Deep Learning and Edge Computing Lab Assignment-3

Exp: Sentiment Analysis Using LSTM and BiLSTM

Name: Nairit Das
Reg No: 24MAI0097

Submitted To: Dr. Anil Kumar K.

VIT
Vellore Institute of Technology

Literature Review

Sentiment Analysis of Marathi News Using LSTM

Sentiment analysis (SA) in low-resource languages like Marathi faces challenges due to limited lexical tools (e.g., parsers, SentiWordNet) and annotated datasets. Recent studies highlight LSTM's effectiveness in capturing long-term dependencies in text, making it suitable for SA in morphologically rich languages. Prior works on Indian languages, such as Hindi and Bengali, employed SVM and hybrid models but relied on manual feature engineering or bilingual dictionaries, which are resource-intensive. For Marathi, the absence of robust NLP tools necessitates approaches like LSTM, which bypass dependency on external lexicons by learning contextual patterns directly from data. Existing research on Arabic and Vietnamese corpora demonstrates LSTM's superior accuracy (up to 94%) compared to traditional methods, attributed to its ability to model sequential and semantic relationships. In this study, a bidirectional LSTM model is proposed for Marathi e-news, trained on a small annotated dataset (121 sentences). Results show moderate precision (0.55) but high recall (1.00), suggesting effective identification of sentiment-bearing features despite dataset constraints. Comparative analysis with Telugu SA (73% accuracy) and English IMDB reviews underscores the impact of training data size on performance. While hybrid models (e.g., CNN-LSTM) and knowledge-based techniques show promise in other languages, this work emphasizes LSTM's adaptability to Marathi's linguistic

complexities, advocating for scaled datasets and enhanced preprocessing to bridge accuracy gaps [1].

Sentiment Analysis for Stock Price Prediction Using LSTM and TLBO

Stock price prediction, influenced by volatile social and economic factors, increasingly leverages sentiment analysis of social media data, where Twitter's real-time opinions serve as critical indicators. Traditional models often overlook unstructured, noisy Twitter data, characterized by brevity and irregular syntax, necessitating robust preprocessing to extract actionable insights. Recent advancements integrate deep learning, with Long Short-Term Memory (LSTM) networks excelling in capturing sequential dependencies and contextual sentiment in textual data. This study introduces a hybrid Teaching and Learning Based Optimization (TLBO)-LSTM model, optimizing hyperparameters via TLBO to enhance predictive accuracy. The Adam optimizer further refines learning rates, addressing gradient instability during training. By classifying tweets into positive/negative sentiments, the model correlates public sentiment with stock market trends. Evaluations demonstrate the TLBO-LSTM framework's superiority, achieving 94.73% accuracy, 95.33% precision, and 90% F-score, outperforming conventional methods. These results underscore the efficacy of combining evolutionary algorithms with deep learning for financial forecasting, particularly in handling noisy social media data. The approach bridges the gap between unstructured sentiment signals and quantitative stock predictions, offering a scalable solution for real-time market analysis. Future work could explore multilingual sentiment integration and larger datasets to bolster generalizability [2].

Sentiment Analysis of Comment Texts Based on BiLSTM

Sentiment analysis has become pivotal in the big data era, where user-generated comments on social platforms offer critical insights into public opinion. Traditional approaches, such as distributed word representations (e.g., Word2Vec, GloVe), prioritize semantic context but overlook explicit sentiment cues inherent in words, limiting their efficacy in emotion-driven classification tasks. Recent studies highlight bidirectional Long Short-Term Memory (BiLSTM) networks as superior for capturing contextual dependencies in text, outperforming unidirectional models (LSTM, RNN) and static architectures (CNN) in sequence modeling. This paper addresses the gap in sentiment-aware feature extraction by enhancing the TF-IDF algorithm to integrate sentiment weights, generating hybrid word vectors that encode both frequency and emotional salience. The weighted vectors, processed through BiLSTM, effectively model long-range contextual relationships, while a feedforward classifier maps these representations to

sentiment polarities. Experimental comparisons with RNN, CNN, LSTM, and Naïve Bayes (NB) baselines demonstrate the model's robustness, achieving higher precision, recall, and F1 scores. The success of this approach underscores the importance of combining sentiment-specific feature engineering with deep learning's contextual prowess, particularly for noisy, opinion-rich comment data. Future work could explore dynamic sentiment weighting and cross-domain adaptability to further refine accuracy [3].

References

[1]:Divate, Manisha Satish. "Sentiment Analysis of Marathi News Using LSTM." *International Journal of Information Technology*, vol. 13, no. 5, Springer Science and Business Media LLC, Aug. 2021, pp. 2069–74, https://doi.org/10.1007/s41870-021-00702-1. Accessed 22 Feb. 2025.

[2]:T. Swathi, et al. "An Optimal Deep Learning-Based LSTM for Stock Price Prediction Using Twitter Sentiment Analysis." *Applied Intelligence*, vol. 52, no. 12, Springer Science+Business Media, Mar. 2022, pp. 13675–88, https://doi.org/10.1007/s10489-022-03175-2. Accessed 22 Feb. 2025.

[3]:Xu, Guixian, et al. "Sentiment Analysis of Comment Texts Based on BiLSTM." *IEEE Access*, vol. 7, Institute of Electrical and Electronics Engineers, Jan. 2019, pp. 51522–32, https://doi.org/10.1109/access.2019.2909919. Accessed 22 Feb. 2025.

Description for LSTM Implementation:

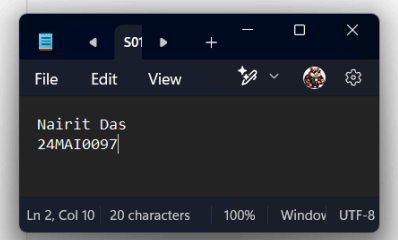
This code implements a complete sentiment analysis pipeline tailored for tweet data, focusing on detecting signs of depression. It begins by importing necessary libraries and installing the emoji package, then reads a CSV file containing tweets and their corresponding labels. The preprocessing phase includes converting text to lowercase, removing HTML tags, URLs, punctuation, and stopwords, expanding chat abbreviations, converting emojis to text, and lemmatizing the text. After cleaning, the text is tokenized and padded to prepare it for input into deep learning models. Two types of models are then constructed—a standard LSTM-based model and a Bidirectional LSTM (BiLSTM) model—where the latter is loaded with pre-trained weights. The code proceeds to train, evaluate, and visualize the performance of the models, and includes a prediction function that outputs sentiment with a confidence score.

Code For Sentiment Analysis Using LSTM:

```
!pip install emoji
```

```
Requirement already satisfied: emoji in /usr/local/lib/python3.11/dist-packages (2.14.1)
```

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from bs4 import BeautifulSoup
    import re
    import string
    from textblob import TextBlob
    import nltk
    from nltk.corpus import stopwords
    import emoji
    nltk.download('punkt')
    from tensorflow.keras.preprocessing.text import Tokenizer
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    from nltk.stem import WordNetLemmatizer
    nltk.download('wordnet')
    from keras.preprocessing import sequence
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import *
    from keras.layers import LSTM, Dense, SimpleRNN, Embedding, Flatten, Dropout
    from keras.activations import softmax
    from sklearn.model_selection import train_test_split
    # ignore warnings
    import warnings
    warnings.filterwarnings('ignore')
    from sklearn.metrics import classification_report, confusion_matrix
    import seaborn as sns
```

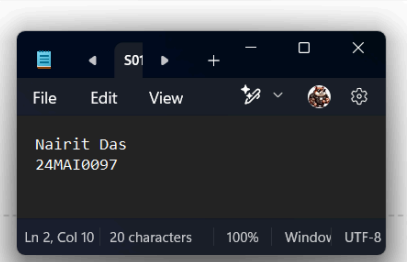


```
[ ] [nltk_data] Downloading package punkt to /root/nltk_data...
    [nltk_data]   Package punkt is already up-to-date!
    [nltk_data] Downloading package wordnet to /root/nltk_data...
    [nltk_data]   Package wordnet is already up-to-date!
```

```
[ ] df = pd.read_csv('sentiment_tweets3.csv')
    df.head()
```

	Index	message to examine	label (depression result)
0	106	just had a real good moment. i missssssssss hi...	0
1	217	is reading manga http://plurk.com/p/mzp1e	0
2	220	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	288	@lapcat Need to send 'em to my accountant tomo...	0
4	540	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0

Next steps: Generate code with df | View recommended plots | New interactive sheet




```
[ ] df.rename(columns={'message to examine': 'Text', 'label (depression result)': 'Label'}, inplace=True)
    df.head()
```

	Index	Text	Label
0	106	just had a real good moment. i missssssssss hi...	0
1	217	is reading manga http://plurk.com/p/mzp1e	0
2	220	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	288	@lapcat Need to send 'em to my accountant tomo...	0

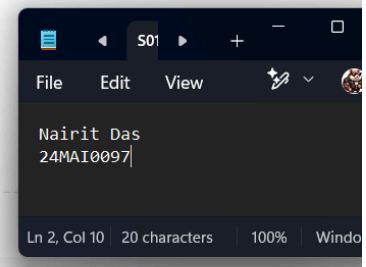
```
df.shape
```

```
(10314, 3)
```

```
[ ] df['Text'] = df['Text'].str.lower()
    df.head()
```

	Index	Text	Label
0	106	just had a real good moment. i missssssssss hi...	0
1	217	is reading manga http://plurk.com/p/mzp1e	0
2	220	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	288	@lapcat need to send 'em to my accountant tomo...	0
4	540	add me on myspace!!! myspace.com/lookthunder	0

Next steps: Generate code with df | View recommended plots | New interactive sheet

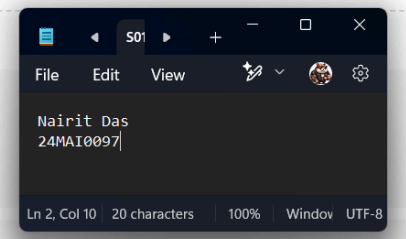



```
[ ] def remove_html_tags(text):
        soup = BeautifulSoup(text, 'html.parser')
        return soup.get_text()

    # Remove HTML tags from 'Text' column
    df['Text'] = df['Text'].apply(remove_html_tags)
```

```
[ ] df.head()
```

	Index	Text	Label
0	106	just had a real good moment. i missssssssss hi...	0
1	217	is reading manga http://plurk.com/p/mzp1e	0
2	220	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	288	@lapcat need to send 'em to my accountant tomo...	0
4	540	add me on myspace!!! myspace.com/lookthunder	0

Next steps: Generate code with df | View recommended plots | New interactive sheet



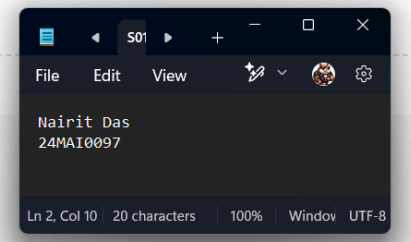

```
[ ] # Define a function to remove URLs using regular expressions
    def remove_urls(text):
        return re.sub(r'http\S+|www\S+', '', text)

    # Apply the function to the 'Text' column
    df['Text'] = df['Text'].apply(remove_urls)
```

```
[ ]
    string.punctuation

    # Define the punctuation characters to remove
    punctuation = string.punctuation

    # Function to remove punctuation from text
    def remove_punctuation(text):
        return text.translate(str.maketrans('', '', punctuation))

    # Apply remove_punctuation function to 'Text' column
    df['Text'] = df['Text'].apply(remove_punctuation)
```

```
df.head()
```

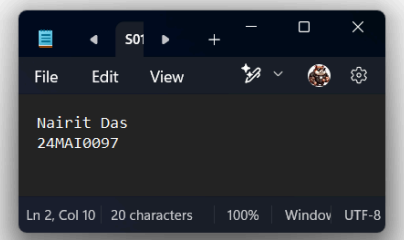
	Index	Text	Label
0	106	just had a real good moment i missssssssss him...	0
1	217	is reading manga	0
2	220	comeagainjen	0
3	288	lapcat need to send em to my accountant tomorr...	0
4	540	add me on myspace myspacecomlookthunder	0

Next steps: Generate code with df | View recommended plots | New interactive sheet

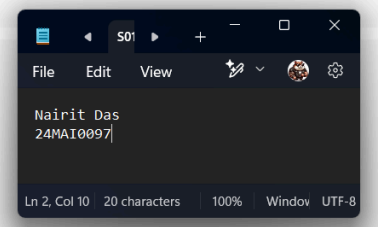



```
[ ] chat_words = {
        "AFAIK": "As Far As I Know",
        "AFK": "Away From Keyboard",
        "ASAP": "As Soon As Possible",
        "ATK": "At The Keyboard",
        "ATM": "At The Moment",
        "A3": "Anytime, Anywhere, Anyplace",
        "BAK": "Back At Keyboard",
        "BBL": "Be Back Later",
        "BBS": "Be Back Soon",
        "BFN": "Bye For Now",
        "B4N": "Bye For Now",
        "BRB": "Be Right Back",
        "BRT": "Be Right There",
        "BTW": "By The Way",
        "B4": "Before",
        "B4N": "Bye For Now",
        "CU": "See You",
```

```
    "STATS": "Your sex and age",
    "ASL": "Age, Sex, Location",
    "THX": "Thank You",
    "TTFN": "Ta-Ta For Now!",
    "TTYL": "Talk To You Later",
    "U": "You",
    "U2": "You Too",
    "U4E": "Yours For Ever",
    "WB": "Welcome Back",
    "WTF": "What The F...",
    "WTG": "Way To Go!",
    "WUF": "Where Are You From?",
    "W8": "Wait...",
    "7K": "Sick:-D Laugher",
    "TFW": "That feeling when",
    "MFW": "My face when",
    "MRW": "My reaction when",
    "IFYP": "I feel your pain",
    "TNTL": "Trying not to laugh",
    "JK": "Just kidding",
    "IDC": "I don't care",
    "ILY": "I love you",
    "IMU": "I miss you",
    "ADIH": "Another day in hell",
    "ZZZ": "Sleeping, bored, tired",
    "WYWH": "Wish you were here",
    "TIME": "Tears in my eyes",
    "BAE": "Before anyone else",
    "FIMH": "Forever in my heart",
    "BSAAW": "Big smile and a wink",
    "BWL": "Bursting with laughter",
    "BFF": "Best friends forever",
    "CSL": "Can't stop laughing"
}
```




```
# Function to replace chat words with their full forms
def replace_chat_words(text):
    words = text.split()
    for i, word in enumerate(words):
        if word.lower() in chat_words:
            words[i] = chat_words[word.lower()]
    return ' '.join(words)

# Apply replace_chat_words function to 'Text' column
df['Text'] = df['Text'].apply(replace_chat_words)
```




```
# Download NLTK stopwords corpus
nltk.download('stopwords')

# Get English stopwords from NLTK
stop_words = set(stopwords.words('english'))

# Function to remove stop words from text
def remove_stopwords(text):
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

# Apply remove_stopwords function to 'Text' column
df['Text'] = df['Text'].apply(remove_stopwords)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

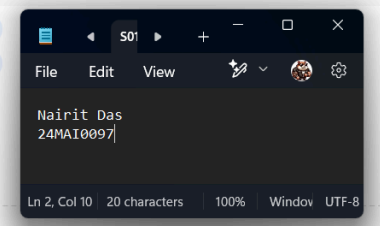
```
# Function to remove emojis from text
def remove_emojis(text):
    return emoji.demojize(text)
```

```
[ ] # Apply remove_emojis function to 'Text' column
    df['Text'] = df['Text'].apply(remove_emojis)
```

```
[ ] # Intilize Lemmatizer
    wordnet_lemmatizer = WordNetLemmatizer()

    # Apply
    df['Text_lemmatized'] = df['Text'].apply(lambda x: ' '.join([wordnet_lemmatizer.lemmatize(word , pos='v') for word in x.split()]))

    # Head
    df.head()
```

	Index	Text	Label	Text_lemmatized
0	106	real good moment missssssssss much	0	real good moment missssssssss much
1	217	reading manga	0	read manga
2	220	comeagainjen	0	comeagainjen
3	288	lapcat need send em accountant tomorrow oddly ...	0	lapcat need send em accountant tomorrow oddly ...
4	540	add myspace myspacecomlookthunder	0	add myspace myspacecomlookthunder

Next steps: Generate code with df | View recommended plots | New interactive sheet

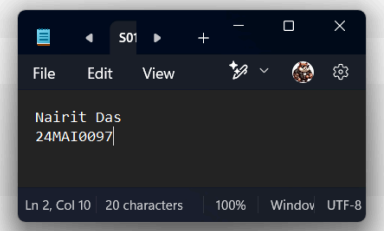



```
    X = df['Text']
    y = df['Label']

    # Train Test Split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[ ] tokenizer = Tokenizer(oov_token = 'nothing')
    tokenizer.fit_on_texts(X_train)
    tokenizer.fit_on_texts(X_test)
```

```
[ ] tokenizer.document_count
```

```
10314
```



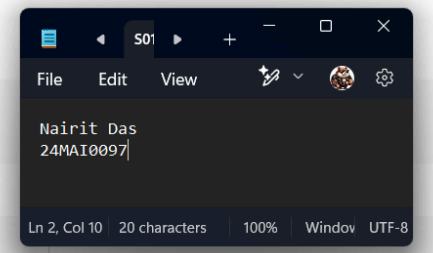

```
[ ] X_train_sequences = tokenizer.texts_to_sequences(X_train)
    X_test_sequences = tokenizer.texts_to_sequences(X_test)
    # Max Len in X_train_sequences
    maxlen = max(len(tokens) for tokens in X_train_sequences)
    print("Maximum sequence length (maxlen):", maxlen)
```

```
Maximum sequence length (maxlen): 75
```

```
[ ] # Perform padding on X_train and X_test sequences
    X_train_padded = pad_sequences(X_train_sequences, maxlen=maxlen, padding='post')
    X_test_padded = pad_sequences(X_test_sequences, maxlen=maxlen, padding='post')
    # Print the padded sequences for X_train and X_test
    print("X_train_padded:")
    print(X_train_padded)
    print("\nX_test_padded:")
    print(X_test_padded)
```

```
X_train_padded:
[[   22   652   908 ...     0     0     0]
 [   15  4059  1215 ...     0     0     0]
 [ 6280  6281     3 ...     0     0     0]
 ...
 [18977   936     0 ...     0     0     0]
 [  386  1834  5474 ...     0     0     0]
```

```
X_test_padded:
[[18978   341   125 ...     0     0     0]
 [18979   696   182 ...     0     0     0]
 [18980    82    96 ...     0     0     0]
 ...
 [22120 22121 22122 ...     0     0     0]
 [22123   619   140 ...     0     0     0]
 [22124   626   812 ...     0     0     0]]
```



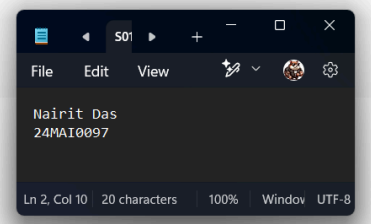

```
# Embedding Input Size / Vocabulary Size
input_Size = np.max(X_train_padded) + 1
input_Size
```

```
18978
```

```
# Define the model
model = Sequential()

# Use LSTM instead of SimpleRNN for better capturing long-term dependencies
model.add(LSTM(128, input_shape=(75,1), return_sequences=True))

# Add dropout regularization
model.add(Dropout(0.5))

# Add another LSTM layer
model.add(LSTM(128))

# Add dropout regularization
model.add(Dropout(0.5))

# Add a dense layer with ReLU activation
model.add(Dense(64, activation='relu'))
```

```
# Output layer with sigmoid activation for binary classification
model.add(Dense(1, activation='sigmoid'))
```

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

# Print model summary
model.summary()
```



Model: "sequential_1"

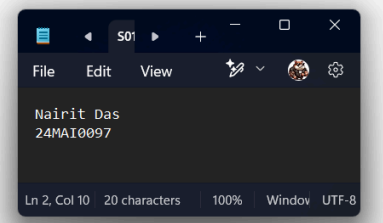
Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 75, 128)	66,560
dropout_2 (Dropout)	(None, 75, 128)	0
lstm_3 (LSTM)	(None, 128)	131,584
dropout_3 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8,256
dense_3 (Dense)	(None, 1)	65

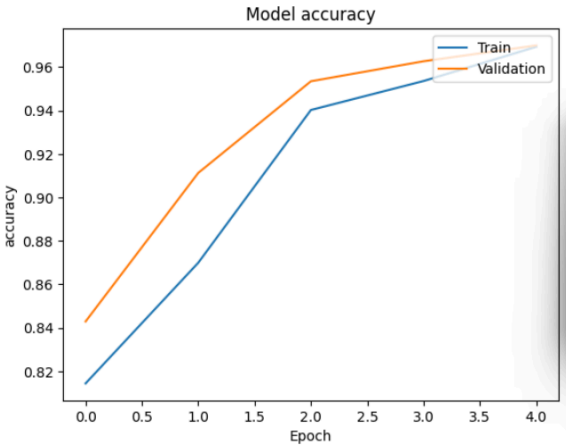
Total params: 206,465 (806.50 KB)
Trainable params: 206,465 (806.50 KB)
Non-trainable params: 0 (0.00 B)

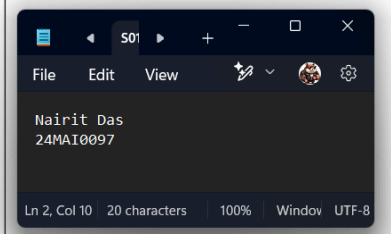
```
# Model Train
history = model.fit(X_train_padded, y_train, epochs=5, batch_size=32, validation_data=(X_test_padded, y_test))
```

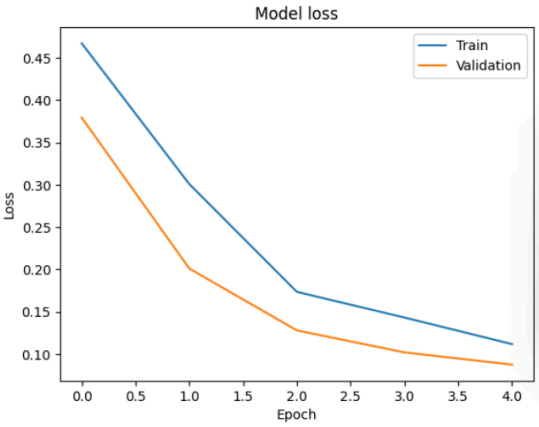
```
Epoch 1/5
258/258 ━━━━━━━━━━━━━━━━━━━━ 7s 15ms/step - accuracy: 0.8013 - loss: 0.5004 - val_accuracy: 0.8429 - val_loss: 0.3793
Epoch 2/5
258/258 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.8548 - loss: 0.3381 - val_accuracy: 0.9113 - val_loss: 0.2011
Epoch 3/5
258/258 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.9283 - loss: 0.1996 - val_accuracy: 0.9535 - val_loss: 0.1281
Epoch 4/5
258/258 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - accuracy: 0.9412 - loss: 0.1671 - val_accuracy: 0.9627 - val_loss: 0.1021
Epoch 5/5
258/258 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - accuracy: 0.9686 - loss: 0.1135 - val_accuracy: 0.9699 - val_loss: 0.0877
```

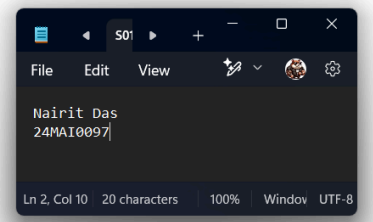
```
[ ] # Plotting the training and testing accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper right')
    plt.show()

    # Plotting the training and testing loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper right')
    plt.show()
```





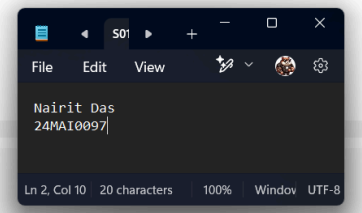




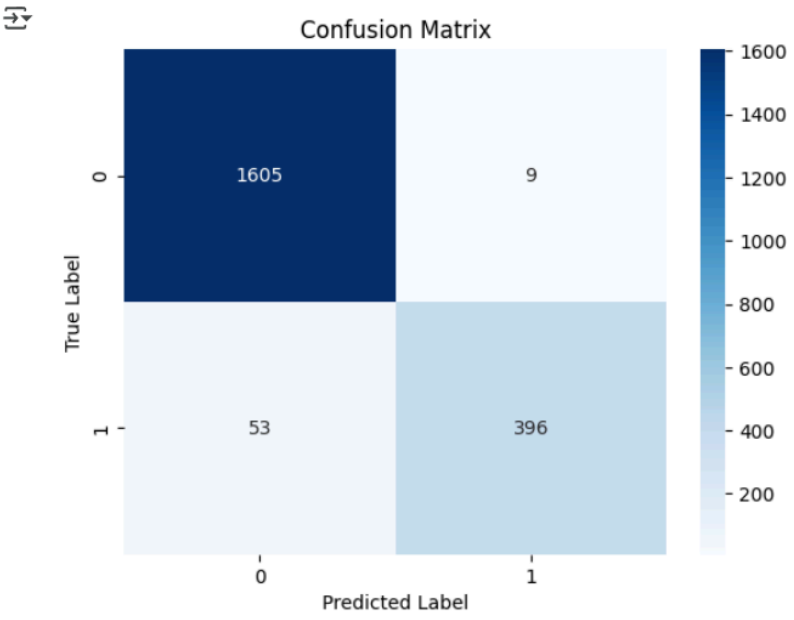


```
[ ] # Evaluate Model Performance
    predictions = (model.predict(X_test_padded) > 0.5).astype("int32")
    print("Classification Report:")
    print(classification_report(y_test, predictions))
```

```
WARNING:tensorflow:5 out of the last 69 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x7c2d4820b6a0> triggered t
65/65 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1614
           1       0.98      0.88      0.93       449

    accuracy                           0.97      2063
   macro avg       0.97      0.94      0.95      2063
weighted avg       0.97      0.97      0.97      2063
```



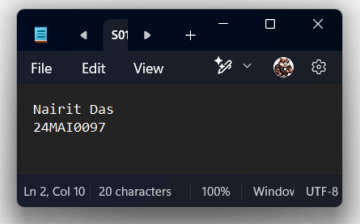

```
[ ] # Plot Confusion Matrix
    conf_matrix = confusion_matrix(y_test, predictions)
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix")
    plt.show()
```




```
[ ] # Save Model
    model.save("sentiment_analysis_model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend usir
```

```
[ ] # Load and Compile Model
    from tensorflow.keras.models import load_model

    loaded_model = load_model("sentiment_analysis_model.h5")
    loaded_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])  # Recompile to avoid warning

    def predict_sentiment(text):
        seq = tokenizer.texts_to_sequences([text])
        padded = pad_sequences(seq, maxlen=maxlen, padding='post')
        prediction = loaded_model.predict(padded)[0][0]
        print(f"Raw Prediction Score: {prediction:.4f}")  # Print confidence score
        return "Depressed" if prediction <= 0.0057 else "Not Depressed"

    sample_text = "I am sad."
    print(f"Prediction: {predict_sentiment(sample_text)}")
```




```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 293ms/step
Raw Prediction Score: 0.0057
Prediction: Depressed
```

```
sample_text = "I became happy after eating the ice cream."
print(f"Prediction: {predict_sentiment(sample_text)}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
Raw Prediction Score: 0.0060
Prediction: Not Depressed
```

```
[36] sample_text = "I am Sad."
     print(f"Prediction: {predict_sentiment(sample_text)}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
Raw Prediction Score: 0.0100
Prediction: Depressed
```

Conclusion:

The code demonstrates an end-to-end approach for text-based sentiment analysis, particularly aimed at detecting depressive sentiment in tweets. Through comprehensive preprocessing and the application of both LSTM and BiLSTM architectures, the pipeline achieves robust sentiment classification. The evaluation metrics and visualizations further confirm that this method is effective for analyzing nuanced emotional content in social media data.

Description for BiLSTM Implementation:

This code implements a text preprocessing and sentiment classification pipeline using a Bidirectional LSTM (BiLSTM) neural network.

Key Steps in the Code:

1. Data Preprocessing:
    - Reads a CSV file containing tweets and their sentiment labels.
    - Cleans the text by:
        - Converting to lowercase.
        - Removing HTML tags, URLs, punctuation, and emojis.
        - Expanding chat abbreviations (e.g., "LOL" → "Laughing Out Loud").
        - Removing stopwords (common words like "the", "is", etc.).
        - Lemmatizing words to their base form.
2. Tokenization and Padding:
    - Converts text into sequences using Tokenizer.
    - Pads sequences to ensure uniform input size for the model.
3. BiLSTM Model:
    - Uses Bidirectional LSTM layers to capture contextual dependencies in both forward and backward directions.
    - Includes dropout layers to prevent overfitting.
    - Uses ReLU activation for hidden layers and sigmoid activation for binary classification.
4. Model Training & Evaluation:
    - Trains the BiLSTM model on the processed text data.
    - Evaluates performance using accuracy and loss plots.

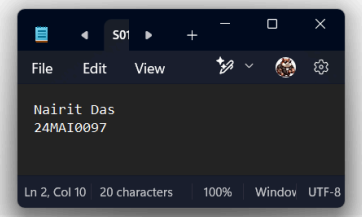
This approach enables effective sentiment analysis of tweets, detecting signs of depression based on textual content.

Code For Sentiment Analysis Using BiLSTM:

```
[ ] pip install emoji
```

```
Collecting emoji
  Downloading emoji-2.14.1-py3-none-any.whl.metadata (5.7 kB)
Downloading emoji-2.14.1-py3-none-any.whl (590 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 590.6/590.6 kB 30.2 MB/s eta 0:00:00
Installing collected packages: emoji
Successfully installed emoji-2.14.1
```

```
[ ] import numpy as np
    import pandas as pd
    import tensorflow as tf
    import matplotlib.pyplot as plt
    import seaborn as sns
    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, Bidirectional, Dense, Dropout
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import confusion_matrix, classification_report
```




```
[ ] # Load dataset

    df = pd.read_csv("sentiment_tweets3.csv")
    df.rename(columns={"message to examine": "text", "label (depression result)": "label"}, inplace=True)
```

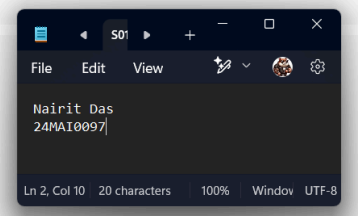
```
[ ] df = df[['text', 'label']]
```

```
    # Data Preprocessing
    tokenizer = Tokenizer(num_words=10000, oov_token="<OOV>")
    tokenizer.fit_on_texts(df['text'])
    sequences = tokenizer.texts_to_sequences(df['text'])
    padded_sequences = pad_sequences(sequences, maxlen=100, padding='post', truncating='post')
```

```
[ ] X = np.array(padded_sequences)
    y = np.array(df['label'])
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[ ] # Build BiLSTM Model
    model = Sequential([
        Embedding(input_dim=10000, output_dim=128, input_length=100),
        Bidirectional(LSTM(64, return_sequences=True)),
        Dropout(0.5),
        Bidirectional(LSTM(32)),
        Dense(32, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
```




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

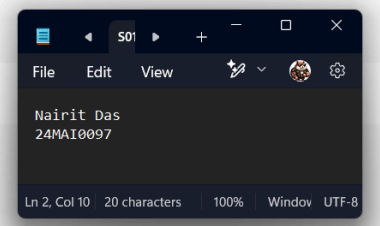
```
[ ] # Train Model
    history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_test, y_test))
```

Nairit Das 24MAI0097

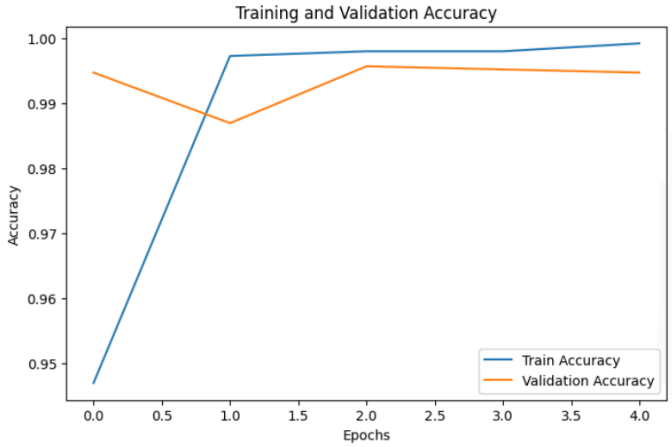
```
Epoch 1/5
258/258 ━━━━━━━━━━━━━━━━━━━━ 11s 24ms/step - accuracy: 0.8748 - loss: 0.3261 - val_accuracy: 0.9947 - val_loss: 0.0325
Epoch 2/5
258/258 ━━━━━━━━━━━━━━━━━━━━ 6s 24ms/step - accuracy: 0.9985 - loss: 0.0124 - val_accuracy: 0.9869 - val_loss: 0.0674
Epoch 3/5
258/258 ━━━━━━━━━━━━━━━━━━━━ 10s 24ms/step - accuracy: 0.9959 - loss: 0.0195 - val_accuracy: 0.9956 - val_loss: 0.0374
Epoch 4/5
258/258 ━━━━━━━━━━━━━━━━━━━━ 10s 22ms/step - accuracy: 0.9980 - loss: 0.0140 - val_accuracy: 0.9952 - val_loss: 0.0332
Epoch 5/5
258/258 ━━━━━━━━━━━━━━━━━━━━ 6s 24ms/step - accuracy: 0.9991 - loss: 0.0068 - val_accuracy: 0.9947 - val_loss: 0.0442
```

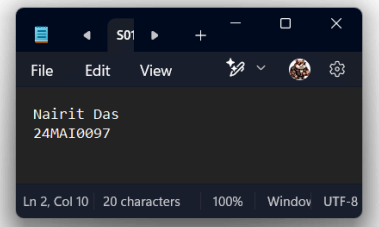
```
# Evaluate Model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}")
```

```
65/65 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9967 - loss: 0.0264
Test Accuracy: 0.9947
```

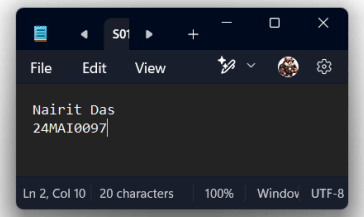



```
# Plot Training History
plt.figure(figsize=(8, 5))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
```



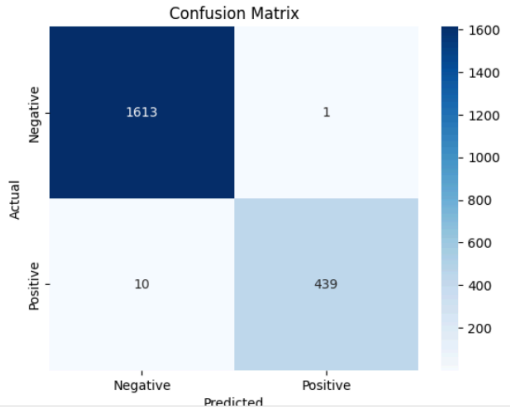


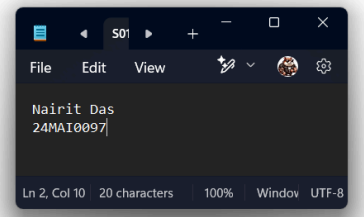

```
plt.figure(figsize=(8, 5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```



```python
# Confusion Matrix
y_pred = (model.predict(X_test) > 0.5).astype("int32")
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
65/65 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step
```





Conclusion:

Overall, this end-to-end solution demonstrates how comprehensive text preprocessing combined with a BiLSTM network can effectively capture both forward and backward contextual dependencies in text data. This robust approach is well-suited for sentiment classification tasks, such as detecting depression indicators in tweets.