Journal of Artificial Intelligence and Computing (JAIC)
ISSN (e): 3005-3358
Volume: 2, Number: 1, Pages: 1- 8, Year: 2023

Comparative Analysis of Logistic Regression, LSTM, and Bi-LSTM Models for Sentiment Analysis on IMDB Movie Reviews

Mureed Hussain, Mudasser Naseer

Computer Science and Information Technology Department, University of Lahore, Lahore, 54000, Pakistan
Corresponding author: Mureed Hussain(Email: mureedsargana@gmail.com, mudasser.naseer@cs.uol.edu.pk)

Received: 12/01/2024, Revised: 22/04/2024, Accepted: 25/06/2024

***Abstract*—This study examines the efficacy of three distinct machine learning models for sentiment analysis: Bidirectional Long Short-Term Memory (Bi-LSTM), Long Short-Term Memory (LSTM), and Logistic Regression. One of the most important tasks in natural language processing is sentiment analysis, which entails categorizing reviews as positive or negative. Fifty thousand sentiment-labeled movie reviews make up the dataset. To determine the most precise and effective technique for sentiment categorization, we put these models into practice and evaluated their respective performances. The highest accuracy of 89.42% is achieved by Logistic Regression, with precision of 88.35%, recall of 91.01%, and F1-score of 89.66%. The LSTM model, well-known for capturing temporal dependencies in text, obtained an accuracy of 86.23%, precision of 89.60%, recall of 82.22%, and an F1-score of 85.75%. The Bi-LSTM model, which processes input sequences in both forward and backward directions to better capture context, demonstrated accuracy of 87.65%, precision of 88.67%, recall of 86.54%, and an F1-score of 87.60%. The results show that although Logistic Regression performs better in accuracy, the Bi-LSTM model offers a substantial trade-off between precision and recall, making it a viable option for sentiment analysis applications. Despite its minor accuracy lag, the LSTM model provides important insights into the sequential relationships of the text data.**

***Index Terms*—LSTM, Bi-LSTM, Logistic Regression, NLP.**

I. Introduction

THIS Sentiment analysis, or opinion mining, is an important branch of natural language processing (NLP) that deals with recognizing and classifying sentiments conveyed in text to elucidate the writer's perspective on a given subject. As user-generated material on the internet continues to develop exponentially, sentiment analysis is becoming more and more crucial for various applications, such as social media monitoring, market analysis, and consumer feedback. This study explores the field of sentiment analysis using the 50,000+ movie reviews from IMDB that have been classified as either positive or negative. The main objective is to assess and compare the efficiency of various machine learning models in classifying these sentiments accurately. First, we start with the popular and effective approach for binary classification tasks, known as logistic regression. Because Logistic Regression can handle huge feature spaces efficiently, it is a powerful baseline even though it is simple. Next, we investigate the potential of recurrent neural networks (RNNs) with the Long Short-Term Memory (LSTM) model, which is particularly well-suited to representing temporal relationships in sequential data. Because LSTMs can preserve long-term dependencies—a critical skill for comprehending sentence context—they are especially well-suited for handling text data. We also look into Bidirectional Long Short-Term Memory (Bi-LSTM) networks, which can process input sequences forward and backward, extending the capabilities of ordinary LSTMs. Due to its bidirectional approach, Bi-LSTMs are especially useful for tasks where comprehending the entire sequence context is crucial since they can capture context more thoroughly. In addition to comparing various models, this study sheds light on each one's benefits and drawbacks regarding sentiment analysis. This research seeks to influence future work in NLP and sentiment analysis by comparing classic and advanced machine learning algorithms and utilizing a real-world dataset and robust assessment measures. The results highlight how crucial it is to choose a model according to the particular needs of the sentiment analysis activity, adding significant information to the field.

II. Literature Review

Numerous deep learning and machine learning models have been used to investigate the sentiment analysis of movie reviews in great detail. Ali et al. [1] developed a sentiment analysis classifier for the IMDB dataset using deep learning models. Qaisar used long short-term memory (LSTM) networks [2]. Using the IMDB dataset, Amulya et al. [3] evaluated deep

learning and machine learning techniques for sentiment analysis. Sentiment analysis tasks have also made use of Bidirectional LSTMs, or Bi-LSTMs. Minaee et al. [08] employed an ensemble of CNN and Bi-LSTM networks for sentiment analysis. Vimali and Murugan [5] concentrated on the output of Bi-LSTM structures to identify the most effective sentiment analysis technique. A CNN model for brief text sentiment analysis based on Bi-LSTM self-attention was suggested by Bhuvaneshwari et al. in [6]. Because LSTMs can identify sequential relationships in text data, they are frequently utilised in sentiment analysis. A regional CNN-LSTM model for dimensional sentiment analysis was presented by Wang et al. [7]. An attention-emotion-enhanced convolutional long short-term memory for sentiment analysis was presented by Huang et al. in [8]. Using LSTMs, Murthy et al. [9] conducted sentiment analysis on text reviews. Instead of using a bidirectional LSTM model for sentiment analysis in large social data, Behera et al. [10] used a standard LSTM model. Sentiment analysis has also made use of logistic regression. Tyagi and Sharma [11] conducted sentiment analysis using Logistic Regression and a useful word score heuristic. Multinomial Logistic Regression was selected by Ramadhan et al. [12] for sentiment analysis because of its competitive performance in terms of CPU and memory usage. Prabhat and Khullar evaluated Naïve Bayes and Logistic Regression [13] for sentiment categorization on large datasets. Gradient Descent Classifier and Logistic Regression were the machine learning techniques Aliman et al. used to determine the optimal sentiment analysis method. In their work on enhancing lexicons for sentiment analysis, Bhargava and Katarya [14] employed logistic regression to predict a logit change in the likelihood of quality of the norm for interest.

III. Problem Statement

An enormous quantity of data has been created by the growth of user-generated material on websites like movie review sites, which must be processed and examined to derive valuable insights. Using textual data to reliably infer the sentiment of movie reviews—which can be either favourable or negative—is a major difficulty. For enormous datasets, traditional methods like manual review are time-consuming and unfeasible. As a result, automated sentiment analysis methods that can quickly and precisely categorize the sentiment of movie reviews are desperately needed. The selection of a suitable machine learning model for this task is still crucial, though, as various models have varying capacities for managing textual input and capturing the subtleties of human language.

IV. Dataset

The IMDB movie reviews dataset, which includes 50,000 reviews divided into positive and negative, was utilised for this study. Every review is a written contribution made by people that shares their thoughts and assessments on different films. Because there are an equal number of positive and negative ratings in the dataset, it is balanced and are used to train unbiased models. We first cleaned the text, converted it to lowercase, removed special characters, and removed common stop words that don't add much to the sentiment analysis to get the data ready for analysis. After cleaning the text, we tokenized it to separate the words and sentences. We utilized TF-IDF (Term Frequency-Inverse Document Frequency) vectorization for the Logistic Regression model to transform the textual data into numerical features that indicate the significance of individual phrases in the reviews. To achieve consistent input length for the neural networks in the LSTM and Bi-LSTM models, we first employed sequence padding after capturing the semantic meaning of words using word embedding. After that, the dataset was divided into training and testing sets, with 20% set aside for testing and 80% utilised to train the models. This division makes it possible to assess the models' performance on hypothetical data with reliability, guaranteeing the generalizability of the findings. The data that had been cleaned, tokenized, and vectorized offered a strong basis on which to train the machine learning models and assess how well they classified sentiment. This thorough data preparation procedure is essential to getting sentiment analysis results that are accurate and trustworthy.

V. Methodology

Our study systematically employs three machine learning models—Logistic Regression, LSTM, and Bi-LSTM—to perform sentiment analysis on the IMDB movie reviews dataset. The methodology comprises several key steps, each critical for the overall process. Here, we detail each step comprehensively.

A. Data Acquisition:

The IMDB movie reviews dataset, which has 50,000 reviews classified as positive or negative, was used for this investigation. It was downloaded and put into a pandas Data Frame to process the dataset further.

B. Data Inspection and Cleaning:

Examining the dataset to comprehend its contents and structure is the first step. We looked at the initial few entries and the distribution of positive and negative evaluations to ensure the dataset was balanced.

Next, all characters in the text data were changed to lowercase, special characters were deleted, and frequently occurring stop words were removed. This step reduces the noise in the evaluations, and the most important words are highlighted.

C. Text Tokenization and Vectorization:

Using the TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer, the cleaned text data was transformed into numerical features for the Logistic Regression model. This technique aids in determining the significance of the words used in the reviews.

Shape of TF-IDF matrix: (50000, 10000).

First 20 feature names (words): ['007' '010' '10' '100' '1000' '101' '1010' '10br' '11' '110' '12' '13'
 '13th' '14' '15' '150' '16' '17' '18' '18th']

```
Mounted at /content/drive
First few records of the dataset:
                                              review sentiment
0  One of the other reviewers has mentioned that ...  positive
1  A wonderful little production. <br /><br />The...  positive
2  I thought this was a wonderful way to spend ti...  positive
3  Basically there's a family where a little boy ...  negative
4  Petter Mattei's "Love in the Time of Money" is...  positive

Distribution of reviews:
sentiment
positive    25000
negative    25000
Name: count, dtype: int64
```

Figure 1: Distribution of reviews

The number of features (words) used for modelling: 10000. The text was tokenized to make understanding the LSTM (Long Short-Term Memory) and Bi-LSTM (Bidirectional LSTM) models easier. After that, these sequences were padded to guarantee that their length was consistent, which allowed the neural network models to use them as input.

D. Data Splitting:

Eighty percent of the dataset was put aside for testing, while the remaining twenty percent was used for training. This division guarantees that the models are assessed using unseen data, offering a reliable measure of their efficacy.

E. Model Training:

Logistic Regression: We initialized and trained a Logistic Regression model using the TF-IDF vectorized data. The model was trained to classify reviews based on the extracted features.
LSTM: Tensor Flow/Keras was used to create an LSTM model. The model design includes an embedding layer, an LSTM layer with dropout for regularization, and a dense output layer with a sigmoid activation function. The tokenized and padded sequences were used to train the model.
Bi-LSTM: A bidirectional LSTM model was built using bidirectional layers instead of linear layers. The model's comprehension of the context in the reviews is enhanced by this design, which enables it to capture dependencies in both forward and backward directions.

F. Model Evaluation:

Several measures were used to assess the trained models on the testing set, including accuracy, precision, recall, F1-score, and confusion matrix. The ROC (Receiver Operating Characteristic) curve and AUC (Area Under the Curve) were presented to illustrate the models' performance further.
Predictions for the Logistic Regression model was made using the TF-IDF vectorized test data. The tokenized and padded sequences of the test data were utilised for the LSTM and Bi-LSTM models.

VI. Result Analysis

Several measures, including accuracy, precision, recall, F1-score, and confusion matrix, were used to assess the performance of the three models: Bi-LSTM, LSTM, and logistic regression. These metrics thoroughly explain how well each model can categorize movie reviews as positive or negative.

A. Logistic Regression:

Out of the three models, the Logistic Regression model performed the best. It had the highest accuracy. The model's high recall implies that it properly detects a significant percentage of all positive reviews, while its high accuracy shows that it can effectively identify positive reviews. The balanced F1-score emphasizes the model's ability to handle both positive and negative feedback reliably.

Accuracy	0.8942
Precision	0.8834521286842613
Recall	0.9101012105576504
F1-score	0.8965786901270774

Figure 2:Evaluation Metrics

	precision	recall	f1-score	support
0	0.91	0.88	0.89	4961
1	0.88	0.91	0.90	5039
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
weighted avg	0.89	0.89	0.89	10000

Figure 3:Classification Report

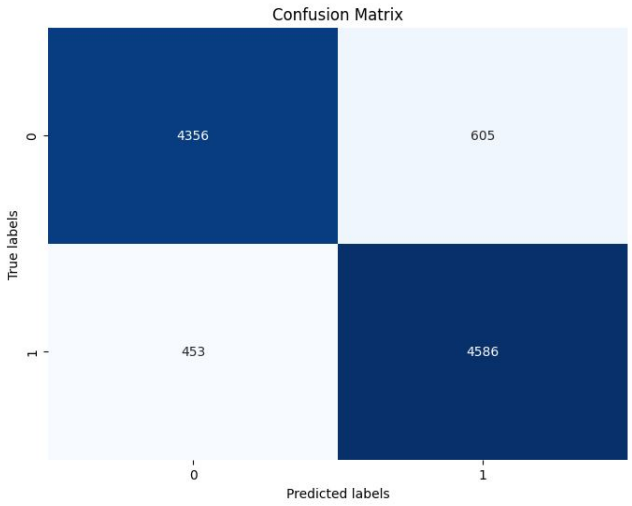

Figure 4;Confusion Matrix

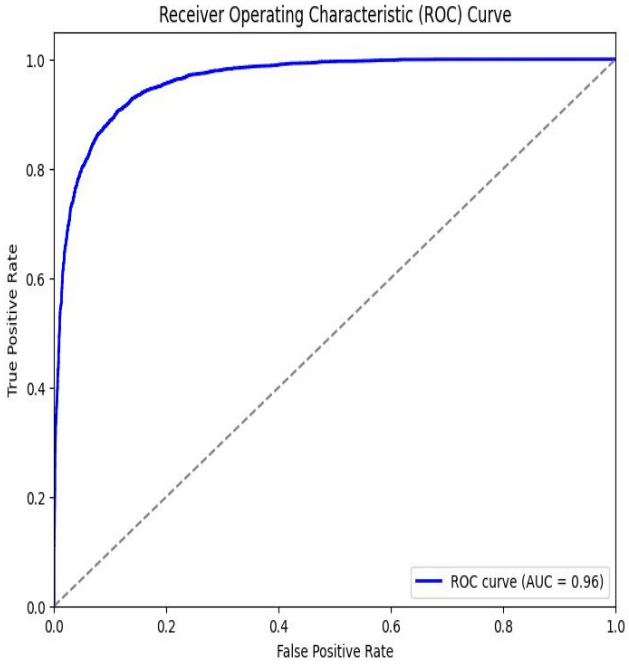

Figure 5:ROC Curve

B. *LSTM:*

Though it performed admirably, the LSTM model, created to capture temporal relationships in text data, trailed the Logistic Regression model by a small amount. The accuracy was quite high, suggesting that favourable evaluations were successfully identified. However, recall was lower than logistic regression, indicating that finding every good review might have been challenging. Despite its strength, the F1-score demonstrated this trade-off between recall and accuracy.

Accuracy	0.8623
Precision	0.8959775086505191
Recall	0.8221869418535424
F1-score	0.8574976715305805

Figure 6: Evaluation Matrix

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.90	0.87	4961
1	0.90	0.82	0.86	5039
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

Figure 7: Classification Report

Confusion Matrix:
[[4480 481]
[896 4143]]

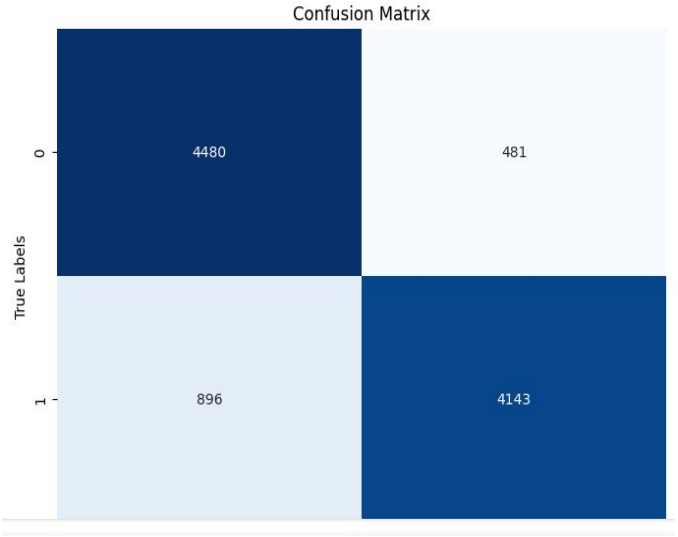

Figure 8:Confusion Matrix

C. *Bi-LSTM:*

The performance of the Bi-LSTM model outperformed the LSTM model as it captures dependencies in both forward and backward directions. It closely followed the Logistic Regression model, with an accuracy of 87.65%. In comparison to the LSTM, the Bi-LSTM produced a better F1 score because of its more balanced accuracy and recall. This suggests that the Bi-LSTM model gains from its enhanced context understanding by processing the text in both directions.

Accuracy	0.8765
Precision	0.8867425782838553
Recall	0.8654494939472117
F1-score	0.875966656623481

Figure 9: Evaluation Matrix

D. *Confusion Matrices:*

The confusion matrices for each model provided further insights into their performance. Logistic Regression had the fewest misclassifications, followed by Bi-LSTM and then LSTM. This aligns with the overall accuracy and other evaluation metrics.

VII. Conclusion

Overall, while all three models demonstrated strong performance in sentiment analysis of movie reviews, the Logistic Regression model slightly outperformed the LSTM and Bi-LSTM models. However, the deep learning models (LSTM and Bi-LSTM) showed considerable promise, particularly in capturing more complex patterns in text data. The choice between these models may depend on specific application requirements, such as interpretability (favouring Logistic Regression) or the ability to capture intricate dependencies (favouring LSTM or Bi-LSTM).

APPENDIX

Dataset Downloaded from the given link below:
IMDB Dataset of 50K Movie Reviews (kaggle.com)

A. Code for Data Exploration task:

```
# Step 1: Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Step 2: Load and Inspect the Dataset
import pandas as pd

# Define the file path
file_path = "/content/drive/My Drive/IMDB Dataset.csv"

# Load the dataset
df = pd.read_csv(file_path)

# Display the first few records to inspect the dataset
print("First few records of the dataset:")
print(df.head())

# Display the distribution of positive and negative reviews
print("\nDistribution of reviews:")
print(df['sentiment'].value_counts())
```

B. Code for Data Cleaning:

```
# Step 1: Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Step 2: Clean the Data
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Ensure you have the necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')

# Define the file path
file_path = "/content/drive/My Drive/IMDB Dataset.csv"

# Load the dataset
df = pd.read_csv(file_path)

# Display the first few records to inspect the dataset
print("First few records of the dataset before cleaning:")
print(df.head())

# Function to clean the text
def clean_text(text):
    # Convert text to lowercase
    text = text.lower()
    # Remove special characters
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    # Tokenize text
    words = word_tokenize(text)
    # Remove stop words
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in
stop_words]
    # Join the cleaned words back into a single string
    cleaned_text = ' '.join(words)
    return cleaned_text

# Apply the clean_text function to the review column
df['cleaned_review'] = df['review'].apply(clean_text)

# Display the first few records after cleaning
print("\nFirst few records of the dataset after cleaning:")
print(df.head())

# Save the cleaned dataset to a new CSV file
cleaned_file_path = "/content/drive/My Drive/IMDB Dataset
Cleaned.csv"
df.to_csv(cleaned_file_path, index=False)

print("\nCleaned dataset saved to:", cleaned_file_path)
```

C. Code for Text Processing and feature extraction

```
# Step 1: Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Step 2: Convert Text to Numerical Format using TF-IDF
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

# Define the file path for the cleaned dataset
file_path = "/content/drive/My Drive/IMDB Dataset
Cleaned.csv"

# Load the cleaned dataset
df = pd.read_csv(file_path)

# Inspect the dataset
print("First few records of the cleaned dataset:")
print(df.head())

# Prepare the text data for TF-IDF transformation
texts = df['cleaned_review'].tolist()

# Initialize the TF-IDF Vectorizer
# max_features determines the number of words to use
max_features = 10000  # You can adjust this number based on
your needs
tfidf_vectorizer =
TfidfVectorizer(max_features=max_features)

# Fit and transform the texts using TF-IDF
X = tfidf_vectorizer.fit_transform(texts)

# Convert the TF-IDF matrix to a dense format
X_dense = X.toarray()

# Display the shape of the data
```

```
print("\nShape of TF-IDF matrix:", X_dense.shape)

# Display the feature names (words)
feature_names = tfidf_vectorizer.get_feature_names_out()
print("\nFirst 20 feature names (words):", feature_names[:20])

# Determine the number of features
num_features = len(feature_names)
print("\nNumber of features (words) used for modeling:",
num_features)
```

D. Code for Logistic Regression Model Training, Testing and Evaluating:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, classification_report,
confusion_matrix, roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns

# Define the file path for the cleaned dataset
file_path = "/content/drive/My Drive/IMDB Dataset
Cleaned.csv"

# Load the cleaned dataset
df = pd.read_csv(file_path)

# Prepare the text data for TF-IDF transformation
texts = df['cleaned_review'].tolist()

# Initialize the TF-IDF Vectorizer
max_features = 10000  # Number of words to use
tfidf_vectorizer =
TfidfVectorizer(max_features=max_features)

# Fit and transform the texts using TF-IDF
X = tfidf_vectorizer.fit_transform(texts)

# Convert the TF-IDF matrix to a dense format
X_dense = X.toarray()

# Convert sentiments to binary labels
y = df['sentiment'].apply(lambda x: 1 if x == 'positive' else
0).values

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_dense, y,
test_size=0.2, random_state=42)

# Initialize the Logistic Regression model
model = LogisticRegression(max_iter=1000)

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print("Evaluation Metrics:")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-score: {f1}")
print("\nClassification Report:")
print(classification_rep)

# Plot confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',
cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

# Plot ROC curve and calculate AUC
y_pred_proba = model.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
auc = roc_auc_score(y_test, y_pred_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC =
%0.2f)' % auc)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

E. Code for LSTM Model Training, Testing and Evaluating:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import
pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM,
Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix
```

```
# Define the file path for the cleaned dataset
file_path = "/content/drive/My Drive/IMDB Dataset
Cleaned.csv"

# Load the cleaned dataset
df = pd.read_csv(file_path)

# Prepare the text data and labels
texts = df['cleaned_review'].tolist()
labels = df['sentiment'].apply(lambda x: 1 if x == 'positive'
else 0).values

# Tokenize the text data
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

# Pad sequences to ensure uniform length
max_sequence_length = 100
X = pad_sequences(sequences,
maxlen=max_sequence_length)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels,
test_size=0.2, random_state=42)

# Build the LSTM model
model = Sequential()
model.add(Embedding(10000, 128,
input_length=max_sequence_length))
model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

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

# Set early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss',
patience=3)

# Train the model
history = model.fit(X_train, y_train, epochs=10,
batch_size=128, validation_split=0.2,
callbacks=[early_stopping])

# Evaluate the model
y_pred = (model.predict(X_test) > 0.5).astype("int32")
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
precision = (conf_matrix[1, 1]) / (conf_matrix[1, 1] +
conf_matrix[0, 1])
recall = (conf_matrix[1, 1]) / (conf_matrix[1, 1] +
conf_matrix[1, 0])
f1_score = 2 * (precision * recall) / (precision + recall)

print(f"Accuracy: {accuracy}")
print("\nClassification Report:")
print(report)
print("\nConfusion Matrix:")
print(conf_matrix)
print(f"\nPrecision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1_score}")

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g',
cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

F. Code for Bi-LSTM Model Training, Testing and Evaluating:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

# Define the file path for the cleaned dataset
file_path = "/content/drive/My Drive/IMDB Dataset
Cleaned.csv"

# Load the cleaned dataset
df = pd.read_csv(file_path)

# Tokenize the text data
texts = df['cleaned_review'].tolist()
labels = df['sentiment'].apply(lambda x: 1 if x == 'positive'
else 0).values

# Convert texts to sequences
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

# Pad sequences to ensure uniform length
max_sequence_length = 100
X = pad_sequences(sequences,
maxlen=max_sequence_length)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels,
test_size=0.2, random_state=42)

# Create PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.long)
y_train = torch.tensor(y_train, dtype=torch.float)
X_test = torch.tensor(X_test, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.float)
```

```
# Create DataLoader
train_dataset = TensorDataset(X_train, y_train)
test_dataset = TensorDataset(X_test, y_test)
batch_size = 32
train_loader = DataLoader(train_dataset,
batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size)

# Define Bi-LSTM model
class BiLSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim,
output_dim):
        super(BiLSTM, self).__init__()
        self.embedding = nn.Embedding(vocab_size,
embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim,
bidirectional=True, batch_first=True)
        self.fc = nn.Linear(hidden_dim * 2, output_dim)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        embedded = self.embedding(x)
        lstm_out, _ = self.lstm(embedded)
        lstm_out = self.dropout(lstm_out)
        out = self.fc(lstm_out[:, -1, :])
        return out

# Instantiate the model
vocab_size = len(tokenizer.word_index) + 1
embedding_dim = 128
hidden_dim = 64
output_dim = 1
model = BiLSTM(vocab_size, embedding_dim, hidden_dim,
output_dim)

# Define loss function and optimizer
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Train the model
num_epochs = 5
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs.squeeze(), labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch+1}/{num_epochs}, Loss:
{running_loss / len(train_loader)}")

# Evaluate the model
model.eval()
predictions = []
true_labels = []
with torch.no_grad():
    for inputs, labels in test_loader:
        outputs = model(inputs)
        preds = torch.round(torch.sigmoid(outputs))
        predictions.extend(preds.tolist())
        true_labels.extend(labels.tolist())

# Calculate evaluation metrics
accuracy = accuracy_score(true_labels, predictions)
precision = precision_score(true_labels, predictions)
recall = recall_score(true_labels, predictions)
f1 = f1_score(true_labels, predictions)
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

REFERENCES

[1] Ali, Nehal Mohamed, Marwa Mostafa Abd El Hamid, and Aliaa Youssif. "Sentiment analysis for movies reviews dataset using deep learning models." *International Journal of Data Mining & Knowledge Management Process (IJDKP) Vol* 9 (2019).

[2] Qaisar, Saeed Mian. "Sentiment analysis of IMDb movie reviews using long short-term memory." In *2020 2nd International Conference on Computer and Information Sciences (ICCIS)*, pp. 1-4. IEEE, 2020.

[3] Amulya, K., S. B. Swathi, P. Kamakshi, and Y. Bhavani. "Sentiment analysis on IMDB movie reviews using machine learning and deep learning algorithms." In *2022 4th international conference on smart systems and inventive technology (ICSSIT)*, pp. 814-819. IEEE, 2022.

[4] Minaee, Shervin, Elham Azimi, and AmirAli Abdolrashidi. "Deep-sentiment: Sentiment analysis using ensemble of cnn and bi-lstm models." *arXiv preprint arXiv:1904.04206* (2019).

[5] Vimali, J. S., and S. Murugan. "A text based sentiment analysis model using bi-directional lstm networks." In *2021 6th International conference on communication and electronics systems (ICCES)*, pp. 1652-1658. IEEE, 2021.

[6] Wang, Jin, Liang-Chih Yu, K. Robert Lai, and Xuejie Zhang. "Dimensional sentiment analysis using a regional CNN-LSTM model." In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*, pp. 225-230. 2016.

[7] Huang, Faliang, Xuelong Li, Changan Yuan, Shichao Zhang, Jilian Zhang, and Shaojie Qiao. "Attention-emotion-enhanced convolutional LSTM for sentiment analysis." *IEEE transactions on neural networks and learning systems* 33, no. 9 (2021): 4332-4345.

[8] Murthy, G. S. N., Shanmukha Rao Allu, Bhargavi Andhavarapu, Mounika Bagadi, and Mounika Belusonti. "Text based sentiment analysis using LSTM." *Int. J. Eng. Res. Tech. Res* 9, no. 05 (2020).

[9] Behera, Ranjan Kumar, Monalisa Jena, Santanu Kumar Rath, and Sanjay Misra. "Co-LSTM: Convolutional LSTM model for sentiment analysis in social big data." *Information Processing & Management* 58, no. 1 (2021): 102435.

[10] Tyagi, Abhilasha, and Naresh Sharma. "Sentiment analysis using logistic regression and effective word score heuristic." *International Journal of Engineering and Technology (UAE)* 7, no. 2 (2018): 20-23.

[11] Aliman, G., Tanya Faye S. Nivera, Jensine Charmille A. Olazo, Daisy Jane P. Ramos, Chris Danielle B. Sanchez, Timothy M. Amado, Nilo M. Arago, Romeo L. Jorda Jr, Glenn C. Virrey, and Ira C. Valenzuela. "Sentiment analysis using logistic regression." *Journal of Computational Innovations and Engineering Applications* 7, no. 1 (2022): 35-40.

[12] Ramadhan, W. P., STMT Astri Novianty, and STMT Casi Setianingsih. "Sentiment analysis using multinomial logistic regression." In *2017 International Conference on Control, Electronics, Renewable Energy and Communications (ICCREC)*, pp. 46-49. IEEE, 2017.

[13] Prabhat, Anjuman, and Vikas Khullar. "Sentiment classification on big data using Naïve Bayes and logistic regression." In *2017 International Conference on Computer Communication and Informatics (ICCCI)*, pp. 1-5. IEEE, 2017.

[14] Bhargava, Kunal, and Rahul Katarya. "An improved lexicon using logistic regression for sentiment analysis." In *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*, pp. 332-337. IEEE, 2017.