Sentiment Analysis of X(Twitter) Data: A Comparative Study of Machine Learning and Deep Learning Approaches

Project report submitted to the Amrita Vishwa Vidyapeetham in partial fulfilment of the requirement for the Degree of

BACHELOR of TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

C Ganesh AM.EN.U4CSE20063
K Parameswar Reddy AM.EN.U4CSE20141
P B Sudeep Sampath AM.EN.U4CSE20154
T Vignesh AM.EN.U4CSE20169

AMRITA SCHOOL OF COMPUTING

AMRITA VISHWA VIDYAPEETHAM
(Estd. U/S 3 of the UGC Act 1956)
AMRITAPURI CAMPUS

KOLLAM -690525

MAY 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM
(Estd. U/S 3 of the UGC Act 1956)

Amritapuri Campus

Kollam -690525

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"Sentiment Analysis of X(Twitter) Data: A Comparative Study of Machine Learning and Deep Learning Approaches"** submitted by **C Ganesh (AM.EN.U4CSE20063), K Parameswar Reddy (AM.EN.U4CSE20141), P B Sudeep Sampath (AM.EN.U4CSE20154) and T Vignesh (AM.EN.U4CSE20169)**, in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology in Computer Science and Engineering from Amrita Vishwa Vidyapeetham, is a bonafide record of the work carried out by them under my guidance and supervision at Amrita School of Computing, Amritapuri during Semester 8 of the academic year 2023-2024.

Lekshmi S N
Project Guide

Sarath S
Project Coordinator

Dr. Swaminathan J
Chairperson
Dept. of Computer Science & Engineering

Asha Ashok

Place : Amritapuri
Date : 15 May 2024

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

(Estd. U/S 3 of the UGC Act 1956)

Amritapuri Campus

Kollam -690525

DECLARATION

We, **C Ganesh (AM.EN.U4CSE20063), K Parameswar Reddy (AM.EN.U4CSE20141), P B Sudeep Sampath (AM.EN.U4CSE20154) and T Vignesh (AM.EN.U4CSE20169)** hereby declare that this project entitled **"Sentiment Analysis of X(Twitter) Data: A Comparative Study of Machine Learning and Deep Learning Approaches"** is a record of the original work done by us under the guidance of **Lekshmi S N**, Dept. of Computer Science and Engineering, Amrita Vishwa Vidyapeetham, that this work has not formed the basis for any degree/diploma/associationship/fellowship or similar awards to any candidate in any university to the best of our knowledge.

Place : Amritapuri

Date : 17 May 2024

Signature of the student

Signature of the Project Guide

Acknowledgement

We gratefully acknowledge the collective efforts of numerous individuals and resources that have contributed to the realization of this project. In particular, we extend our heartfelt appreciation to our mentor, **Lekshmi S N**, whose guidance, mentorship, and insightful feedback have been indispensable in shaping the trajectory of our work. We also express our sincere gratitude to **Sarath S**, our project coordinator, and **Asha Ashok**, our project reviewer, for their invaluable support and constructive input throughout the endeavor.

We also recognize the contributions of the authors of the research papers that served as pivotal references for our project. Their work has been integral to our literature review, fostering a deeper understanding and inspiring innovative solutions.

Furthermore, we acknowledge the exceptional commitment, diverse skill sets, and collaborative spirit of each member of our team. Their unwavering dedication during the development phase has been instrumental in the success of this project.

Lastly, we would like to acknowledge the Department of Computer Science Engineering for providing the conducive environment and resources that facilitated the execution of this project. Their support has been crucial in our journey towards achieving our goals

Abstract

The ever-growing volume of social media data presents both opportunities and challenges. This research delves into the domain of sentiment analysis on X (formerly Twitter), a platform pulsating with real-time public opinion. We explore the effectiveness of various Machine Learning (ML) algorithms (Logistic Regression, Linear SVM, Random Forest, Naive Bayes) and Deep Learning (DL) architectures (LSTM, CNN-LSTM, BiLSTM, CNN) in extracting sentiment from X conversations. By meticulously comparing their performance, this study aims to identify the most robust and efficient techniques for sentiment analysis tailored to the unique characteristics of X data, including informal language, slang, and sarcasm. The findings not only shed light on the strengths and weaknesses of each approach but also contribute to the development of advanced sentiment analysis tools. This research holds significant implications for businesses, governments, and social scientists seeking to leverage the power of X data to gain a deeper understanding of public sentiment in the digital age.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The advent of social media has revolutionized how individuals convey their emotions, opinions, and sentiments across various platforms, with X formerly known as Twitter being one of the most prevalent arenas for such exchanges. This platform offers a continuous stream of data in the form of tweets, which are reflective of real-time public sentiment on a wide range of topics. Sentiment analysis harnesses this data to decipher the subjective information embedded within tweets, a task that involves sophisticated computational techniques to categorize and predict sentiment accurately.

The primary aim of this research is to conduct a comparative analysis of several machine learning and deep learning approaches to sentiment analysis on X data. The study will evaluate traditional machine learning algorithms such as Logistic Regression, Linear SVM, Random Forest, and Naive Bayes, and compare them against advanced deep learning models like LSTM, CNN-LSTM, and BILSTM. This comparison seeks to identify which methodologies can most effectively classify and predict sentiments expressed in X feeds, thereby providing a robust foundation for future advancements in real-time

sentiment analysis. Additionally, the study intends to explore the impact of different preprocessing and feature extraction techniques in enhancing the accuracy of these models.

Chapter 2

Problem Definition

The core problem addressed by this project is the development of a robust and efficient system for sentiment analysis on X data. Existing sentiment analysis tools often struggle with the complexities of informal language, slang, and sarcasm prevalent on social media platforms like X. Additionally, the sheer volume and real-time nature of X data necessitate efficient and scalable solutions.

This project aims to overcome these limitations by exploring the comparative capabilities of Machine Learning and Deep Learning approaches. By identifying the most effective algorithms, the project seeks to contribute to the development of accurate and scalable sentiment analysis tools for X data.

2.1 Motivation

Deep learning approaches have shown promising results in natural language processing, particularly in handling the complexities of informal language. This project aims to leverage the power of deep learning to develop a more robust and accurate method for extracting and analyzing opinions from Twit-

ter data.

The ability to accurately analyze sentiment on X presents a multitude of opportunities. Businesses can utilize these insights to understand customer sentiment towards their products or services, enabling them to improve brand reputation and optimize marketing strategies. Governments can gain valuable feedback on public policies and social issues. Social scientists can leverage sentiment analysis tools to study social movements and emerging trends.

2.2 Challenges

- **Data Pre-Processing**: Requires substantial computational resources and time. Advanced algorithms and efficient parallel processing techniques are necessary.
- **Evolving language**: Twitter language is constantly evolving with new slang and trends, requiring models to adapt and learn continuously.
- **Context and ambiguity**: Tweets often lack context and contain ambiguities, making it difficult for models to correctly interpret sentiment and sarcasm

2.3 Objectives

- To contribute to the development of improved sentiment analysis tools specifically designed for handling the complexities of social media language used on X,
- This research provides a comprehensive comparison of various ML and

DL algorithms for sentiment analysis on X data.

- **E**mpower businesses, governments, and social scientists to leverage public opinion data.

2.4 Contributions

- **A** novel deep learning architecture tailored for Twitter sentiment analysis, demonstrating improved accuracy and robustness compared to existing methods.
- **T**his research provides a comprehensive comparison of various ML and DL algorithms for sentiment analysis on X data.
- **I**n-depth analysis of opinion dynamics on Twitter, revealing key trends and influencing factors valuable for diverse applications.

Chapter 3

Related Work

A significant body of research explores sentiment analysis on social media data, with a focus on Machine Learning (ML) algorithms. Krishnan et al. (2022) [2] and Mittal et al. (2019) [6] delve into the effectiveness of algorithms like Logistic Regression, Naive Bayes, Support Vector Machines, and Random Forests for sentiment classification tasks. These studies highlight the strengths and limitations of these approaches when applied to the unique characteristics of social media data, such as informal language and brevity.

Deep Learning (DL) architectures have emerged as a powerful alternative for sentiment analysis, particularly with their ability to capture complex relationships within text data. Huang et al. (2022) [3] provide a comprehensive exploration of deep learning approaches in sentiment analysis, demonstrating their potential for improved accuracy. Research specifically focused on social media sentiment analysis using LSTMs shows promising results. Gattan (2022) [4] and Chaudhary et al. (2023) [1] demonstrate the effectiveness of LSTMs in handling informal language structures commonly found on platforms like Twitter. Additionally, studies by Ramasamy et al. (2021) [**?**] and Ramneet et al. (2020) [8] explore the performance of sentiment analysis

on Twitter data using Support Vector Machines (SVMs), providing further insights into effective algorithms for similar platforms like X.

While LSTMs and SVMs have shown promise, a comprehensive comparison of various ML and DL algorithms specifically tailored for X sentiment analysis remains a gap in the current research landscape. This project aims to address this gap by systematically evaluating the performance of different algorithms, including those mentioned above (Logistic Regression, Naive Bayes, Random Forests, LSTMs, and SVMs) and identifying the most suitable approach for sentiment analysis on X data. By incorporating sentiment analysis from prior research on platforms like Twitter (e.g., Gattan (2022) [4], Chaudhary et al. (2023) [1]), this project can leverage existing knowledge and tailor it to the specific characteristics of X.

Chapter 4

Requirements

The design of this project contains both hardware and software. The specifications are listed below.

4.1 Hardware

The experimental setup leveraged a cloud-based environment for model training. Hardware resources allocated for training included 29GB of RAM and a powerful NVIDIA P100 GPU with 16GB of dedicated memory. The P100 is a high-performance GPU specifically designed for demanding deep learning workloads. This combination ensured efficient processing of the training data and facilitated the model's learning.

4.2 Software

The training process was seamlessly executed within a Kaggle notebook, leveraging essential libraries such as TensorFlow, Keras, scikit-learn, numpy, and seaborn, alongside ipywidgets for interactive visualization and analysis.

This software environment provided a robust and efficient platform for developing and fine-tuning our custom deep learning model for Telugu script recognition.

4.3 Dataset

Researchers working on sentiment analysis have a valuable tool available on Kaggle: the Sentiment140 dataset. This public resource contains 1.6 million tweets extracted from Twitter using their API. Each tweet comes with a pre-assigned sentiment label (negative, neutral, or positive), making it ideal for training and testing sentiment analysis models. The dataset is offered in a comma-separated values (CSV) format and includes the following features for each tweet: a sentiment label, a unique tweet identifier, the date and time of creation, a flag indicating the query used for extraction (often blank), the username of the author, and the actual content of the tweet. The strengths of Sentiment140 lie in its substantial size (providing a wealth of training data), pre-labeled sentiments (streamlining training and evaluation), public availability, and the use of real-world tweets (reflecting the complexities of social media language).

Chapter 5

Proposed System

5.1 Methodology

This section gives a general description of the procedure that was followed in preparing the dataset for further processing, referring to Subsubsection 5.1.1, and gives detail on the main element of the approach: the Machine learning approaches using pre-trained models are discussed in Subsubsection 5.1.3. After this, the architecture of the Convolutional Neural Network (CNN), and Bidirectional Long Short-Term Memory(BiLSTM) refer to Subsubsection 5.1.2.

5.1.1 Data Acquisition and Preprocessing for training

The foundation of our study rests on a robust data collection and preprocessing strategy, aimed at capturing a representative sample of Twitter sentiments across varied topics. The dataset compilation process was thorough and methodical. This stage involves preparing the textual data from the tweets for sentiment analysis. Here, we will employ a combination of techniques to clean the data and extract meaningful features. To prepare the

textual data for analysis, we performed several preprocessing operations:

Noise Removal

We will remove elements from the tweets that generally do not contribute to sentiment analysis, such as URLs, emojis, special characters, and numbers.

Tokenization

Each tweet will be broken down into individual words or tokens. This simplifies manipulation of the text data for further processing.

Normalization

All text will be converted to lowercase for consistency. This avoids duplicates caused by case differences.

Stop Word Removal

Commonly used words in English that carry little inherent sentiment value, such as ”the,” ”is,” and ”at,” will be omitted from the analysis.

Stemming/Lemmatization

These techniques aim to reduce words to their base or root form (e.g., ”running” becomes ”run” and ”better” becomes ”good”). This further reduces the complexity of the dataset and improves consistency.

5.1.2 Feature Extraction

Effective feature extraction is crucial for sentiment analysis, as it directly influences the performance and accuracy of the classification models. We

will explore two primary approaches for feature extraction:

TF-IDF Vectorization

This technique is commonly used to represent documents (in our case, tweets) as numerical vectors suitable for use with machine learning algorithms. TF-IDF (Term Frequency-Inverse Document Frequency) considers two aspects: The frequency of a word within a specific tweet (term frequency). The overall importance of the word across the entire corpus of tweets (inverse document frequency). Combining the TF and IDF scores to create a TF-IDF weight for each word in each tweet. This weight reflects the word's significance within a specific tweet relative to its overall frequency in the corpus. Finally, each tweet is represented as a vector where each dimension corresponds to a word in the vocabulary. The value in a specific dimension represents the TF-IDF weight of the corresponding word in that particular tweet.

Word Embeddings

To better capture the semantic relationships between words and improve model sophistication, we employed pre-trained word embeddings from models like Word2Vec and GloVe. These models represent words in a continuous vector space where semantically similar words are positioned closer to one another, providing a richer and more expressive feature set for our models. In this work, we experimented with Word2Vec, which creates word embeddings by predicting the nearby words of a given word using a feed-forward neural network. Every word's embedding is effectively a one-dimensional vector of d values, where d is a parameter that the user enters.

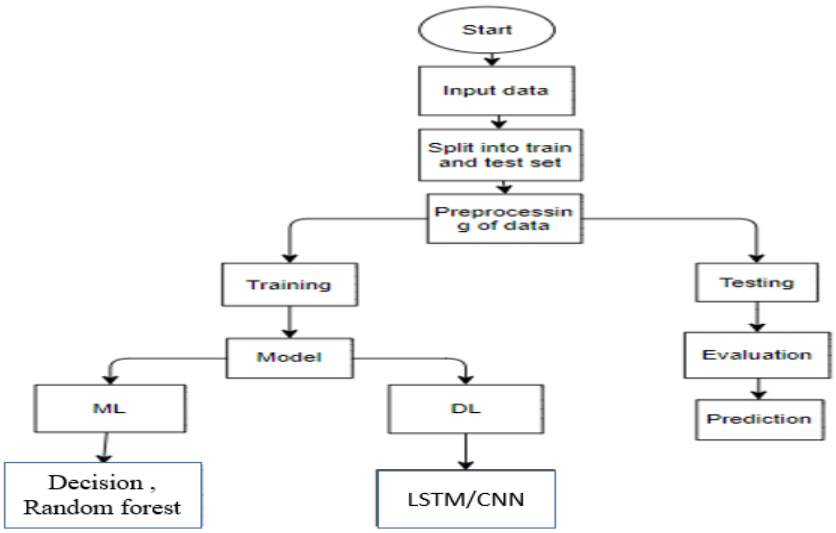

Figure 5.1

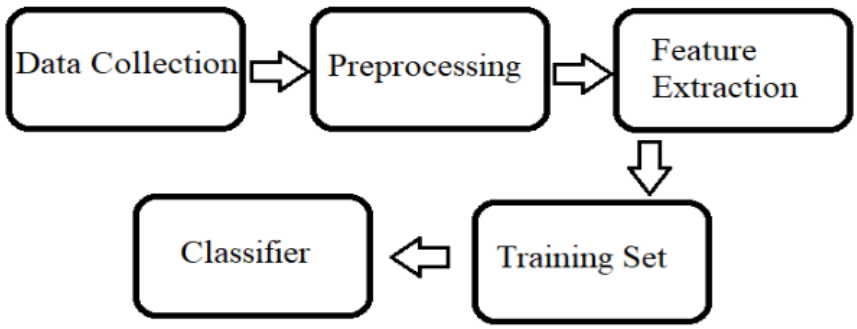

Figure 5.2: High-Level Diagram.

5.1.3 Model Selection and Training

We implemented a broad array of machine learning and deep learning models, each with distinct characteristics tailored to text-based sentiment analysis.

Machine Learning Algorithms

Logistic Regression: Binary categorization can be done statistically using logistic regression. It calculates, using an observation's features, the likelihood

that the observation belongs to a particular class. Consider a model that uses the sender address and keywords, among other variables, to determine the probability that an email is spam.

SVM: The SVM algorithm locates a hyperplane, or decision boundary, over a space that divides the classes according to their attributes. To lower the upper bound on the estimated generalization error, the objective is to maximize the margin, or establish the greatest distance between the separating hyperplanes. The answer is to map the inputs into a high-dimensional feature space for non-linearly separable data.

Random Forest: It is an ensemble learning method for regression and classification. The predictions from several decision trees, each concentrating on a different collection of features, are combined. In particular for complex data, the final prediction could be more accurate because it functions similarly to a vote in a classroom.

Naive Bayes: A probabilistic framework for resolving classification issues is provided by Naïve Bayes. Using the training data (corpus), Naïve Bayes first determines the likelihood of each unique word for each class as it appears in the corpus. The Naïve Bayes algorithm selects the class with the highest probability to classify a test record by multiplying the pre-calculated probabilities of each word in the test document.

Deep Learning Algorithms

Long Short-Term Memory (LSTM)

Our proposed model uses LSTM which is a special type of Recurrent Neural Network (RNN). In RNN, neurons are connected with each other in the form of directed cycle. The RNN model sequentially processes the information because it uses internal memory to process a sequence of words or inputs. RNN

performs the same task for each element because the output is dependent on all previous nodes inputs and remembered information. Our proposed model uses LSTM which is a special type of Recurrent Neural Network (RNN). In RNN, neurons are connected in the form of directed cycle. The RNN model sequentially processes the information because it uses internal memory to process a sequence of words or inputs. RNN performs the same task for each element because the output is dependent on all previous nodes inputs and remembers information.

Long Short-Term Memory (LSTM) is used to handle the vanishing gradient problem. The LSTM model saves long-term dependencies using three different gates in an effective way. The architecture of LSTM model is shown in Fig. 2. The structure of LSTM is chain like and it is similar to RNN, however, LSTM uses three gates to regulate and preserve information into every node state.

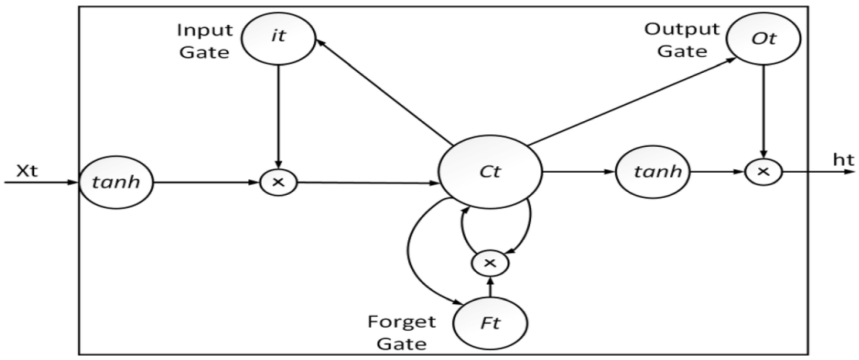

Figure 5.3: The Architecture of LSTM Model.

LSTM Layers

The architecture consists of an LSTM layer with 64 units and a dropout rate of 0.5. The LSTM layer processes the input sequence and captures sequential dependencies within the data. The output of the LSTM layer, denoted as

Output(l), is computed as follows:

$$\text{Output}(l) = \text{LSTM}(\text{Input}(l-1))$$

Where Input(l-1) represents the input feature map from the previous layer.

Activation Functions

Following the LSTM layer, ReLU activation functions are applied element-wise to introduce non-linearity into the network. ReLU enhances the model's ability to learn complex relationships between sequential features by outputting the input directly if positive, and zero otherwise. The ReLU function is defined as:

ReLU(x) = max(0, x)

Dropout Layers

Dropout layers have been conveniently embedded to a model to an extent that it was almost impossible to make the overfit. Given that these layers randomly exclude a certain fraction of neurons during each iteration of the training process, this, in turn, makes the model try to create features in a robust way that it would not depend on any specific features, hence improving generalization on new data. The output of the Dropout(x, p) function is obtained by applying dropout to the input feature map x as follows:

Dropout(x, p) = randomly drop neurons with probability p

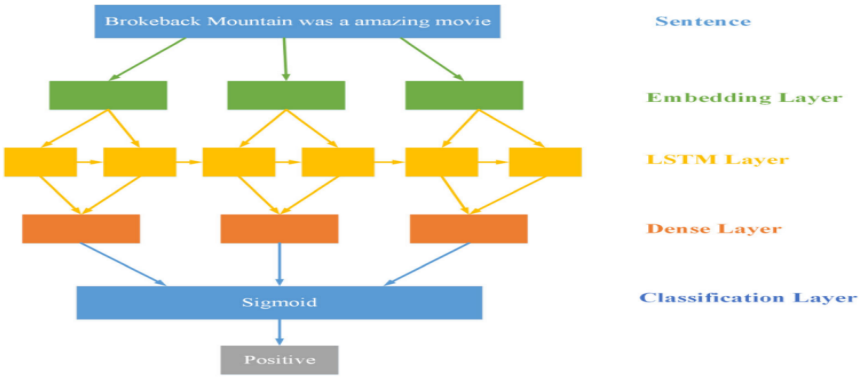

Figure 5.4: Proposed model using LSTM

Convolutional neural network (CNN)

The CNN is a special type of neural network and is employed from the field of image processing. However, CNN model has been effectively used in text classification. In CNN model, a subset of input to its preceding layers is connected using a convolutional layer that is why CNN layers are called feature map. The CNN model uses polling layer to reduce the computational complexity. The polling techniques in CNN reduce the output size of one stack layers to next in such a way that important information is preserved. There are many polling techniques available, however, max-polling is mostly used in which pooling window contains max value element. The flattened layer is used to feed the output of polling layer and maps it to next layers. The final layer in CNN typically is fully connected.

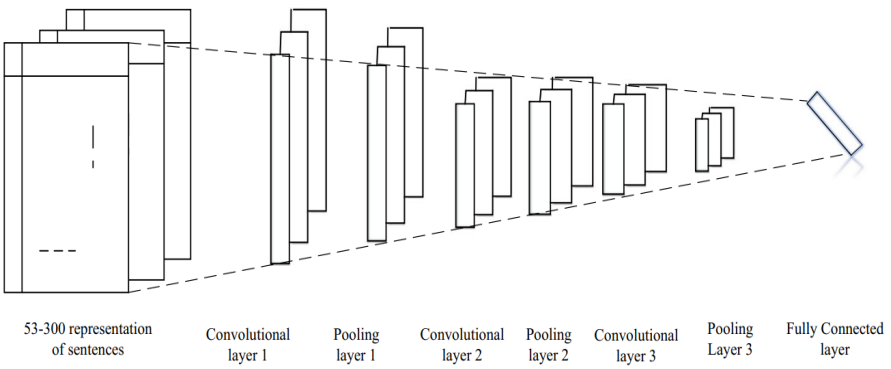

Figure 5.5: The Architecture of CNN Model.

Embedding layer

Pre-processed dataset provides a unique and meaningful sequence of words and every word has unique ID. Embedding layer initializes the words to assign random weights and it learns the embedding to embed all words in the training dataset. This layer is used in different ways and is mostly used to learn embedding of words that can be saved to use in another model. In this paper, we used pre-trained Word2Vc model for words embedding.

Convolutional Layers

The architecture consists of a series of convolutional layers, each containing a set of filters of different sizes to extract spatial information from the input images at different scales. The resulting feature map from the lth convolutional layer, called Output(l), is computed by convoluting the previous input feature map Input(l-1) with learnable filters of the current layer, Filters(l):

$$\text{Output}(l) = \text{Convolution}(\text{Input}(l-1), \text{Filters}(l))$$

This operation essentially slides the filters across the input feature map, performing element-wise mul- tiplication between corresponding elements and accumulating the products. The resulting feature map captures local spatial patterns within the image.

Global max-pooling

We applied global max-pooling at the end of network layers, it provides the global best results from the whole network after applying different convolution layers.

Activation Function

We use RELU activation function in our model. RELU gives zero at negative values and it increases with positive values.

Dense layer

Dense layer, also called fully connected layer, is used to perform classification on the extracted features of the convolutional layers. Using dense layer, every current input (neuron) in the layer of the network is connected to every input (neuron) in the proceeding layer of the network. The model is compiled with binary crossentropy loss function, Adam optimizer, and accuracy metric.

Sigmoid

Sigmoid is a function that is mostly used in the final layer of the neural network. It takes the average of the random results into 1 and 0 form. Figure 4 shows the proposed model using CNN.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 116)]             0

 embedding (Embedding)       (None, 116, 100)          52747100

 conv1d (Conv1D)             (None, 114, 32)           9632

 max_pooling1d (MaxPooling1  (None, 38, 32)            0
 D)

 conv1d_1 (Conv1D)           (None, 36, 64)            6208

 max_pooling1d_1 (MaxPoolin  (None, 12, 64)            0
 g1D)

 conv1d_2 (Conv1D)           (None, 10, 128)           24704

 global_max_pooling1d (Glob  (None, 128)               0
 alMaxPooling1D)

 dense (Dense)               (None, 5)                 645

=================================================================
Total params: 52788289 (201.37 MB)
Trainable params: 52788289 (201.37 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 5.6: CNN Model Summary.

Proposed Hybrid CNN-LSTM model

In Fig. 3, we show the main architecture of the proposed hybrid CNN-LSTM model. It takes a corpus as input and in the pre-processing phase, it performs sentence segmentation, tokenization, stop word removal and stemming tasks. After this, it applies word embedding layer using Word2Vec. The convolutional layer extracts the high-level features and LSTM layer detects long-term dependencies between words. In the end, we apply the classification layer using sigmoid function.

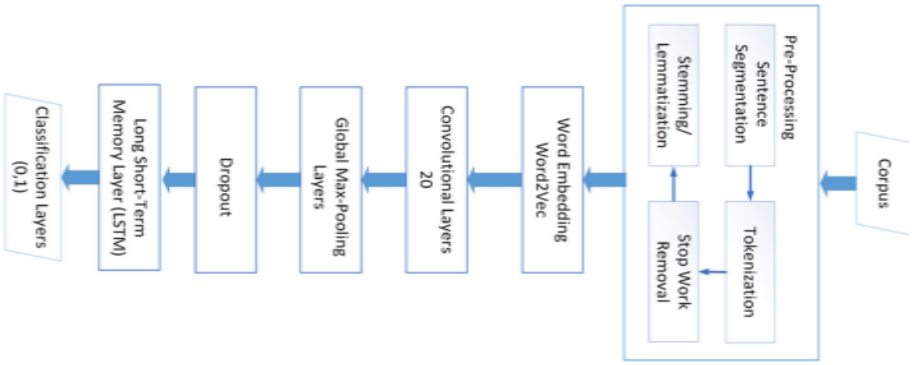

Figure 5.7: Methodology of Proposed CNN-LSTM Model.

Embedding Layer

An embedding layer is added to the model with input dimension 40000 and output dimension 10. The input$_l engthargumentisnotspecified, allowingthemodeltodynamicallyc$

Convolutional Layers

Five 1D convolutional layers are added to the model. Each convolutional layer has 32 filters, a kernel size of 3, 'same' padding, and ReLU activation function. After each convolutional layer, a MaxPooling1D layer with pool size 2 is added to reduce the dimensionality of the feature maps.

LSTM Layers

After the convolutional layers, an LSTM layer with 100 units, dropout rate of 0.2, and recurrent dropout rate of 0.2 is added. The LSTM layer captures temporal dependencies in the input sequence.

Dense Layers

A dense layer with 1 unit and a sigmoid activation function is added to perform binary classification. This layer outputs the probability of the positive class (sigmoid activation is used for binary classification).

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, None, 10)	400000
conv1d (Conv1D)	(None, None, 32)	992
max_pooling1d (MaxPooling1D)	(None, None, 32)	0
conv1d_1 (Conv1D)	(None, None, 32)	3104
max_pooling1d_1 (MaxPooling1D)	(None, None, 32)	0
conv1d_2 (Conv1D)	(None, None, 32)	3104
max_pooling1d_2 (MaxPooling1D)	(None, None, 32)	0
conv1d_3 (Conv1D)	(None, None, 32)	3104
max_pooling1d_3 (MaxPooling1D)	(None, None, 32)	0
conv1d_4 (Conv1D)	(None, None, 32)	3104
max_pooling1d_4 (MaxPooling1D)	(None, None, 32)	0
lstm_2 (LSTM)	(None, 100)	53200
dense_4 (Dense)	(None, 1)	101

Figure 5.8: CNN-LSTM Model Summary.

Bidirectional Long Short-Term Memory(BILSTM)

A development over the conventional one-way LSTM network is the BiLSTM network. It comprises of two separate LSTMs, one moving ahead and the other moving backward, allowing for feature extraction. The final network output is created by joining the two LSTMs' output vectors. This distinct neural network topology outperforms LSTM networks in terms of effectiveness and performance when extracting features from time series data

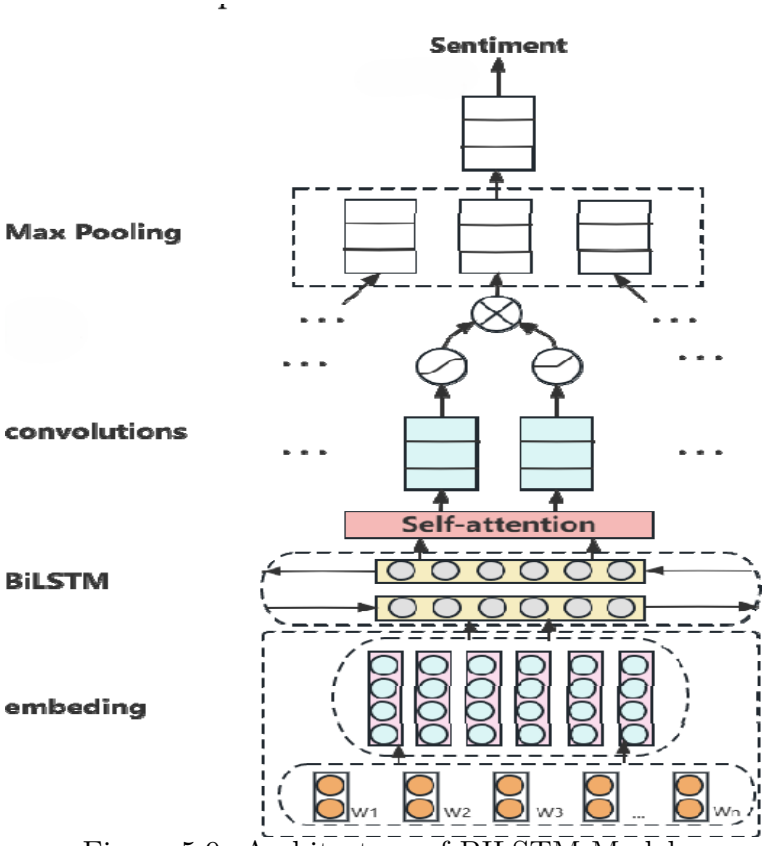

Figure 5.9: Architecture of BILSTM Model.

Bidirectional LSTM Layers

The architecture begins with two Bidirectional LSTM layers, each with 100 units and a dropout rate of 0.3. Bidirectional LSTM layers capture sequential dependencies in both forward and backward directions, enhancing the model's ability to understand context from both past and future inputs. Both LSTM layers return sequences (return$_s equences = True) to maintain sequence in formation throug$

Convolutional Layer

Convolutional Layer

Following the BiLSTM layers, a 1D convolutional layer with 100 filters and a kernel size of 5 is added. The convolutional layer applies filters over the

sequence representations learned by the BiLSTM layers, extracting higher-level features from the sequential input data.

Global Max Pooling Layer

A global max pooling layer is added to the model. This layer extracts the maximum value from each feature map across the entire sequence, capturing the most salient features from the convolutional layer's output.

Dense Layers

Two dense layers are added after the pooling layer. The first dense layer has 16 units with ReLU activation function, introducing non-linearity into the model and facilitating the learning of complex relationships between features. The second dense layer has 1 unit with sigmoid activation function, producing a binary output for sentiment classification.

Callbacks

Callbacks are defined to save the best model during training and stop training early if the validation loss does not improve for a specified number of epochs. Two callbacks are used: ModelCheckpoint to save the best model, and EarlyStopping to halt training if the validation loss does not improve for three consecutive epochs.

The model is then compiled with binary crossentropy loss function, Adam optimizer, and accuracy metric. It is trained using the fit method with the specified batch size, number of epochs, validation split, and callbacks.

```
Model: "Sentiment_Model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 100)         6000000

 bidirectional (Bidirection  (None, None, 200)         160800
 al)

 bidirectional_1 (Bidirecti  (None, None, 200)         240800
 onal)

 conv1d (Conv1D)             (None, None, 100)         100100

 global_max_pooling1d (Glob  (None, 100)               0
 alMaxPooling1D)

 dense (Dense)               (None, 16)                1616

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 6503333 (24.81 MB)
Trainable params: 503333 (1.92 MB)
Non-trainable params: 6000000 (22.89 MB)
_________________________________________________________________
```

Figure 5.10: BiLSTM Model Summary.

5.1.4 Hyper Parameter

- Batch Size: This parameter determines the number of training samples processed by the model during each update step.
- Learning Rate (η): This hyper-parameter controls the step size taken by the optimizer during weight updates. A smaller learning rate can lead to slower convergence.
- Optimizer Selection: Different optimizers, such as Adam or RMSprop, utilize various update rules to adjust model weights based on the gradients. The optimal choice can significantly impact convergence speed and performance.
- Dropout Rates (p): The probability of randomly dropping out neurons

during training.

Hyper-parameter	**Values**
Learning Rate (lr)	[0.001]
Optimizer	Categorical choice: Adam
Dropout Rate	Uniform: $[0.0, 0.5]$
Batch Size	Categorical: $[64, 128, 256, 1024]$
Epochs	Suggested: $[10, 12]$

Table 5.1: Hyper-parameters and Their Ranges

Chapter 6

Result and Analysis

6.1 Experimental Results

This section details the experiment's methodology, the experimental setup (refer Subsubsection 6.1.1), and the evaluation metrics employed (refer Subsubsection 6.1.1).

6.1.1 Experimental Setup

The experimental setup leveraged a cloud-based environment for model training. The training process was executed within a Kaggle notebook, providing a collaborative and reproducible platform. Hardware resources allocated for training included 29GB of RAM and a powerful NVIDIA P100 GPU with 16GB of dedicated memory. This combination ensured efficient processing of the training data and facilitated the model's learning. TensorFlow served as the backend engine, providing a robust and versatile framework for building, and training the model.

The performance of the model was evaluated using several metrics commonly employed in classification tasks. These metrics provide insights into

various aspects of the model's effectiveness:

- **Precision:** Precision reflects the proportion of positive predictions that were truly positive. It is calculated as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{6.1}$$

  Where,

  - True Positives (TP): Correctly classified positive cases.
  - False Positives (FP): Incorrectly classified negative cases as positive.

- **Recall:** Recall indicates the ability of the model to identify all relevant positive cases. It is calculated as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{6.2}$$

  Where,

  - True Positives (TP): Same as in Precision.
  - False Negatives (FN): Incorrectly classified positive cases as negative.

- **Accuracy:** Accuracy is the most straightforward metric, representing the overall correctness of the model's predictions:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Samples}} \tag{6.3}$$

  Where,

  - True Positives (TP): Same as in Precision and Recall.
  - True Negatives (TN): Correctly classified negative cases.
  - Total Samples: The total number of images or data points used for evaluation.

- **F1 Score:** F1 score addresses a potential limitation of accuracy, particularly in imbalanced datasets. It provides a harmonic mean of precision and recall, offering a balanced view:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6.4}$$

  Where,

  - Precision eq(6.1) and Recall eq(6.2) are as defined above.

6.2 Results and Discussion

This section provides the results that include visualizations of training accuracy, loss, and confusion matrix for ML and DL models - LSTM, CNN, CNN-LSTM and BILSTM are presented herein (section 6.2). Notably, the evaluation highlights the remarkable performance of the BILSTM, showcasing its superiority over both CNN-LSTM and CNN. According to Table 6.2, the DL models outperform the ML algorithms. Table 6.2 The results for the sentiment analysis task are shown below figures the deep learning models performed the best overall: BILSTM is the top performing model at 84% accuracy and 84% F1-macro, while CNN with 82% accuracy and F1-macro and CNN-LSTM follow with 79% accuracy and F1-macro of about 79%. It's important to note that all models were trained and evaluated using a predefined grid of hyperparameter values outlined in Table 5.1.

Model Name	Training Accuracy	Testing Accuracy	Loss	Learning rate	Optimizer	Dropout rate	Epochs	Batch Size
LSTM	83.23	76.10	0.3648	0.001	Adam	0.2983	10	256
CNN	84.57	81.77	0.4016	0.001	Adam	0.5000	5	128
CNN-LSTM	89.24	79.14	0.2495	0.001	Adam	0.2971	10	256
BiLSTM	84.02	84.15	0.3573	0.001	Adam	0.2883	12	256

Table 6.1: Model Performance Metrics Summary

Model Name	Accuracy	Precision	Recall	F1-Score
Logistic Regression	77.23	76.10	77	77
SVM	75.70	76.00	76	76
Random Forest	71.77	72	72	72
Naive Bayes	75.85	76.10	76	76
LSTM	76.10	76	76	76
CNN	81.77	82	82	82
CNN-LSTM	79.14	79	79	79
BiLSTM	84.15	84	84	84

Table 6.2: Model Performance Metrics Summary

- Logistic Regression showed commendable performance with an F1-score of 77%.
- Linear SVM and Naive Bayes each achieved an F1-score of 76%, displaying robustness despite the varied natures of these models.
- Random Forest algorithm had a slightly lower efficiency, realizing an F1-score of 72%.

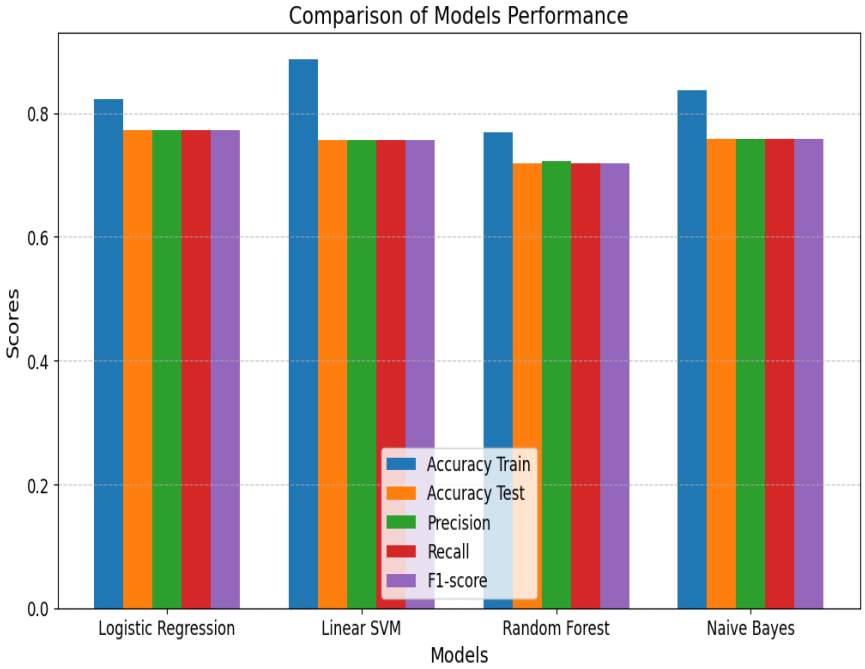

Figure 6.1: Performance Comparison of ML Algorithms.

- The LSTM model scored an F1-score of 76%, proving its capability in handling sequential data.
- Advanced models like the CNN achieved an F1-score of 82%, highlighting its prowess in local feature extraction from textual data.
- The CNN-LSTM hybrid model outperformed standalone models with an F1-score of 79%, benefiting from combined feature extraction and sequence learning.
- The highest performance was observed in the BILSTM with an F1-score of 84%, showcasing its excellence in bi-directional context understanding in text sequences.

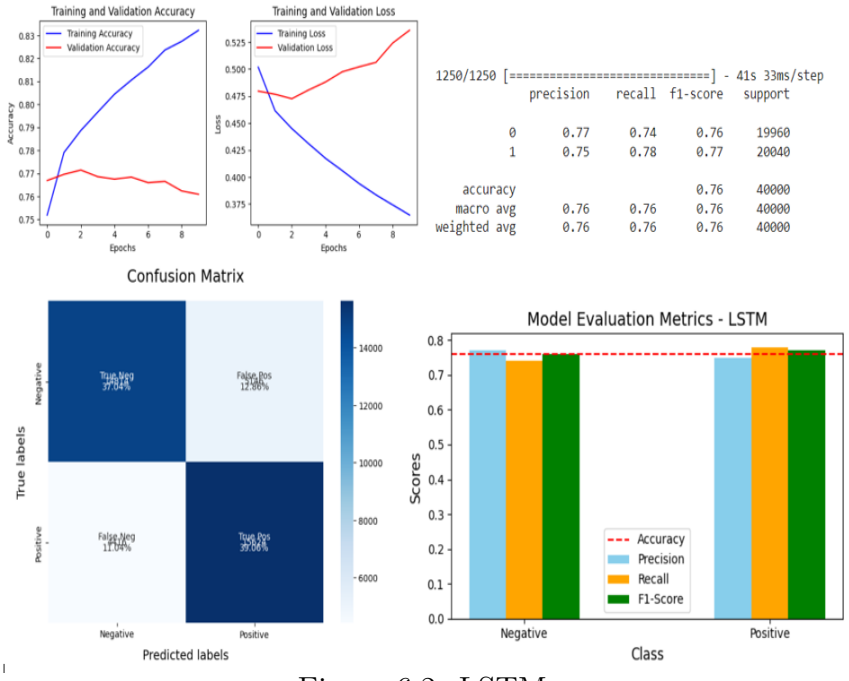

Figure 6.2: LSTM.

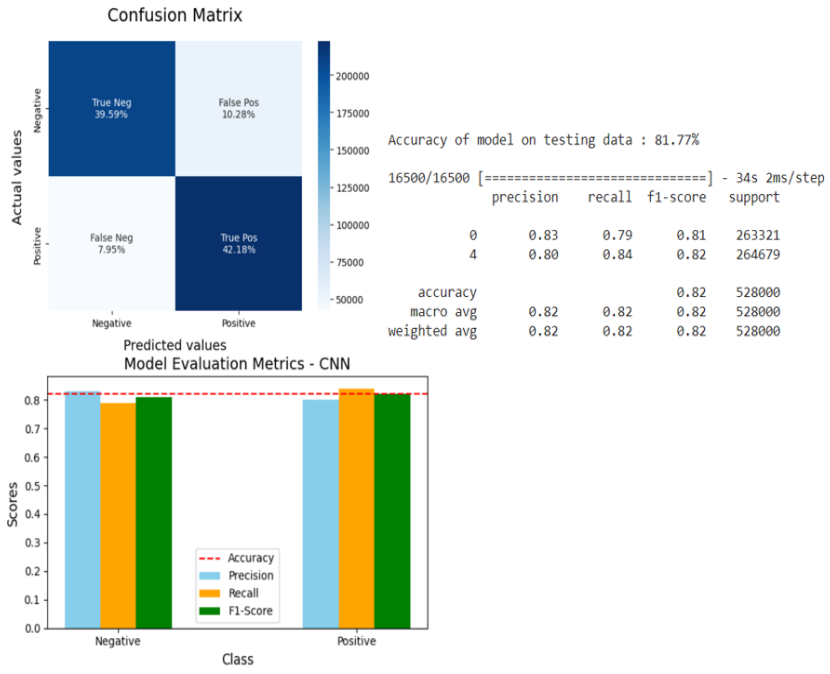

Figure 6.3: CNN.

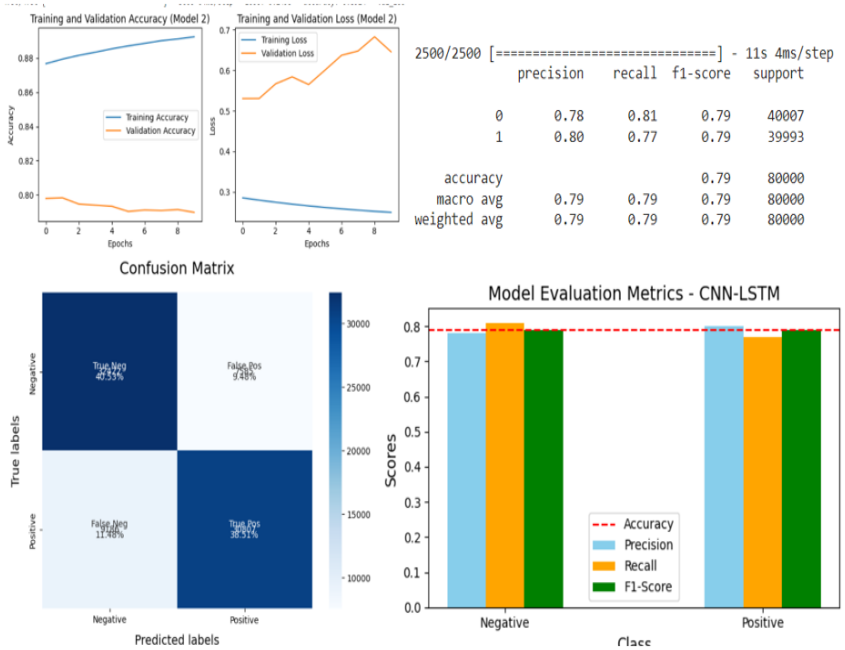

Figure 6.4: CNN-LSTM.

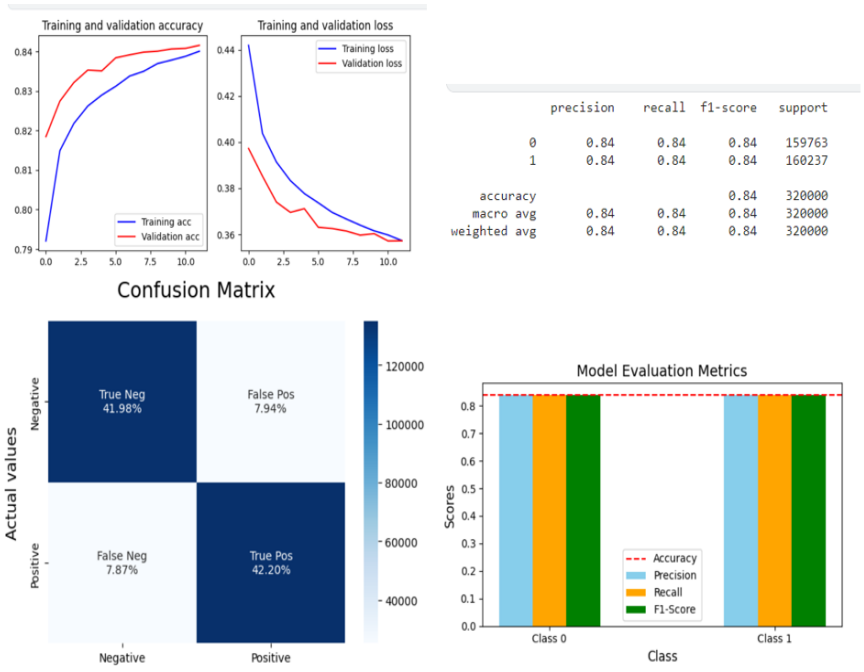

Figure 6.5: BiLSTM.

Chapter 7

Conclusion

In this comparative study of machine learning and deep learning approaches for sentiment analysis on Twitter data, our findings highlight significant insights. The deep learning algorithms, specifically BILSTM and CNN, demonstrated exceptional performance outshining the conventional machine learning models such as Logistic Regression, Linear SVM, and Naive Bayes. The superior ability of BILSTM and CNN models to process and analyze high-dimensional and sequential data makes them particularly well-suited for the intricacies of Twitter data, which is often noisy and highly contextual.

Our results also underscore the importance of pre-processing and feature extraction techniques in enhancing the performance of sentiment analysis models. Techniques like tokenization, normalization, and the use of sophisticated word embeddings contributed to the robustness of the models, allowing for more accurate sentiment prediction.

Moreover, the practical implications of our study are manifold. The high accuracy of sentiment classification achieved by the BILSTM, CNN-LSTM and CNN models can be leveraged by businesses for better understanding consumer sentiment, by policymakers for gauging public opinion on various

issues, and by researchers to further explore and enhance sentiment analysis techniques.

Future studies could expand on this work by exploring the integration of other novel deep learning architectures and by applying the models to real-time data streams for dynamic sentiment analysis. Furthermore, investigating the impact of multilingual datasets on model performance could enhance the generality and applicability of sentiment analysis tools across different demographic and cultural backgrounds.

In conclusion, this research not only provides a thorough evaluation of various sentiment analysis models but also sets the stage for future advancements in the field, promising enhanced capabilities for real-time analysis and interpretation of social media sentiments.

References

[1] Chaudhary, L., Girdhar, N., Sharma, D., Andreu-Perez, J., Doucet, A., Renz, M. (2023). A Review of Deep Learning Models for Twitter Sentiment Analysis: Challenges and Opportunities. IEEE Transactions on Computational Social Systems. https://ieeexplore.ieee.org/iel7/6570650/6780646/10298248.pdf

[2] Krishnan, H., Elayidom, M. S., Santhanakrishnan, T. (2022). A Comprehensive Survey on Sentiment Analysis in Twitter Data. Int. J. Distributed Syst. Technol., 13(1), 1-22. https://link.springer.com/article/10.1007/s13278-023-01030-x

[3] Huang, M., Yang, J. O., Zhang, Y. (2022). Deep Learning Approaches in Sentiment Analysis. 2022 2nd International Conference on Networking, Communications and Information Technology (NetCIT), 75-79. http://scis.scichina.com/en/2020/111102.pdf

[4] Gattan, A. M. (2022). Deep Learning Technique of Sentiment Analysis for Twitter Database. Int. J. Interact. Mob. Technol., 16. 13.https://e-space.mmu.ac.uk/626042/8/Sentiment%20Analysis%20of%20Tweets%20through%20Altmetrics%20-%20A%20Machine%20Learning%20Approach%201.pdf

[5] Alshammari, N. F., Almansour, A. (2019). State-of-the-art review on Twitter Sentiment Analysis. 2019 2nd International Conference on Computer Applications Information Security (ICCAIS), 1-8. https://dl.acm.org/doi/pdf/10.1145/3185045

[6] Mittal, A., Patidar, S. (2019). Sentiment Analysis on Twitter Data: A Survey. Proceedings of the 7th International Conference on Computer and Communications Management. https://link.springer.com/article/10.1007/s13278-023-01030-x

[7] Ramasamy, L. K., Kadry, S., Nam, Y. (2021). Performance analysis of sentiments in Twitter dataset using SVM models. International Journal of Electrical and Computer Engineering, 11(5), 2275-2284. https://e-space.mmu.ac.uk/626042/8/Sentiment%20Analysis%20of%20Tweets%20through%20Altmetrics%20-%20A%20Machine%20Learning%20Approach%201.pdf

[8] Ramneet, Gupta, D., Madhukar, M. (2020). Analysis of Machine Learning Approaches for Sentiment Analysis of Twitter Data. Journal of Computational and Theoretical Nanoscience. https://e-space.mmu.ac.uk/626042/8/Sentiment%20Analysis%20of%20Tweets%20through%20Altmetrics%20-%20A%20Machine%20Learning%20Approach%201.pdf

[9] Khalique, F., Hamdani, M. R., Masood, S., Chaudhry, B. B., Rauf, A. (2020). A Compendium of Classification Techniques, Tools and Evaluation Datasets for Twitter Sentiment Analysis. Journal of Intelligent Systems and Computing. https://www.irjweb.com/Twitter%20Sentimental%20Analysis%20with%20Rumor%20Elimination%20and%20Review%20Classification.pdf

[10] Agustiningsih, K. K., Utami, E., Al Fatta, H. (2021). Sentiment Analysis of COVID-19 Vaccine on Twitter Social Media: Systematic Literature Review. 2021 IEEE 5th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE), 121-126. https://www.mdpi.com/2076-393X/10/11/1929

[11] Jianqiang Zhao and Xiaolin Gui and Xuejun Zhang. 2018. Deep Convolution Neural Networks for Twitter Sentiment Analysis. IEEE Access, 6 (23253-23260). https://api.semanticscholar.org/CorpusID:44095154

[12] Ankit Tariyal and Sachin Goyal and Neeraj Tantububay. (2018).Sentiment Analysis of Tweets Using Various Machine Learning Techniques. International Conference on Advanced Computation and Telecommunication (ICACAT), 1-5. https://api.semanticscholar.org/CorpusID:209458402

[13] Vishu Tyagi and Ashwini Kumar and Sanjoy Das. (2020). Sentiment Analysis on Twitter Data Using Deep Learning approach. 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), 187-190. https://api.semanticscholar.org/CorpusID:232153643

[14] Marvin M. Agüero-Torales and José Ignacio Abreu Salas and Antonio Gabriel López-Herrera. (2021). Deep learning and multilingual sentiment analysis on social media data: An overview. Appl. Soft Comput, 107,107373. https://api.semanticscholar.org/CorpusID:233533982

[15] Zulfadzli Drus and Haliyana Khalid. (2019). Sentiment Analysis in Social Media and Its Application: Systematic Literature Review. Procedia Computer Science. https://api.semanticscholar.org/CorpusID:213629215

Appendix A

Source code

xcolor listings

A.1 Requirments

- numpy
- pandas
- opencv-python
- matplotlib
- scikit-learn
- tensorflow
- seaborn
- tqdm

A.2 DataLoder

```
# DataFrame
import pandas as pd

# plotting
import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# nltk
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# sklearn
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix,
   classification_report

from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split


```

```
#tensorflow
import tensorflow.compat.v2 as tf
import tensorflow_datasets as tfds
import tensorflow as tf

# Utility
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import re
import string
import pickle


from tensorflow.keras.preprocessing import sequence
from sklearn.datasets import fetch_20newsgroups
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import
    pad_sequences
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Dense,Input,Embedding,
    Dropout,Conv1D,MaxPooling1D,GlobalMaxPooling1D,Dropout,
    Bidirectional,Flatten,BatchNormalization,SimpleRNN,LSTM
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model

```

```
#!pip install keras

#from keras.models import Sequential
#from keras.layers import Embedding, Conv1D, MaxPooling1D,
   LSTM, Dense
#from keras.preprocessing.text import Tokenizer
#from keras.preprocessing.sequence import pad_sequences

# Construct a tf.data.Dataset
data = pd.read_csv('/kaggle/input/sentiment140/training
   .1600000.processed.noemoticon.csv',encoding='latin',
   names = ['polarity','id','date','query','user','text'])
```

A.3 Preprocessing

```
# Removing the unnecessary columns.
data.drop(['date','query','user','word count'], axis=1,
   inplace=True)

data.drop('id', axis=1, inplace=True)

#Checking if any null values present
(data.isnull().sum() / len(data))*100

#convrting pandas object to a string type
data['text'] = data['text'].astype('str')

nltk.download('stopwords')
```

```
stopword = set(stopwords.words('english'))
print(stopword)

nltk.download('punkt')
nltk.download('wordnet')

import re
import string

# Regular expressions for URL and username patterns
urlPattern = r"((http://)[^ ]*|(https://)[^ ]*|(www\.)[^ ]*)"
userPattern = '@[^\s]+'

# Stopwords list
stopwords = set(['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', '
```

```
    there', 'when', 'where', 'why', 'how', 'all', 'any', '
    both', 'each', 'few', 'more', 'most', 'other', 'some', '
    such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
    'than', 'too', 'very', 's', 't', 'can', 'will', 'just',
    'don', 'should', 'now', 'd', 'll', 'm', 'o', 're', 've',
     'y', 'ain', 'aren', 'couldn', 'didn', 'doesn', 'hadn',
    'hasn', 'haven', 'isn', 'ma', 'mightn', 'mustn', 'needn'
    , 'shan', 'shouldn', 'wasn', 'weren', 'won', 'wouldn'])

def lowercasing(tweet):
    return tweet.lower()

def remove_urls(tweet):
    return re.sub(urlPattern, '', tweet)

def remove_usernames(tweet):
    return re.sub(userPattern, '', tweet)

def remove_punctuations(tweet):
    return tweet.translate(str.maketrans("", "", string.
    punctuation))

def tokenize_words(tweet):
    return tweet.split()

def remove_stopwords(tokens):
    return [w for w in tokens if w not in stopwords]

def lemmatize_words(tokens):
```

```
    return tokens

def process_tweets(tweet):
    tweet = lowercasing(tweet)
    tweet = remove_urls(tweet)
    tweet = remove_usernames(tweet)
    tweet = remove_punctuations(tweet)
    tokens = tokenize_words(tweet)
    tokens = remove_stopwords(tokens)
    finalwords = lemmatize_words(tokens)
    return ' '.join(finalwords)

data['processed_tweets'] = data['text'].apply(lambda x:
   process_tweets(x))
print('Text Preprocessing complete.')
```

A.4 Model_LSTM

```
from keras.models import Sequential
from keras import layers
from keras import regularizers
from keras import backend as K
from keras.callbacks import ModelCheckpoint
model2 = Sequential()
model2.add(layers.Embedding(max_words, 128))
model2.add(layers.LSTM(64,dropout=0.5))
model2.add(layers.Dense(16, activation='relu'))
model2.add(layers.Dense(8, activation='relu'))
```

```
model2.add(layers.Dense(1,activation='sigmoid'))
model2.compile(optimizer='adam',loss='binary_crossentropy',
    metrics=['accuracy'])
#checkpoint2 = ModelCheckpoint("rnn_model.hdf5", monitor='
   val_accuracy', verbose=1,save_best_only=True, mode='auto
   ', period=1,save_weights_only=False)
checkpoint2 = ModelCheckpoint("rnn_model.hdf5", monitor='
   val_accuracy', verbose=1, save_best_only=True, mode='
   auto', period=1)
checkpoint2 = ModelCheckpoint("rnn_model.hdf5",
                              monitor='val_accuracy',
                              verbose=1,
                              save_best_only=True,
                              mode='auto')

model2.compile(optimizer='adam', loss='binary_crossentropy'
   , metrics=['accuracy'])

history = model2.fit(X_train, y_train, epochs=10,
   validation_data=(X_test, y_test), callbacks=[checkpoint2
   ])
```

A.5 Model_BiLSTM

```
from gensim.models import Word2Vec

Embedding_dimensions = 100

```

```
# Creating Word2Vec training dataset.
Word2vec_train_data = list(map(lambda x: x.split(), X_train
    ))

%%time

# Defining the model and training it.
word2vec_model = Word2Vec(Word2vec_train_data,
                  vector_size=Embedding_dimensions,
                  workers=8,
                  min_count=5)

print("Vocabulary Length:", len(word2vec_model.wv.
    key_to_index))

# Defining the model input length.
input_length = 60

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import
    pad_sequences

from tensorflow.keras.preprocessing.text import Tokenizer

# Assuming X_data is your list of texts
X_data = ["your", "list", "of", "texts", "here"]

vocab_length = 60000

```

```
tokenizer = Tokenizer(filters="", lower=False, oov_token="<oov>")
tokenizer.fit_on_texts(X_data)
tokenizer.num_words = vocab_length
print("Tokenizer vocab length:", vocab_length)

X_train_texts = [str(text) for text in X_train]
X_test_texts = [str(text) for text in X_test]

X_train = pad_sequences(tokenizer.texts_to_sequences(X_train_texts), maxlen=input_length)
X_test = pad_sequences(tokenizer.texts_to_sequences(X_test_texts), maxlen=input_length)

print("X_train.shape:", X_train.shape)
print("X_test.shape:", X_test.shape)

#creating embedding matrix shape
embedding_matrix = np.zeros((vocab_length, Embedding_dimensions))

for word, token in tokenizer.word_index.items():
    if word2vec_model.wv.__contains__(word):
        embedding_matrix[token] = word2vec_model.wv.__getitem__(word)

print("Embedding Matrix Shape:", embedding_matrix.shape)

from tensorflow.keras import Sequential
```

```
55 from tensorflow.keras.layers import Bidirectional,
   GlobalMaxPool1D, Dense, LSTM, Conv1D, Embedding
56
57 def getModel(embedding_matrix):
58     embedding_layer = Embedding(input_dim=vocab_length,
59                                 output_dim=
   Embedding_dimensions,
60                                 trainable=False)
61
62     embedding_layer.build((None,))
63     embedding_layer.set_weights([embedding_matrix])
64
65     model = Sequential([
66         embedding_layer,
67         Bidirectional(LSTM(100, dropout=0.3,
   return_sequences=True)),
68         Bidirectional(LSTM(100, dropout=0.3,
   return_sequences=True)),
69         Conv1D(100, 5, activation='relu'),
70         GlobalMaxPool1D(),
71         Dense(16, activation='relu'),
72         Dense(1, activation='sigmoid'),
73     ],
74     name="Sentiment_Model")
75     return model
```

A.6 Model_CNN-LSTM

```
1
2 model = Sequential()
3 model.add(Embedding(40000, 10))  # Remove input_length
    argument
4 model.add(Conv1D(filters=32, kernel_size=3, padding='same',
    activation='relu'))
5 model.add(MaxPooling1D(pool_size=2))
6 model.add(Conv1D(filters=32, kernel_size=3, padding='same',
    activation='relu'))
7 model.add(MaxPooling1D(pool_size=2))
8 model.add(Conv1D(filters=32, kernel_size=3, padding='same',
    activation='relu'))
9 model.add(MaxPooling1D(pool_size=2))
10 model.add(Conv1D(filters=32, kernel_size=3, padding='same',
    activation='relu'))
11 model.add(MaxPooling1D(pool_size=2))
12 model.add(Conv1D(filters=32, kernel_size=3, padding='same',
    activation='relu'))
13 model.add(MaxPooling1D(pool_size=2))
14 model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
15 model.add(Dense(1, activation='sigmoid'))
16 model.compile(loss='binary_crossentropy', optimizer='adam',
    metrics=['accuracy'])
17 model.summary()
18
19 import matplotlib.pyplot as plt
20
21 # Train the second model and collect history
22 history_model2 = model.fit(X_train, y_train,
```

```
    validation_data=(X_val, y_val), epochs=10, batch_size
    =256, verbose=1)

# Set figure size
plt.figure(figsize=(8, 4))

# Plotting Training and Validation Accuracy
plt.subplot(1, 2, 1)
plt.plot(history_model2.history['accuracy'], label='
    Training Accuracy')
plt.plot(history_model2.history['val_accuracy'], label='
    Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy (Model 2)')
plt.legend()

# Plotting Training and Validation Loss
plt.subplot(1, 2, 2)
plt.plot(history_model2.history['loss'], label='Training
    Loss')
plt.plot(history_model2.history['val_loss'], label='
    Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss (Model 2)')
plt.legend()

plt.tight_layout()  # Adjust layout to prevent overlap
```

```
plt.show()
```

A.7 Model_CNN

```

from tensorflow.keras.layers import Input,Conv1D,
   MaxPooling1D,Dense,GlobalMaxPooling1D,Embedding
from tensorflow.keras.models import Model

D=100
i=Input((T,))
x=Embedding(V+1,D)(i)
x=Conv1D(32,3,activation='relu')(x)
x=MaxPooling1D(3)(x)
x=Conv1D(64,3,activation='relu')(x)
x=MaxPooling1D(3)(x)
x=Conv1D(128,3,activation='relu')(x)
x=GlobalMaxPooling1D()(x)
x=Dense(5,activation='softmax')(x)
model=Model(i,x)
model.summary()
```

A.8 Training

```
training_model = getModel(embedding_matrix)
training_model.summary()
import tensorflow as tf
```

```
from tensorflow.keras.callbacks import ModelCheckpoint,
    EarlyStopping

# Define callbacks
callbacks = [
    ModelCheckpoint("best_model.keras", save_best_only=True
    ),
    EarlyStopping(patience=3, monitor='val_loss',
    restore_best_weights=True)
]

training_model.compile(loss='binary_crossentropy',
    optimizer='adam', metrics=['accuracy'])


history_model3 = training_model.fit(
    X_train, y_train,
    batch_size=1024,
    epochs=12,
    validation_split=0.1,
    callbacks=callbacks,
    verbose=1,
)
```

A.9 Plot_boxing

```
acc,  val_acc  = history.history['accuracy'], history.
    history['val_accuracy']
```

```
loss, val_loss = history.history['loss'], history.history['
    val_loss']
epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()

from sklearn.metrics import confusion_matrix,
    classification_report

def ConfusionMatrix(y_pred, y_test):
    # Compute and plot the Confusion matrix
    cf_matrix = confusion_matrix(y_test, y_pred)

    categories  = ['Negative','Positive']
    group_names = ['True Neg','False Pos', 'False Neg','
    True Pos']
    group_percentages = ['{0:.2%}'.format(value) for value
```

```
    in cf_matrix.flatten() / np.sum(cf_matrix)]
28
29      labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,
    group_percentages)]
30      labels = np.asarray(labels).reshape(2,2)
31
32      sns.heatmap(cf_matrix, annot = labels, cmap = 'Blues',
    fmt = '',
33                  xticklabels = categories, yticklabels =
    categories)
34
35      plt.xlabel("Predicted values", fontdict = {'size':14},
    labelpad = 10)
36      plt.ylabel("Actual values"   , fontdict = {'size':14},
    labelpad = 10)
37      plt.title ("Confusion Matrix", fontdict = {'size':18},
    pad = 20)
38
39  # Predicting on the Test dataset.
40  y_pred = training_model.predict(X_test)
41
42  # Converting prediction to reflect the sentiment predicted.
43  y_pred = np.where(y_pred>=0.5, 1, 0)
44
45  # Printing out the Evaluation metrics.
46  ConfusionMatrix(y_pred, y_test)
47
48  # Print the evaluation metrics for the dataset.
49  print(classification_report(y_test, y_pred))
```

A.10 Predicting

```
sequence = tokenizer.texts_to_sequences(['this data science
    article is the worst ever'])
test = pad_sequences(sequence, maxlen=max_len)
pred = model2.predict(test)
if pred > 0.5:
  print('Positive')
else:
  print('Negative')
# print(pred)

import pickle

file = open('vectoriser.pickle','wb')
pickle.dump(vector, file)
file.close()

file = open('logisticRegression.pickle','wb')
pickle.dump(lg, file)
file.close()

file = open('SVM.pickle','wb')
pickle.dump(svm, file)
file.close()

file = open('RandomForest.pickle','wb')
pickle.dump(rf, file)
file.close()
```

```
27
28 file = open('NaivesBayes.pickle','wb')
29 pickle.dump(nb, file)
30 file.close()
31
32 def load_models():
33     # Load the vectoriser.
34     file = open('vectoriser.pickle', 'rb')
35     vectoriser = pickle.load(file)
36     file.close()
37     # Load the LR Model.
38     file = open('logisticRegression.pickle', 'rb')
39     lg = pickle.load(file)
40     file.close()
41     return vectoriser, lg
42
43 def predict(vectoriser, model, text):
44     # Predict the sentiment
45     processes_text=[process_tweets(sen) for sen in text]
46     textdata = vectoriser.transform(processes_text)
47     sentiment = model.predict(textdata)
48
49     # Make a list of text with sentiment.
50     data = []
51     for text, pred in zip(text, sentiment):
52         data.append((text,pred))
53     # Convert the list into a Pandas DataFrame.
54     df = pd.DataFrame(data, columns = ['text','sentiment'])
55     df = df.replace([0,1], ["Negative","Positive"])
```

```
56      return df
57
58  if __name__=="__main__":
59      # Loading the models.
60      vectoriser, lg = load_models()
61
62      # Text to classify should be in a list.
63      text = ["I love machine learning",
64              "Work is too hectic.",
65              "Mr.Sharama, I feel so good"]
66
67      df = predict(vectoriser, lg, text)
68      print(df.head())
```