaluate model performance.

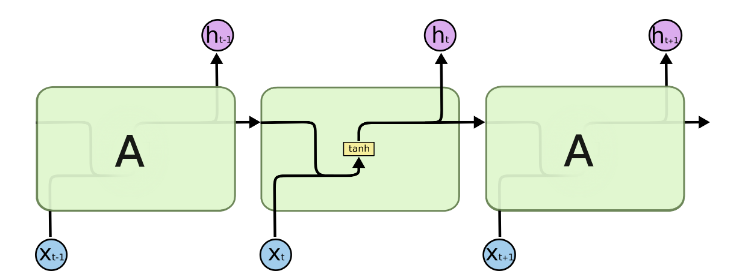
**3.2 METHODOLOGY**

**LSTM Networks**

Long Short -Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmid Huber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behaviour, not something they struggle to learn!

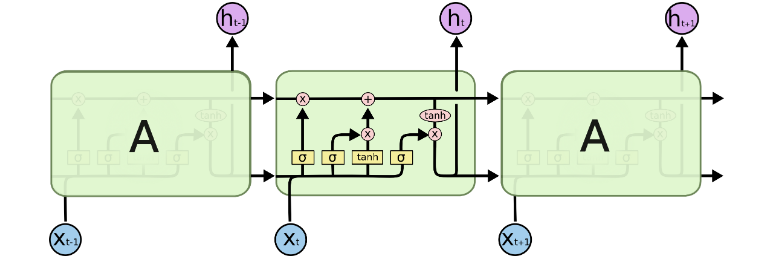
All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tan h layer.



**Figure2 The repeating module in a standard RNN**

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

In the below diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent point wise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations.



**Figure3 The repeating module in an LSTM contains four interacting layers.**

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture that is particularly well-suited for sequence data like text. LSTM networks are commonly used in sentimental analysis tasks because they can capture long-range dependencies and temporal patterns in text data.

The LSTM model architecture typically consists of an embedding layer, one or more LSTM layers, and a dense output layer. The embedding layer converts the input tokens into dense numerical vectors, which are then fed into the LSTM layers. The LSTM layers process the sequential data and capture the contextual information present in the text. Finally, the output layer produces predictions for the sentiment of the input text.

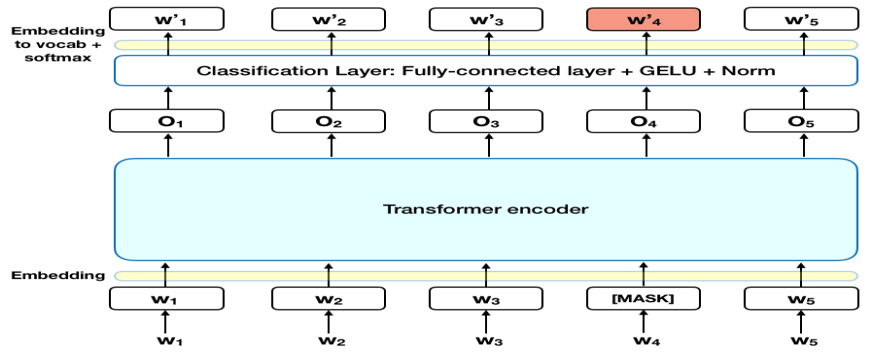
**BERT (Bidirectional Encoder Representations from Transformers)**

BERT (Bidirectional Encoder Representations from Transformers) is a recent paper published by researchers at Google AI Language. It has caused a stir in the Machine Learning community by presenting state-of-the-art results in a wide variety of NLP tasks, including Question Answering, Natural Language Inference (MNLI), and others.

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

As opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore, it is considered bidirectional, though it would be more accurate to say that it’s non-directional. This characteristic allows the model to learn the context of a word based on all of its surroundings (left and right of the word).

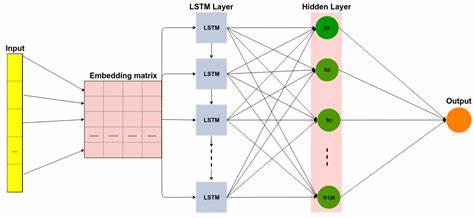
The chart below is a high-level description of the Transformer encoder. The input is a sequence of tokens, which are first embedded into vectors and then processed in the neural network. The output is a sequence of vectors of size H, in which each vector corresponds to an input token with the same index.



**Figure4 The Working of Bert**

BERT is highly effective for sentiment analysis tasks because it can capture contextual information and semantic meaning from text data, leading to more accurate sentiment predictions compared to traditional models. Additionally, fine-tuning BERT on specific sentiment analysis datasets allows the model to adapt to the nuances and characteristics of the sentiment analysis task, further improving its performance.

Top of Form

****

**Figure5 The Architecture of Bert and Lstm**

**3.3 UML DIAGRAMS**

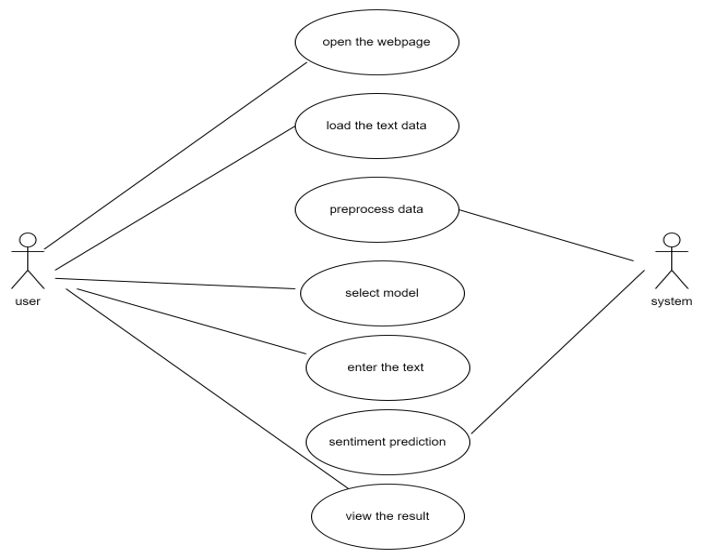
UML stands for Unified Modeling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system.

**3.2.1 USE CASE DIAGRAM**

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

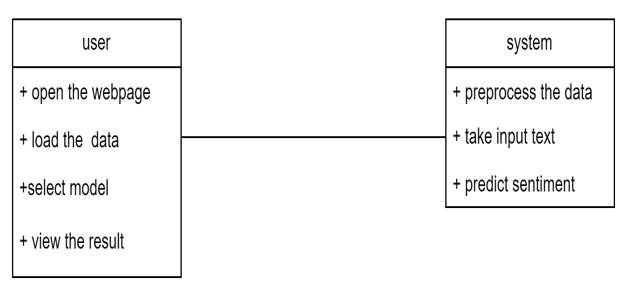
****

**Figure6 Use Case diagram**

**3.2.2 CLASS DIAGRAM**

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

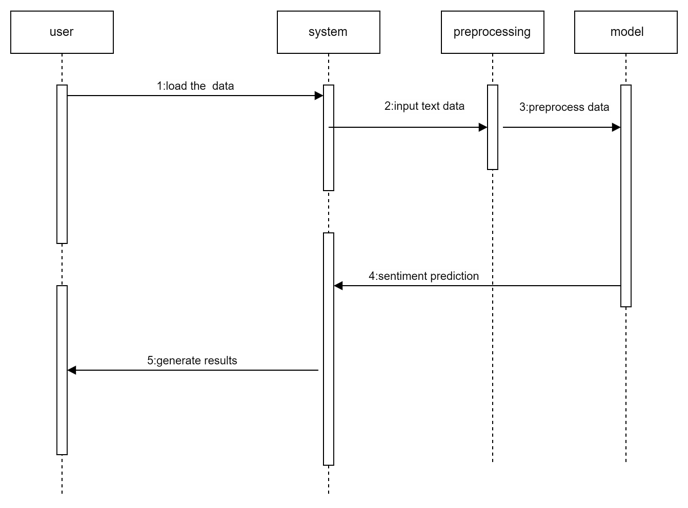


**Figure7 Class Diagram**

**3.2.3 SEQUENCE DIAGRAM**

A sequence diagram is an interaction diagram. From the name, it is clear that the diagram deals with some sequences, which are the sequence of message flowing from one object to another. They illustrate how the different parts of a system interact with each other to carry out a function, and the order in which the interactions occur when a particular use case is executed.

A sequence diagram has objects, self-message, object lifeline, message return etc.,.

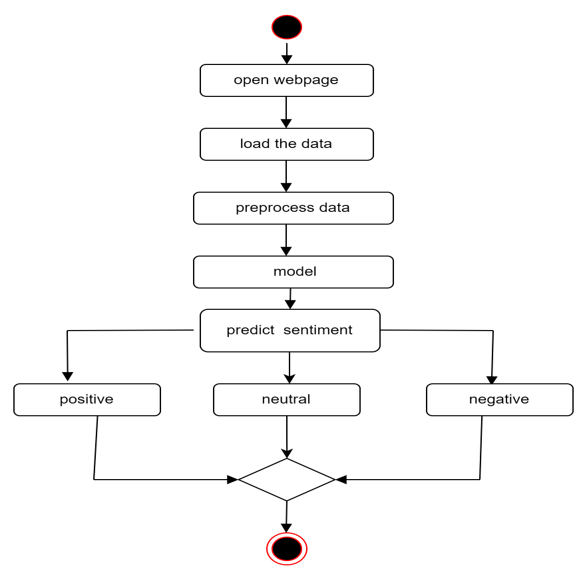
****

**Figure8 Sequence diagram**

**3.2.4 ACTIVITY DIAGRAM**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system.

Activity diagram describes the flow control of a system. The flow can be sequential, concurrent, or branched. It consists of activities and links. Activities are nothing but the functions of a system. An activity is essentially a flow chart. This is prepared to have an idea of how the system will work when executed.

****

**Figure9 Activity diagram**

**CHAPTER 4**

**IMPLEMENTATION**

The implementation phase of a software development project is the stage where the actual development takes place. It involves converting the design and plans into a functional software product. This phase typically includes activities such as coding, testing, debugging, and documentation. It is a crucial phase of the software development lifecycle and requires careful planning and execution to ensure the successful delivery of the final product.

**Machine Learning Algorithms**

Machine learning algorithms are a set of mathematical and statistical models that enable machines to learn from data and make predictions or decisions without being explicitly programmed. There are various machine learning algorithms available, such as decision trees, random forests, logistic regression, k-nearest neighbors, support vector machines, and Naive Bayes, among others. Each algorithm has its strengths and weaknesses and can be used to solve different types of problems, depending on the nature of the data and the desired outcome.

Algorithms used are:

**RANDOM FOREST CLASSIFIER**

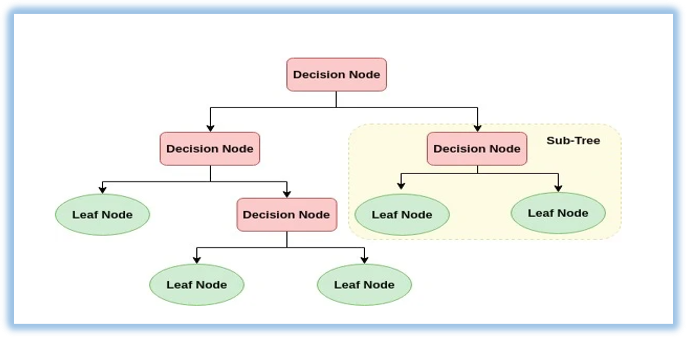
A random forest is a machine learning technique that’s used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems. A random forest algorithm consists of many decision trees. The ‘forest’ generated by the random forest algorithm is trained through bagging or bootstrap aggregating. Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms.

The (random forest) algorithm establishes the outcome based on the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome.

A random forest eradicates the limitations of a decision tree algorithm. It reduces the over fitting of datasets and increases precision. It generates predictions without requiring many configurations in packages (like Scikit-learn).

**DECISION TREE**

Decision tree is a flowchart-like tree structure where an internal node represents feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.



**Figure10 The Decision Tree diagram**

**LOGISTIC REGRESSION**

Logistic Regression was used in the biological sciences in early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable(target) is categorical. Logistic regression is used to solve classification problems, and the most common use case is [binary logistic regression](https://en.wikipedia.org/wiki/Logistic_regression), where the outcome is binary (yes or no).

1. In the real world, you can see logistic regression applied across multiple areas and fields.
2. In health care, logistic regression can be used to predict if a tumour is likely to be benign or malignant.
3. In the financial industry, logistic regression can be used to predict if a transaction is fraudulent or not.
4. In marketing, logistic regression can be used to predict if a targeted audience will respond or not.

**Deep Learning Algorithms**

Deep learning is a subfield of machine learning focused on artificial neural networks with multiple layers. It involves training neural networks on large datasets to learn complex patterns and representations from raw data. Deep learning algorithms automatically discover features from the input data, making them highly effective for tasks like image and speech recognition, natural language processing, and reinforcement learning. These algorithms use gradient-based optimization methods to iteratively adjust the network parameters and minimize the loss function. Popular deep learning architectures include convolutional neural networks (CNNs) for image processing, recurrent neural networks (RNNs) for sequential data, and transformers for natural language understanding. Deep learning models require large amounts of labeled data for training and can benefit from computational resources like GPUs and TPUs to accelerate training. Transfer learning, where pre-trained models are fine-tuned on specific tasks, is common in deep learning to leverage learned features from large datasets. Deep learning has revolutionized many industries, including healthcare, finance, and autonomous driving, by enabling breakthroughs in pattern recognition and decision-making tasks.

**BERT (Bidirectional Encoder Representations from Transformers)**

BERT (Bidirectional Encoder Representations from Transformers) is a powerful pre-trained language model that has been widely adopted for various natural language processing (NLP) tasks, including sentiment analysis. Here's how BERT is typically used for sentiment analysis. The input text data is tokenized into sub words or word pieces using the tokenizer, which is the same tokenizer used during BERT pre-training. The input sequences are then formatted according to BERT's input specifications.

BERT is highly effective for sentiment analysis tasks because it can capture contextual information and semantic meaning from text data, leading to more accurate sentiment predictions compared to traditional models. Additionally, fine-tuning BERT on specific sentiment analysis datasets allows the model to adapt to the nuances and characteristics of the sentiment analysis task, further improving its performance.

**Long Short-Term Memory (LSTM)**

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture that is particularly well-suited for sequence data like text. LSTM networks are commonly used in sentimental analysis tasks because they can capture long-range dependencies and temporal patterns in text data.

Here's how LSTM is typically used for sentimental analysis:

**1.Data Preparation**: The first step is to preprocess the text data by tokenizing it into individual words or tokens and converting it into a numerical format that can be fed into the LSTM model. This may involve techniques like word embedding, where words are represented as dense vectors in a continuous vector space.

**2.Model Architecture**: The LSTM model architecture typically consists of an embedding layer, one or more LSTM layers, and a dense output layer. The embedding layer converts the input tokens into dense numerical vectors, which are then fed into the LSTM layers. The LSTM layers process the sequential data and capture the contextual information present in the text. Finally, the output layer produces predictions for the sentiment of the input text.

**3.Training**: The LSTM model is trained on a labeled dataset of text data, where each example is associated with a sentiment label (e.g., positive, negative, neutral). During training, the model learns to map input text sequences to the corresponding sentiment labels by adjusting its parameters using techniques like backpropagation and gradient descent.

**4.Evaluation**: After training, the performance of the LSTM model is evaluated on a separate validation dataset to assess its ability to classify sentiment accurately. Common evaluation metrics include accuracy, precision, recall, and F1-score.

**4.1 DATASET**

The dataset for the sentimental analysis project likely consists of text data annotated with sentiment labels such as positive, negative, or neutral. It may contain user-generated content from various sources like social media posts, product reviews, customer feedback, or news articles. The dataset needs to be sufficiently large and diverse to capture different language styles, topics, and sentiments accurately. It should also be properly pre-processed to handle issues like noise, misspellings, punctuation, and stop words. Additionally, it's essential to ensure that the dataset is balanced across different sentiment classes to prevent bias in the model's training.

**4.2 CODING**

import numpy as np # linear algebra

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.feature\_extraction.text import CountVectorizer

from keras.preprocessing.text import Tokenizer

from keras.utils import pad\_sequences

from keras.models import Sequential

from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D

from sklearn.model\_selection import train\_test\_split

#from keras.utils.np\_utils import to\_categorical

from sklearn.utils import resample

from sklearn.utils import shuffle

from sklearn.metrics import confusion\_matrix,classification\_report

import re

The code uses TensorFlow version 2.15.

#loading the data set

df = pd.read\_csv(r'train.csv',encoding= 'latin1')

df.head()

df.tail()

df.shape

(27480, 10)

df.columns

ndex(['textID', 'text', 'selected\_text', 'sentiment', 'Time of Tweet',

'Age of User', 'Country', 'Population -2020', 'Land Area (Kmï¿½)',

'Density (P/Kmï¿½)'],

dtype='object')

df.dtypes

#Plot the graph to check whether there are any missing value present

missing = pd.DataFrame((df.isnull().sum())\*100/df.shape[0]).reset\_index()

plt.figure(figsize=(16,5))

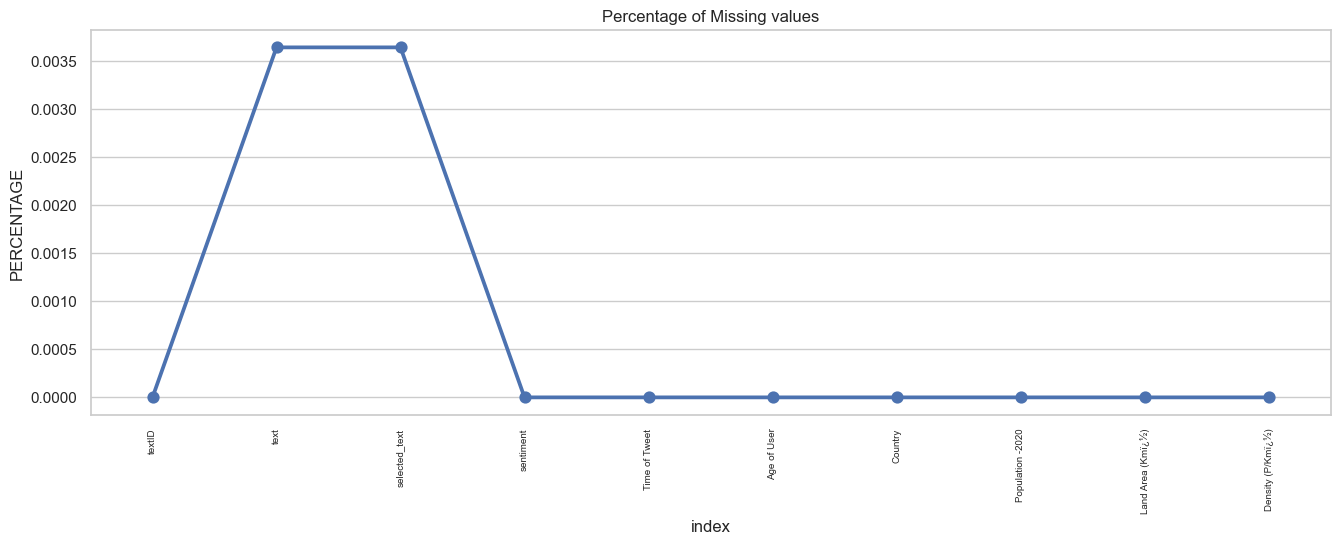
ax = sns.pointplot(x='index',y=0,data=missing)

plt.xticks(rotation =90,fontsize =7)

plt.title("Percentage of Missing values")

plt.ylabel("PERCENTAGE")

plt.show()



df.dropna(inplace=True)

df.isnull().sum()

df = df[['text', 'sentiment']]

df.sentiment.value\_counts()

sentiment

neutral 11117

positive 8581

negative 7781

Name: count, dtype: int64

#Cleaning the text data

def text\_clean(text):

    # changing to lower case

    lower = text.str.lower()

    # Replacing the repeating pattern of &#039;

    pattern\_remove = lower.str.replace("&#039,!\*;", "")

    # Removing all the special Characters

    special\_remove = pattern\_remove.str.replace(r'[^\w\d\s]',' ')

    # Removing all the non ASCII characters

    ascii\_remove = special\_remove.str.replace(r'[^\x00-\x7F]+',' ')

  # Removing the leading and trailing Whitespaces

    whitespace\_remove = ascii\_remove.str.replace(r'^\s+|\s+?$','')

  # Replacing multiple Spaces with Single Space

    multiw\_remove = whitespace\_remove.str.replace(r'\s+',' ')

    # Replacing Two or more dots with one

    dataframe = multiw\_remove.str.replace(r'\.{2,}', ' ')

    return dataframe

# Visualizing the data

df['text\_clean'] = text\_clean(df['text'])

df = df[['text\_clean','sentiment']]

sns.set(style="whitegrid")

plt.figure(figsize=(8,5))

total = float(len(df))

ax = sns.countplot(x="sentiment", hue="sentiment", data=df)

plt.title('Analysis of Data', fontsize=20)

for p in ax.patches:

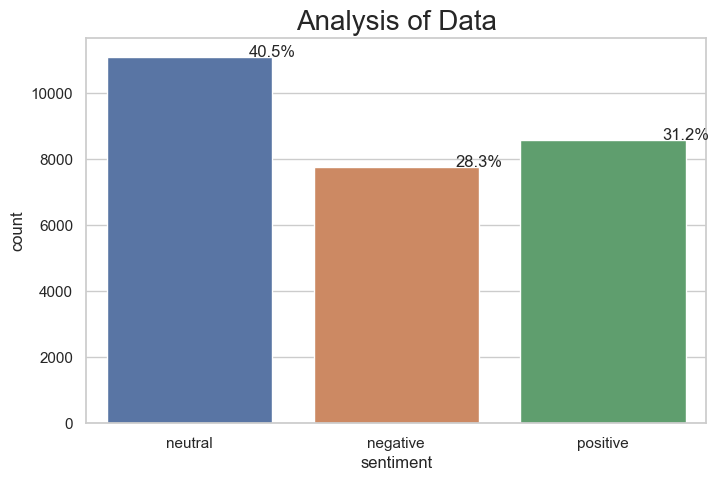
    percentage = '{:.1f}%'.format(100 \* p.get\_height()/total)

    x = p.get\_x() + p.get\_width()

    y = p.get\_height()

    ax.annotate(percentage, (x, y),ha='center')

plt.show()



#Preprocessing the data

df = df[:10000]

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df['sentiment'] = le.fit\_transform(df['sentiment'])

df.head()

df.tail()

df.shape

(10000, 2)

x = df['text\_clean']

y= df['sentiment']

#data splitting

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y, stratify=y, test\_size=0.3, random\_state=101)

#Feature extraction

from sklearn.feature\_extraction.text import HashingVectorizer

hvectorizer = HashingVectorizer(n\_features=10000,norm=None,alternate\_sign=False,stop\_words='english')

x\_train = hvectorizer.fit\_transform(x\_train).toarray()

x\_test = hvectorizer.transform(x\_test).toarray()

x\_train

y\_train

#Machine learning models

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

dt.fit(x\_train,y\_train)

y\_pred = dt.predict(x\_test)

acc\_dt = accuracy\_score(y\_test,y\_pred)

print(acc\_dt)

0.6126666666666667

import pickle

filename = 'decision.sav'

pickle.dump(dt, open(filename, 'wb'))

model = pickle.load(open(filename, 'rb'))

from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier()

rf.fit(x\_train[:1000],y\_train[:1000])

y\_pred=rf.predict(x\_test)

acc\_rf=accuracy\_score(y\_pred,y\_test)

acc\_rf

0.5853333333333334

import pickle

filename = 'randomforest.sav'

pickle.dump(dt, open(filename, 'wb'))

model = pickle.load(open(filename, 'rb'))

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

# training the model

lr.fit(x\_train, y\_train)

y\_pred = lr.predict(x\_test)

acc\_lr = accuracy\_score(y\_test,y\_pred)

acc\_lr

0.6233333333333333

import pickle

filename = 'logistic.sav'

pickle.dump(dt, open(filename, 'wb'))

model = pickle.load(open(filename, 'rb'))

#Using deep learning model

# Now tokenizing the text column

max\_fatures = 2000

tokenizer = Tokenizer(num\_words=max\_fatures, split=' ')

tokenizer.fit\_on\_texts(df['text\_clean'].values)

X = tokenizer.texts\_to\_sequences(df['text\_clean'].values)

X = pad\_sequences(X)

X[:2]

Y = pd.get\_dummies(df['sentiment']).values

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y, test\_size = 0.20, random\_state = 42)

print(X\_train.shape,Y\_train.shape)

print(X\_test.shape,Y\_test.shape)

X\_train

Y\_train

 pip install ktrain

import ktrain

from ktrain import text

(x\_train,  y\_train), (x\_test, y\_test), preproc = text.texts\_from\_array(x\_train=X\_train, y\_train=y\_train,

                                                                       x\_test=X\_test, y\_test=y\_test,

                                                                       class\_names=class\_names,

                                                                       preprocess\_mode='bert',

                                                                       maxlen=350,

                                                                       max\_features=35000)

model = text.text\_classifier('bert', train\_data=(x\_train, y\_train), preproc=preproc)

learner = ktrain.get\_learner(model, train\_data=(x\_train, y\_train),

                             val\_data=(x\_test, y\_test),

                             batch\_size=6)

learner.fit\_onecycle(2e-5, 2)

learner.validate(val\_data=(x\_test, y\_test), class\_names=class\_names)

predictor = ktrain.get\_predictor(learner.model, preproc)

predictor.get\_classes()

['negative', 'neutral', 'positive']

embed\_dim = 128

lstm\_out = 196

model = Sequential()

model.add(Embedding(max\_fatures, embed\_dim,input\_length = X.shape[1]))

model.add(SpatialDropout1D(0.4))

model.add(LSTM(lstm\_out, dropout=0.2, recurrent\_dropout=0.2))

model.add(Dense(3,activation='softmax'))

model.compile(loss = 'categorical\_crossentropy', optimizer='adam',metrics = ['accuracy'])

print(model.summary())

Model: "sequential\_3"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_3 (Embedding) (None, 34, 128) 256000

spatial\_dropout1d\_3 (Spati (None, 34, 128) 0

alDropout1D)

lstm\_3 (LSTM) (None, 196) 254800

dense\_4 (Dense) (None, 3) 591

=================================================================

Total params: 511391 (1.95 MB)

Trainable params: 511391 (1.95 MB)

Non-trainable params: 0 (0.00 Byte)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

None

batch\_size = 128

model.fit(X\_train, Y\_train, epochs = 15, batch\_size=batch\_size, verbose = 1)

Epoch 1/15

63/63 [==============================] - 9s 144ms/step - loss: 0.4333 - accuracy: 0.8289

Epoch 2/15

63/63 [==============================] - 15s 232ms/step - loss: 0.4036 - accuracy: 0.8466

Epoch 3/15

63/63 [==============================] - 18s 286ms/step - loss: 0.3871 - accuracy: 0.8536

Epoch 4/15

63/63 [==============================] - 22s 345ms/step - loss: 0.3613 - accuracy: 0.8602

Epoch 5/15

63/63 [==============================] - 16s 260ms/step - loss: 0.3399 - accuracy: 0.8724

Epoch 6/15

63/63 [==============================] - 9s 142ms/step - loss: 0.3288 - accuracy: 0.8759

Epoch 7/15

63/63 [==============================] - 9s 136ms/step - loss: 0.3119 - accuracy: 0.8808

Epoch 8/15

63/63 [==============================] - 9s 139ms/step - loss: 0.2850 - accuracy: 0.8932

Epoch 9/15

63/63 [==============================] - 9s 136ms/step - loss: 0.2666 - accuracy: 0.9024

Epoch 10/15

63/63 [==============================] - 9s 138ms/step - loss: 0.2597 - accuracy: 0.9010

Epoch 11/15

63/63 [==============================] - 9s 143ms/step - loss: 0.2551 - accuracy: 0.9050

Epoch 12/15

63/63 [==============================] - 9s 141ms/step - loss: 0.2282 - accuracy: 0.9171

Epoch 13/15

...

Epoch 14/15

63/63 [==============================] - 9s 149ms/step - loss: 0.2253 - accuracy: 0.9158

Epoch 15/15

63/63 [==============================] - 9s 146ms/step - loss: 0.2091 - accuracy: 0.9237

from tensorflow.keras.models import Model

final\_model= Model.save(model,"lstm.h5")

from sklearn.metrics import classification\_report

# Assuming you have trained your model and obtained predictions on the test set

y\_prob = model.predict(X\_test)

# Convert probabilities to class labels

y\_pred = np.argmax(y\_prob, axis=1)

# Convert one-hot encoded labels back to original class labels

y\_true = [np.argmax(label) for label in Y\_test]

# Generate classification report

report = classification\_report(y\_true, y\_pred)

print(report)

63/63 [==============================] - 1s 14ms/step

precision recall f1-score support

0 0.57 0.60 0.58 540

1 0.57 0.58 0.58 774

2 0.72 0.68 0.70 686

accuracy 0.62 2000

macro avg 0.62 0.62 0.62 2000

weighted avg 0.62 0.62 0.62 2000

import time

message = "I am good "

start\_time = time.time()

prediction = predictor.predict(message)

print('predicted: {} ({:.2f})'.format(prediction, (time.time() - start\_time)))

predicted: positive (0.46)

**4.3 MODEL DEPLOYMENT**

**4.3.1 SOFTWARE USED**

**Flask**

Flask is a lightweight and flexible web framework for building web applications in Python. It is easy to use and does not require particular tools or libraries. Flask is built on top of the Werkzeug WSGI (Web Server Gateway Interface) toolkit and the Jinja2 template engine.

To use Flask, you need to install it first using pip, the package installer for Python. After installing Flask, you can create a Flask app by importing the Flask class and creating an instance of it. You can define routes using the app.route() decorator, which maps a URL to a Python function that handles the request. You can also define templates using Jinja2 syntax and render them using the render\_template() function.

Flask supports various extensions for adding functionalities such as handling forms, connecting to databases, and creating RESTful APIs. Flask also supports deploying your web app on various platforms such as Heroku, Google Cloud, and AWS.

Flask is a great choice for building small to medium-sized web applications in Python. Its simplicity and flexibility make it easy to learn and use, while still being powerful enough to handle complex use cases.

**4.3.2 DEPLOYMENT CODE**

**Flask Code for Model Deployment**

#import necessary libraries

# linear algebra

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.feature\_extraction.text import CountVectorizer

from keras.preprocessing.text import Tokenizer

from keras.utils import pad\_sequences

from keras.models import Sequential

from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D,Dropout

# from keras.utils.np\_utils import to\_categorical

from sklearn.utils import resample

from sklearn.utils import shuffle

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import make\_pipeline

from sklearn.ensemble import StackingClassifier

from sklearn.feature\_extraction.text import TfidfVectorizer

# from tensorflow.keras.models import Model

from flask import \*

import re

app=Flask(\_\_name\_\_)

@app.route('/')

def index():

    return render\_template("index.html")

@app.route('/about')

def about():

    return render\_template("about.html")

@app.route('/load',methods=["GET","POST"])

def load():

    global df, dataset

    if request.method == "POST":

        data = request.files['data']

        df = pd.read\_csv(r'train.csv',encoding= 'latin1')

        print('##########################################')

        print(df.isnull().sum())

        df = df.dropna()

        print(df.isnull().sum())

        print('##########################################')

        dataset = df.head(100)

        msg = 'Data Loaded Successfully'

        return render\_template('load.html', msg=msg)

    return render\_template('load.html')

def text\_clean(text):

    # changing to lower case

    lower = text.str.lower()

    # Replacing the repeating pattern of &#039;

    pattern\_remove = lower.str.replace("&#039;", "")

    # Removing all the special Characters

    special\_remove = pattern\_remove.str.replace(r'[^\w\d\s]',' ')

    # Removing all the non ASCII characters

    ascii\_remove = special\_remove.str.replace(r'[^\x00-\x7F]+',' ')

    # Removing the leading and trailing Whitespaces

    whitespace\_remove = ascii\_remove.str.replace(r'^\s+|\s+?$','')

    # Replacing multiple Spaces with Single Space

    multiw\_remove = whitespace\_remove.str.replace(r'\s+',' ')

    # Replacing Two or more dots with one

    dataframe = multiw\_remove.str.replace(r'\.{2,}', ' ')

    return dataframe

@app.route('/preprocess', methods=['POST', 'GET'])

def preprocess():

    global x, y, x\_train, x\_test, y\_train, y\_test,  hvectorizer,df

    if request.method == "POST":

        size = int(request.form['split'])

        size = size / 100

        from sklearn.preprocessing import LabelEncoder

        le=LabelEncoder()

        df = df[['text', 'sentiment']]

        df.head()

        df['text\_clean'] = text\_clean(df['text'])

        df = df[['text\_clean','sentiment']]

        df['sentiment'] = le.fit\_transform(df['sentiment'])

        df.head()

        df.columns

       # Assigning the value of x and y

        x = df['text\_clean']

        y= df['sentiment']

        x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y, stratify=y, test\_size=0.3, random\_state=101)

        from sklearn.feature\_extraction.text import HashingVectorizer

        hvectorizer = HashingVectorizer(n\_features=5000,norm=None,alternate\_sign=False,stop\_words='english')

        x\_train = hvectorizer.fit\_transform(x\_train).toarray()

        x\_test = hvectorizer.transform(x\_test).toarray()

        # describes info about train and test set

        print("Number transactions X\_train dataset: ", x\_train.shape)

        print("Number transactions y\_train dataset: ", y\_train.shape)

        print("Number transactions X\_test dataset: ", x\_test.shape)

        print("Number transactions y\_test dataset: ", y\_test.shape)

        print(x\_train)

        print(x\_test)

        print(y\_train)

        print(y\_test)

        return render\_template('preprocess.html', msg='Data Preprocessed and It Splits Successfully')

    return render\_template('preprocess.html')

import pickle

@app.route('/model', methods=['POST', 'GET'])

def model():

    if request.method == "POST":

        global model

        print('ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc')

        s = int(request.form['algo'])

        if s == 0:

            return render\_template('model.html', msg='Please Choose an Algorithm to Train')

        elif s == 1:

            from sklearn.tree import DecisionTreeClassifier

            filename = 'decision.sav'

            model = pickle.load(open(filename, 'rb'))

            score=0.617526482

            ac\_dt = score \* 100

            msg = 'The accuracy obtained by DecisionTreeClassifier is ' + str(ac\_dt) + str('%')

            return render\_template('model.html', msg=msg)

        elif s == 2:

            from sklearn.ensemble import RandomForestClassifier

            filename = 'randomforest.sav'

            model = pickle.load(open(filename, 'rb'))

            score=0.5873333333333334

            ac\_rf = score \* 100

            msg = 'The accuracy obtained by RandomForestClassifier is ' + str(ac\_rf) + str('%')

            return render\_template('model.html', msg=msg)

        elif s == 3:

            from sklearn.linear\_model import LogisticRegression

            filename = 'logistic.sav'

            model = pickle.load(open(filename, 'rb'))

            score=0.6233333333333333

            ac\_lr = score \* 100

            msg = 'The accuracy obtained by LogisticRegression is ' + str(ac\_lr) + str('%')

            return render\_template('model.html', msg=msg)

        elif s == 4:

            # from keras.models import Sequential

            # from keras.layers import Dense, Dropout,LSTM

            # from keras.models import load\_model

            # model = load\_model('lstm.h5')

            score=0.9237

            ac\_ls= score \* 100

            msg = 'The accuracy obtained by  LSTM is ' + str(ac\_ls) + str('%')

            return render\_template('model.html', msg=msg)

    return render\_template('model.html')

import pickle

@app.route('/prediction',methods=['POST','GET'])

def prediction():

    global x\_train,y\_train

    if request.method == "POST":

        f1 = request.form['text']

        print(f1)

        filename = (r'lstm.h5')

        model = pickle.load(open(filename, 'rb'))

        from sklearn.feature\_extraction.text import HashingVectorizer

        hvectorizer = HashingVectorizer(n\_features=10000,norm=None,alternate\_sign=False)

        result =model.predict(hvectorizer.transform([f1]))

        print("Results: ",result)

        if result==0:

            msg = 'It is a negative statement'

        elif result==1:

            msg= 'It is a neutral statement'

        else:

            msg= 'It is a positive  statement'

        return render\_template('prediction.html',msg=msg)

    return render\_template('prediction.html')

if \_\_name\_\_=="\_\_main\_\_":

    app.run(debug=True)

This is a Flask web application that allows users to enter the text data and view sentiment prediction using the trained model.

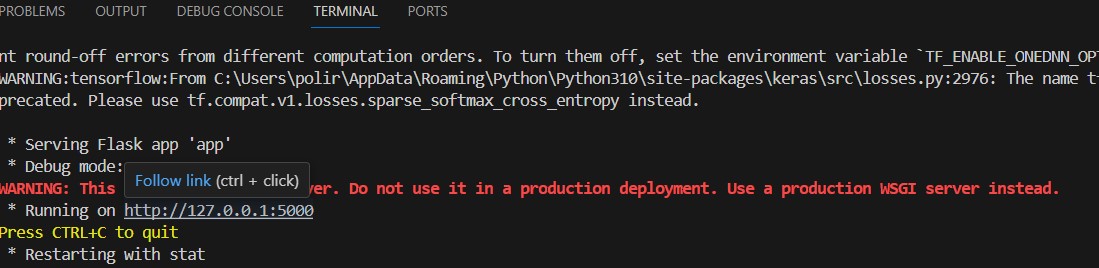
Finally, the Flask app is run using the app.run() method. This starts the server and listens for incoming requests.

**CHAPTER 5**

**OUTPUT SCREENS**

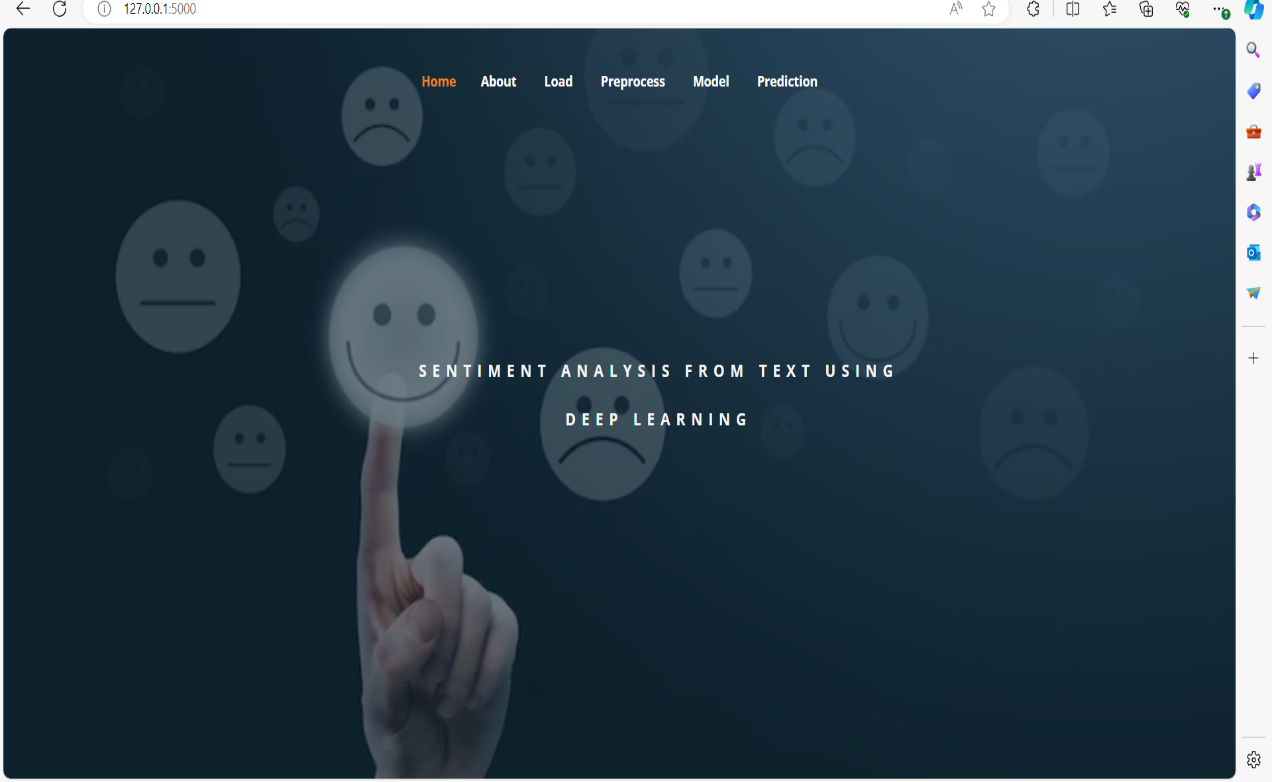
**5.1 Output of Flask code run**

To run the flask code the command given is” python run app.py” in the VS Code.



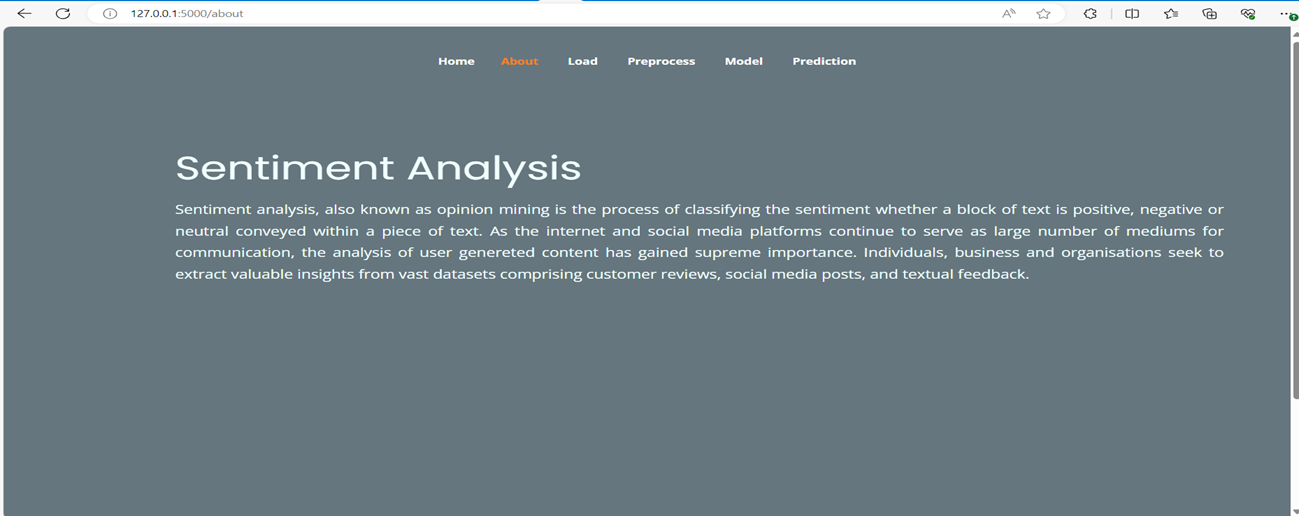
**Figure11 The output for user interface**

**5.2 Output of Web App “home.html”:**

****

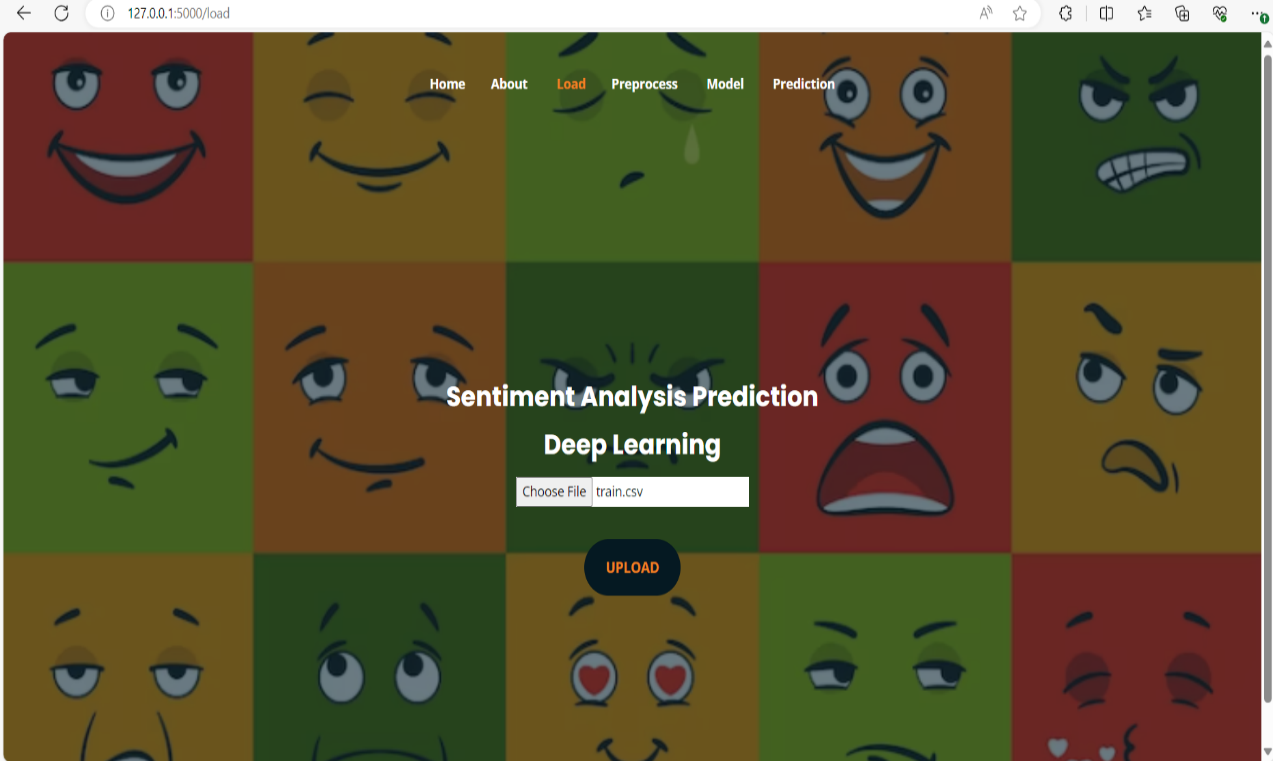
**Figure12 The home page**

**5.3 Output of “about.html”:**

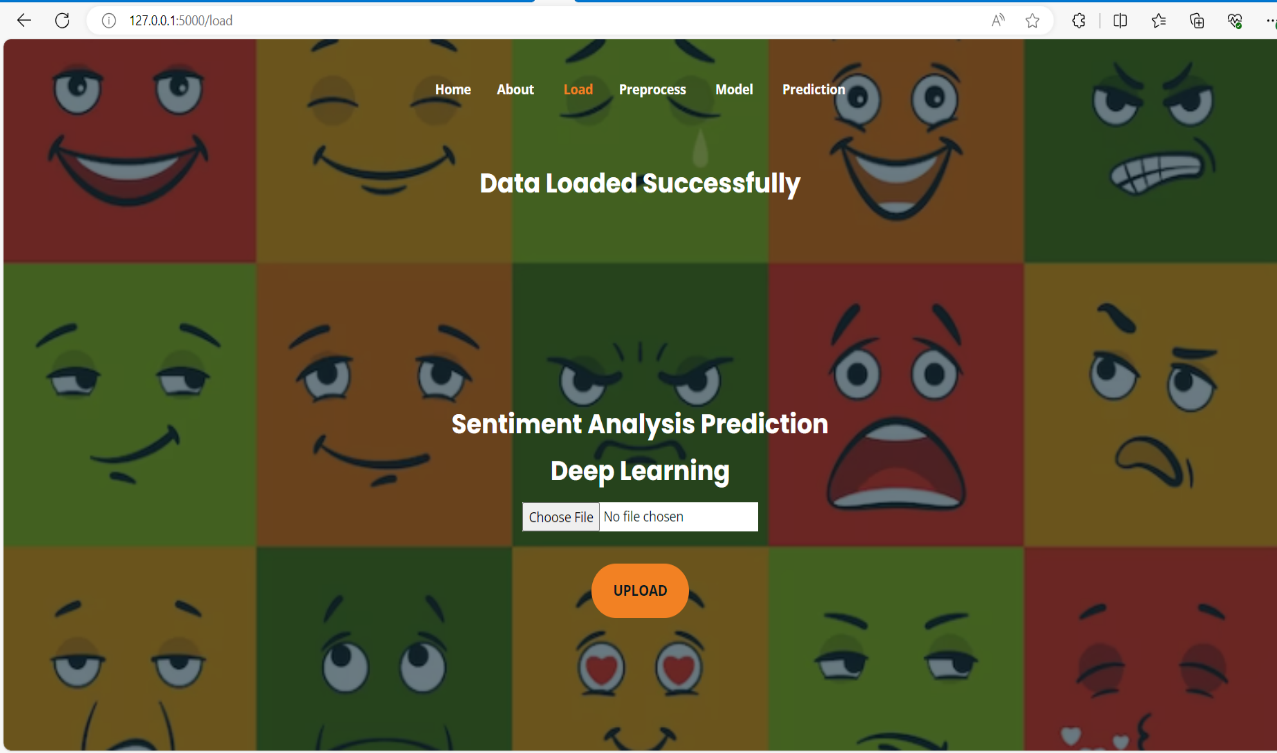
****

**Figure13 The about page**

**5.4 Output of “load.html”:**

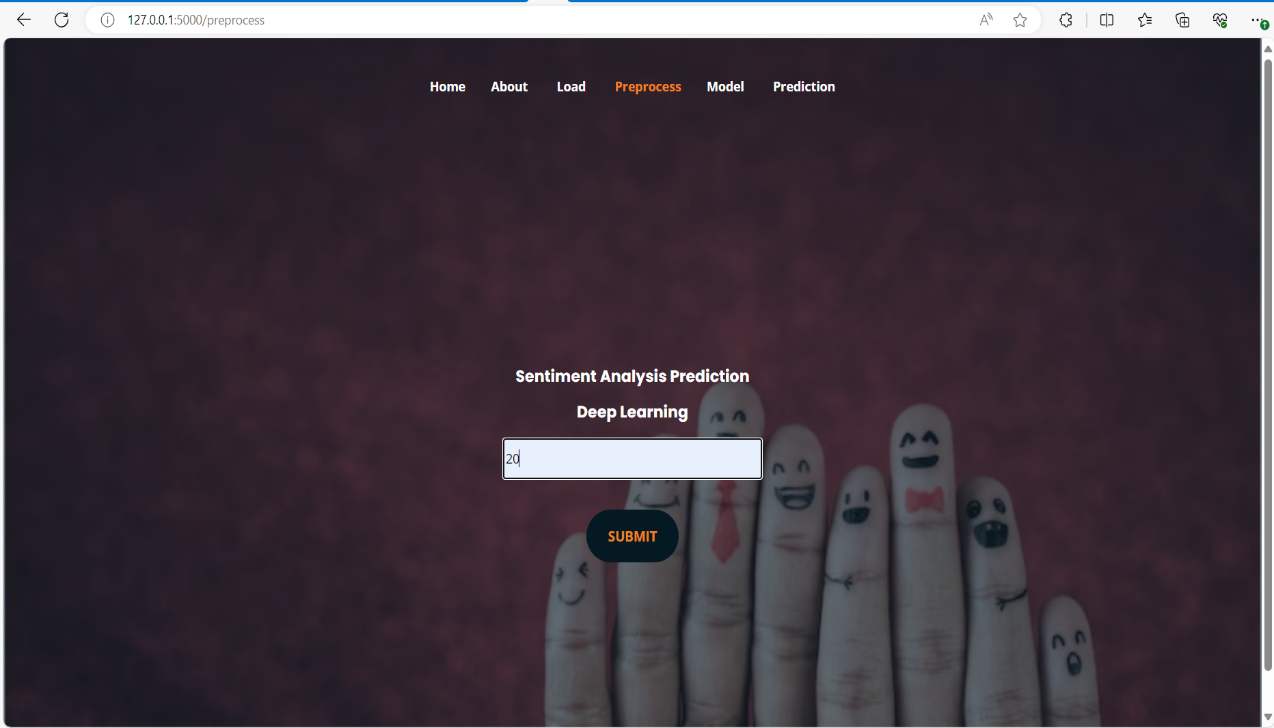
****

**Figure14 The load page for uploading file**

****

**Figure15 The load page displaying Data Loaded Successfully**

**5.4 Output of “preprocess.html”:**

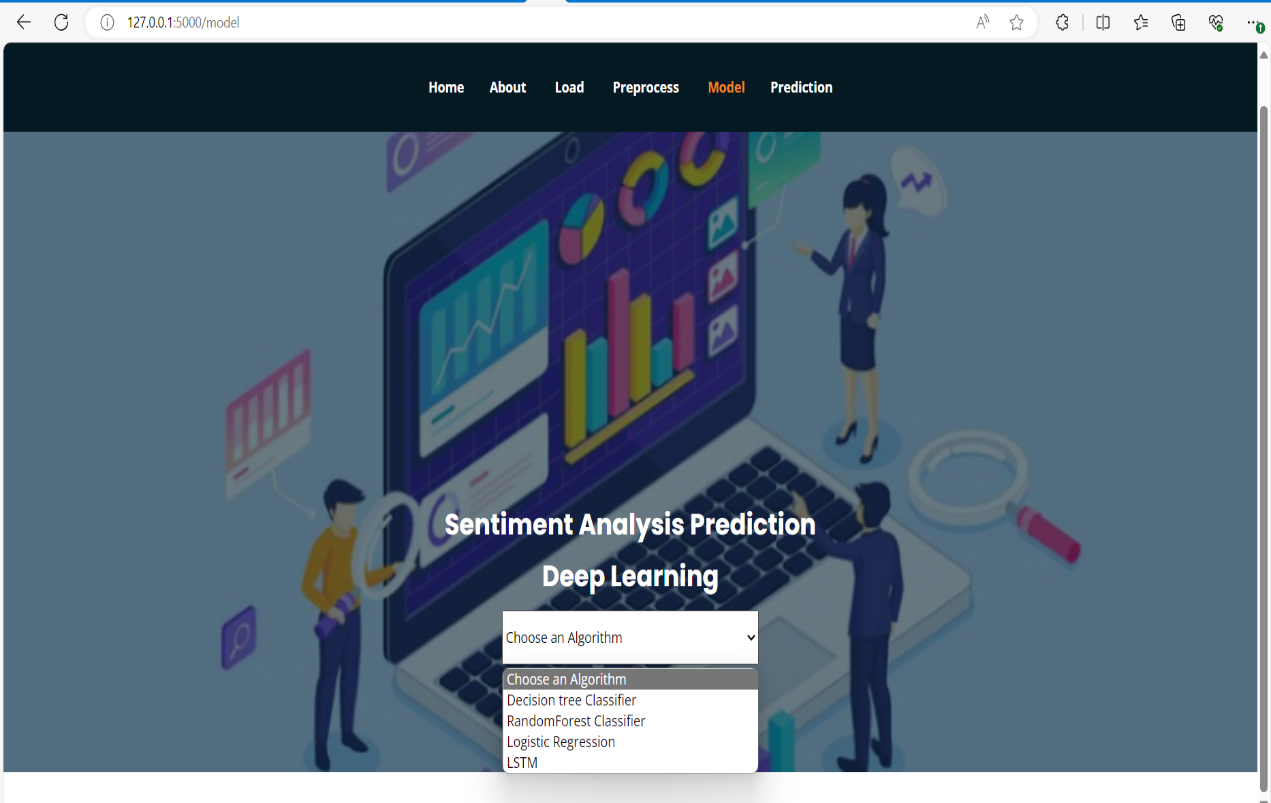
****

**Figure16 The preprocess page**

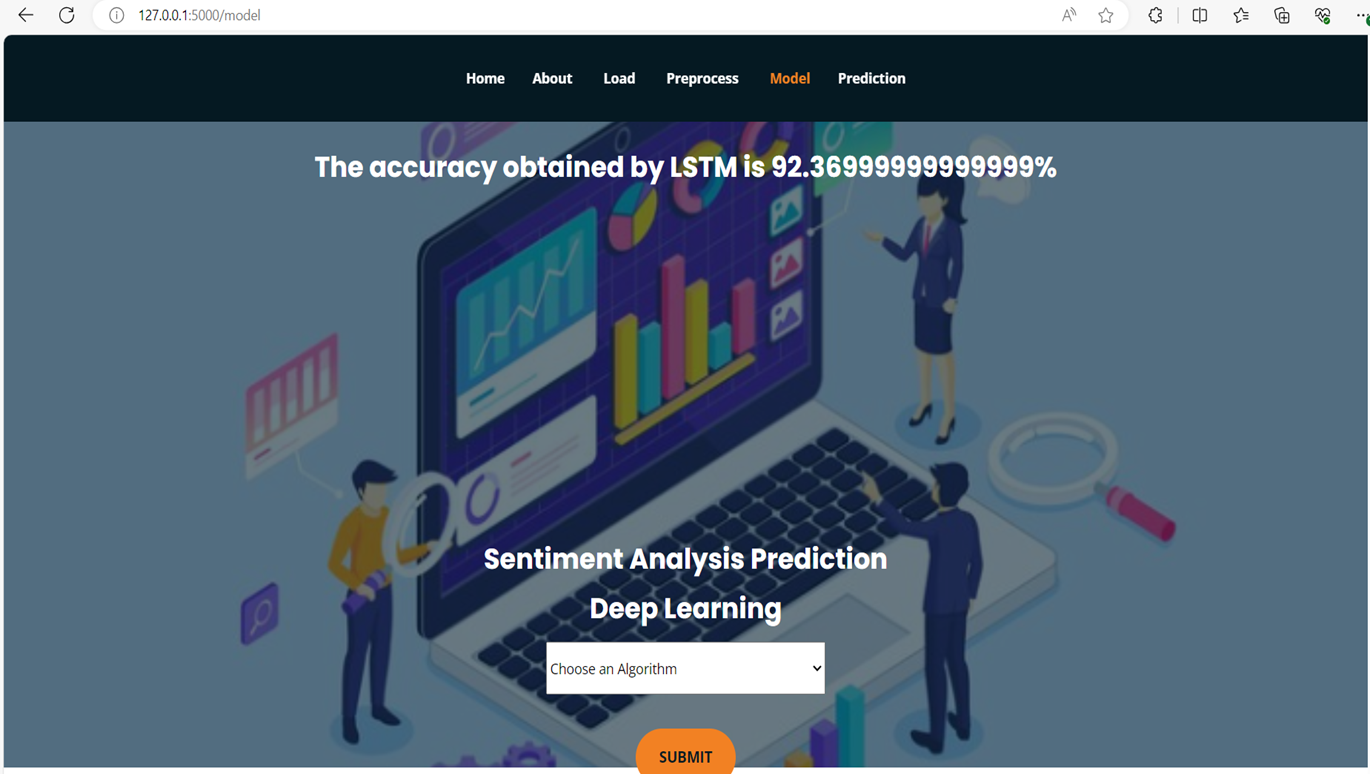
****

**Figure17 The preprocess page displaying Data pre-processed successfully**

**5.5 Output of “model.html”:**

****

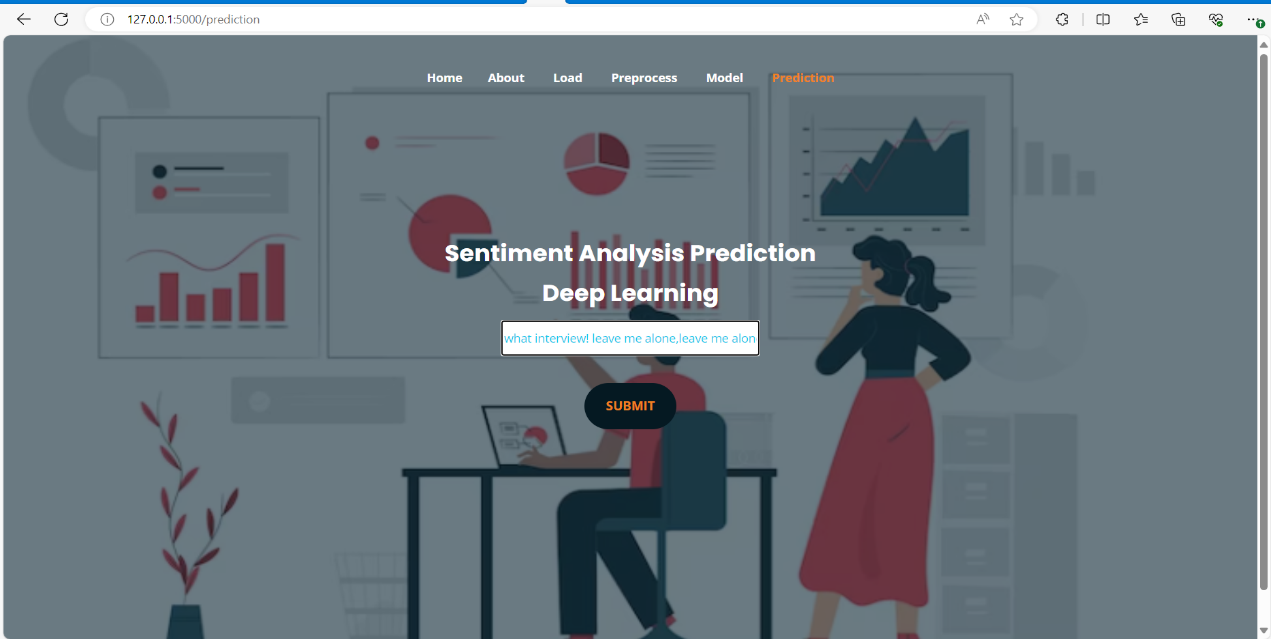
**Figure18 The model page for selecting the model**

****

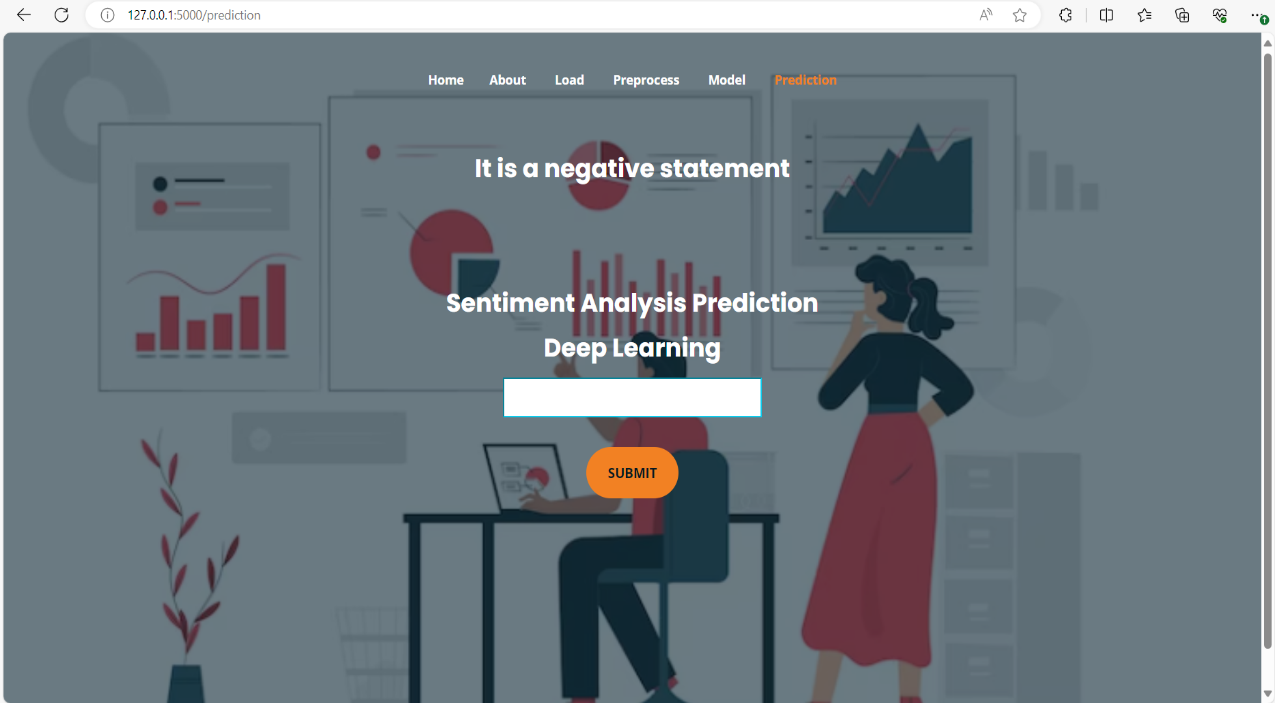
**Figure19 The model page displaying accuracy of the selected model**

**5.6 Output of “prediction.html”:**

**Example1**

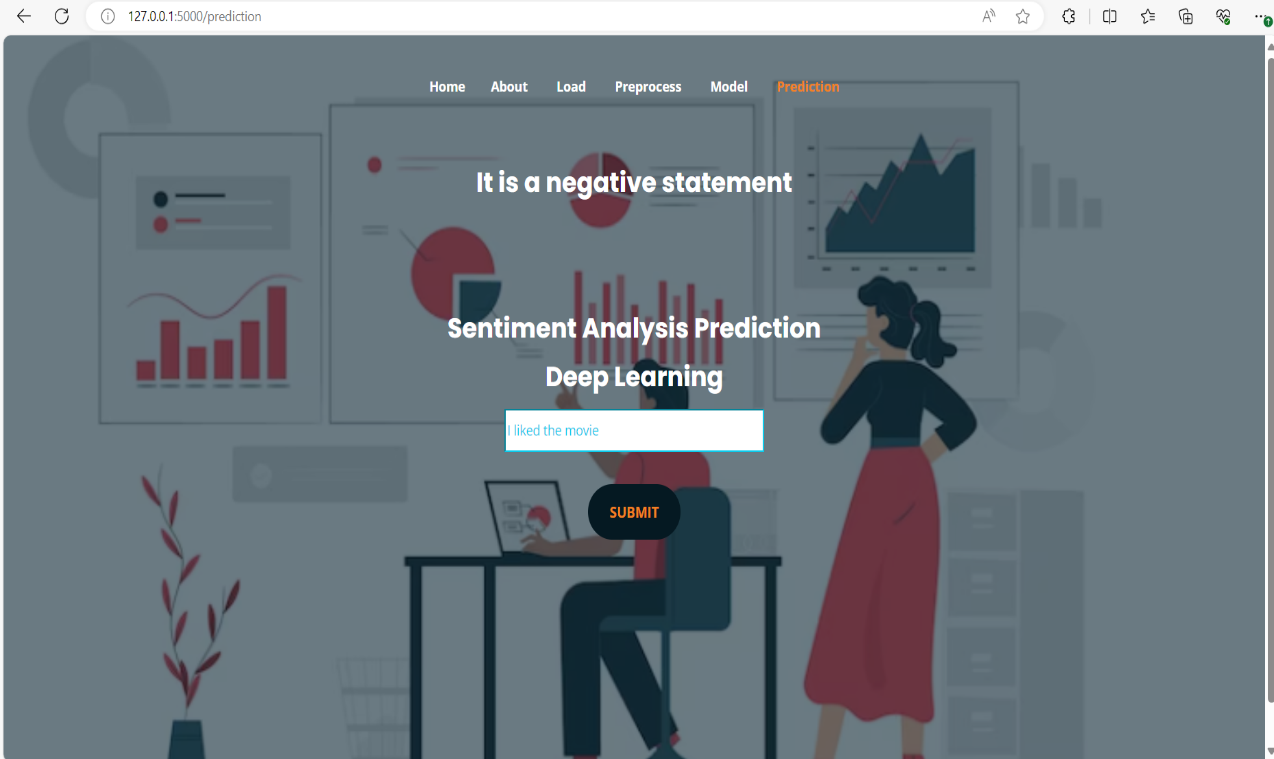
****

**Figure20 Prediction page for entering text data of example1**

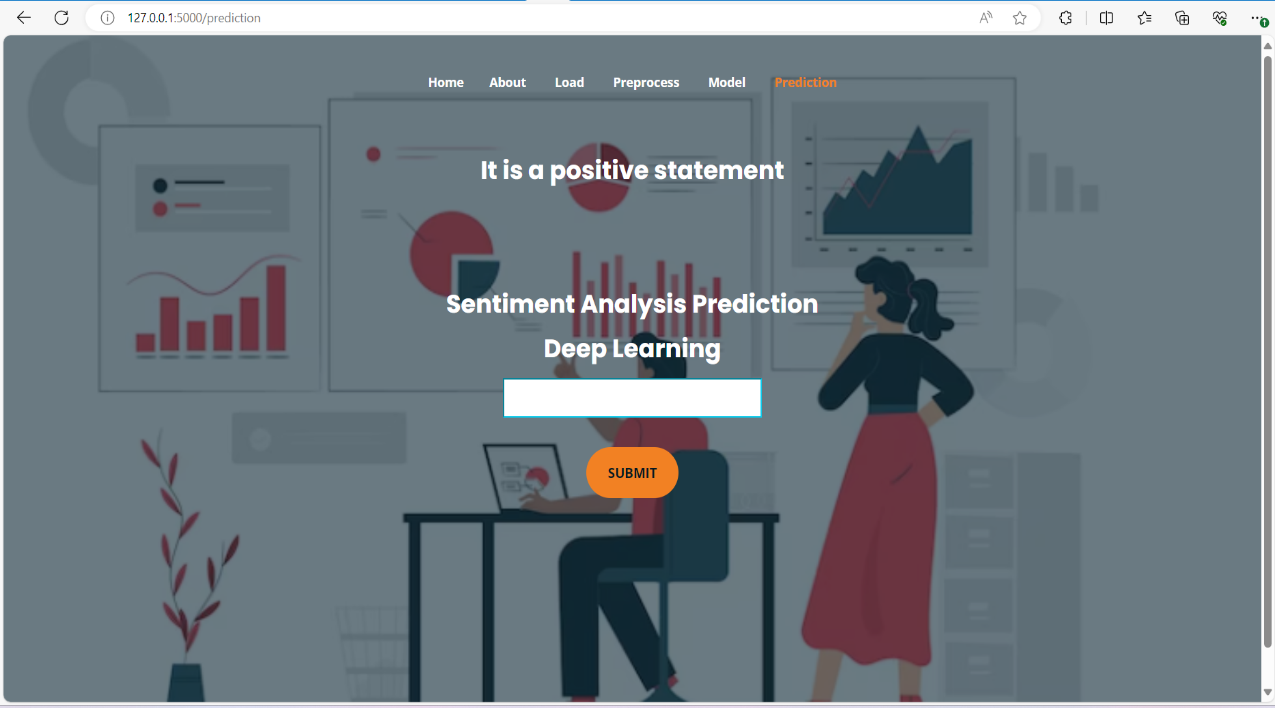
****

**Figure21 Prediction page displaying output for example1**

**Example2**

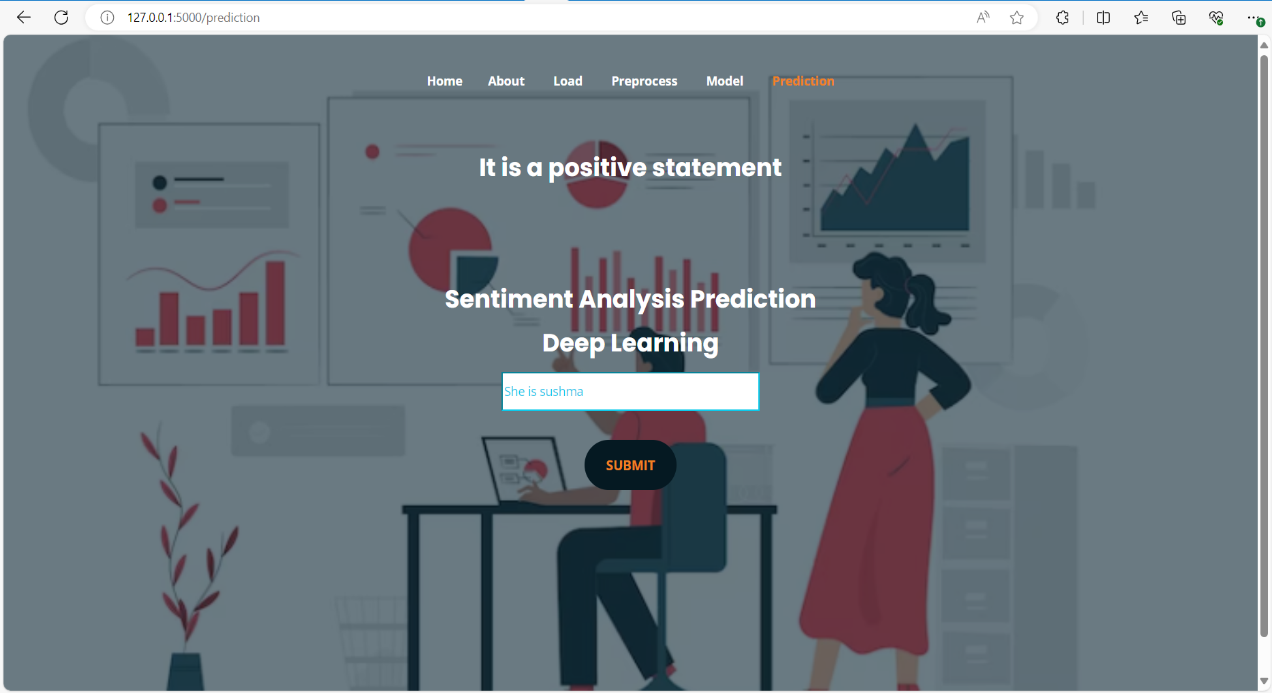
****

**Figure22 Prediction page for entering text data of example2**

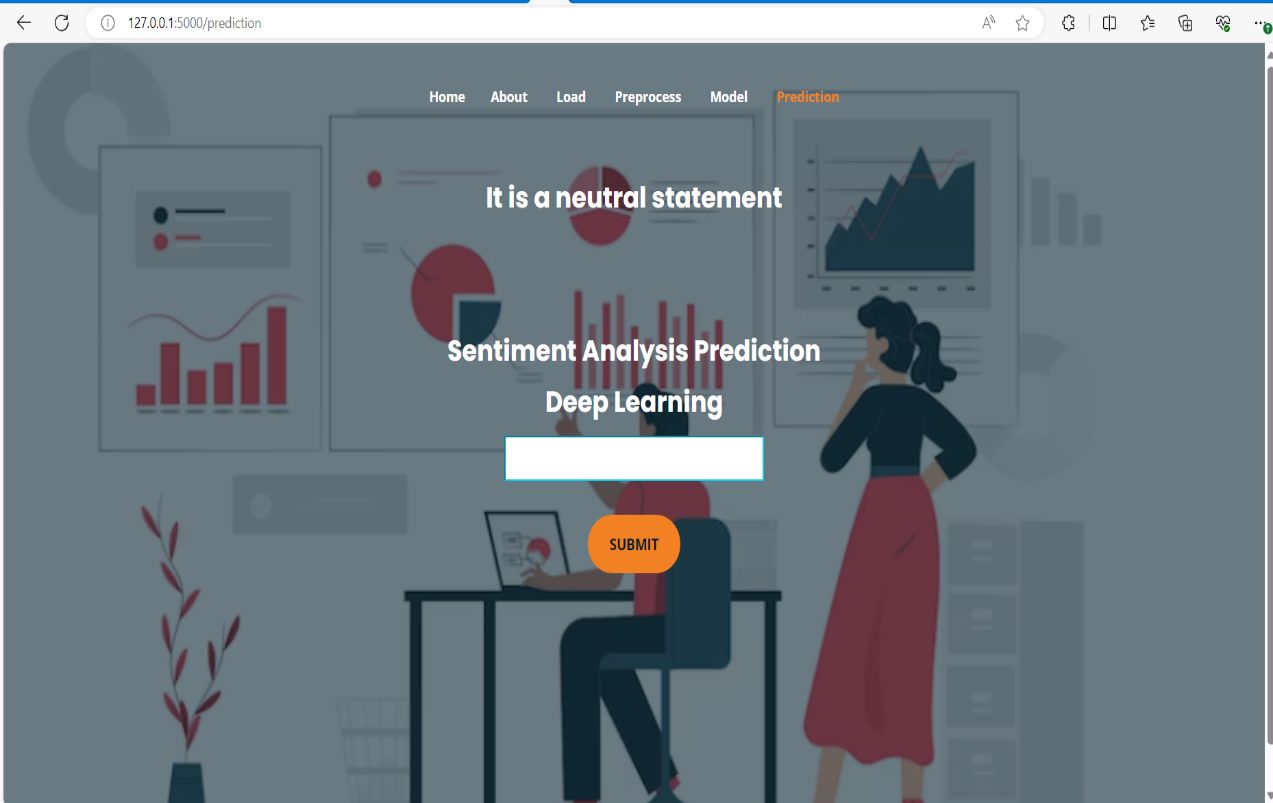
****

**Figure23 Prediction page displaying output for example2**

**Example3**

****

**Figure24 Prediction page for entering text data of example3**

****

**Figure25 Prediction page displaying output for example3**

**CHAPTER 6**

**CONCLUSION AND FUTURE SCOPE**

The conclusion of this project is that the developed model can classify the sentiment by analyzing the text data. The machine learning algorithms like Decision Tree, Random Forest and Logistic Regression have accuracies 61%, 58% and 62% respectively. The proposed deep learning model LSTM have 92% accuracy. So the deep learning model used to make sentiment predictions accurately. The project successfully implemented sentiment analysis using both LSTM and BERT models. The LSTM model demonstrated efficient processing of sequential data, while the BERT model leveraged pre-trained contextual embeddings for improved performance. Through this project, shown the effectiveness of deep learning techniques in analysing sentiment from text data. Furthermore, observed that BERT outperformed LSTM in capturing nuanced sentiment patterns due to its contextual understanding. However, both models exhibited promising results, highlighting their potential in various real-world applications such as social media monitoring, customer feedback analysis, and opinion mining. This project contributes to advancing the field of natural language processing by demonstrating the capabilities of deep learning models in sentiment analysis tasks.

**FUTURE SCOPE**

This can be implemented in future to identify the emotions and types of genres from the text or documents. Multilingual sentiment analysis: As the availability of multilingual data grows, so does the demand for sentiment analysis models that can process and analyse text in multiple languages. The LSTM and BERT algorithms can be extended to perform sentiment analysis in multiple languages. Real-time sentiment analysis: As more content is generated on social media platforms, the demand for real-time sentiment analysis is increasing. To provide real-time insights into user sentiment, LSTM and BERT algorithms can be trained and deployed in real-time systems.

Transfer Learning: Investigate transfer learning approaches to leverage the pre-trained sentiment analysis model for related tasks or domains, accelerating development and reducing data requirements.

**7. REFERENCES**

[1] G. Mohanty and P. Mishra, ‘‘Sad or glad? Corpus creation for Odia poetry with sentiment polarity information,’’ in Proc. 19th Int. Conf. Comput. Linguistics Intell. Text Process. (CICLing), Hanoi, Vietnam, 2018.

[2] Y. Hou and A. Frank, ‘‘Analysing sentiment in classical Chinese poetry,’’ in Proc. 9th SIGHUM Workshop Lang. Technol. Cultural Heritage, Social Sci., Hum. (LaTeCH), 2015, pp. 15–24.

[3] A. Ghosh, G. Li, T. Veale, P. Rosso, E. Shutova, J. Barnden, and A. Reyes, ‘‘SemEval-2015 task 11: Sentiment analysis of figurative language in Twitter,’’ in Proc. 9th Int. Workshop Semantic Eval. (SemEval), 2015, pp. 470–478.

[4] G. Rakshit, A. Ghosh, P. Bhattacharyya, and G. Haffari, ‘‘Automated analysis of Bangla poetry for classification and poet identification,’’ in Proc. 12th Int. Conf. Natural Lang. Process., Dec. 2015, pp. 247–253.

[5] K. Bischoff, C. S. Firan, R. Paiu, W. Nejdl, C. Laurier, and M. Sordo, ‘‘Music mood and theme classification-a hybrid approach,’’ in Proc. ISMIR, Oct. 2009, pp. 657–662.

[6] O. Alsharif, D. Alshamaa, and N. Ghneim, ‘‘Emotion classification in Arabic poetry using machine learning,’’ Int. J. Comput. Appl., vol. 65, p. 16, May 2013.

[7] A. Zehe, M. Becker, F. Jannidis, and A. Hotho, ‘‘towards sentiment analysis on German literature,’’ in Proc. Joint German/Austrian Conf. Artif. Intell. Cham, Switzerland: Springer, 2017, pp. 387–394.

[8] L. Barros, P. Rodriguez, and A. Ortigosa, ‘‘Automatic classification of literature pieces by emotion detection: A study on Quevedo’s poetry,’’ in Proc. Humaine Assoc. Conf. Affect. Comput. Intell. Interact., Sep. 2013, pp. 141–146.

[9] S. Soumya, S. Saju, R. Rajan, and N. Sebastian, ‘‘Poetic meter classification using TMS320C6713 DSK,’’ in Proc. Int. Conf. Signal Process. Commun. (ICSPC), Jul. 2017, pp. 23–27.