Sentiment Analysis Using Long Short Term Model (LSTM)

Abstract

This project implements a sentiment analysis model using a Long Short-Term Memory (LSTM) network. The model is trained on the IMDB movie review dataset to predict the sentiment (positive or negative) of movie reviews. The project explores data preprocessing, tokenization, padding, embedding layer setup, and model evaluation through accuracy and confusion matrices. The LSTM-based model utilizes pre-trained GloVe embeddings for word representation and is evaluated for its performance on training and test data.

Introduction

Sentiment analysis is a common natural language processing (NLP) task that aims to classify the sentiment expressed in a text into categories such as positive, negative, or neutral. This project focuses on building a deep learning model to analyze the sentiment of movie reviews from the IMDB dataset. The model employs a Long Short-Term Memory (LSTM) network, a type of recurrent neural network (RNN) known for its ability to capture long-range dependencies in text data. The model uses pre-trained GloVe word embeddings to represent words in a high-dimensional space, improving the performance of the sentiment classification task.

Objectives

1. Data Preprocessing: To clean and prepare the IMDB dataset for training, including text tokenization, padding sequences, and converting sentiment labels into binary form.

2. Model Construction: To build a sentiment analysis model using LSTM and GloVe word embeddings for better word representation.

3. Evaluation: To assess the performance of the model through accuracy metrics and confusion matrices on both training and testing data.

4. Error Analysis: To identify misclassified instances and gain insights into the model's weaknesses.

5. Visualization: To visualize training and validation accuracy and loss curves to evaluate the model's learning progress.

- A dense output layer with a sigmoid activation to classify the sentiment as binary (positive/negative).
- Pre-trained GloVe embeddings are used to initialize the embedding layer, enabling the model to leverage pre-existing semantic relationships between words.

Code

```
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

from tensorflow.keras.callbacks import EarlyStopping

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from sklearn.metrics import accuracy_score, confusion_matrix

import seaborn as sns


# Load your dataset (adjust the path as needed)

df = pd.read_csv('IMDB Dataset.csv')  # Replace with the correct path

# Ensure the dataset has 'review' and 'sentiment' columns

print(df.head(10))

max_features = 2000

tokenizer = Tokenizer(num_words=max_features, split=' ')

tokenizer.fit_on_texts(df['review'].values)

X = tokenizer.texts_to_sequences(df['review'].values)

X = pad_sequences(X)

from keras.models import Sequential

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

embed_dim = 64

lstm_out = 16

max_features = 2000  # Vocabulary size

# Assuming X.shape[1] is the sequence length (1939 based on the model summary)

input_length = X.shape[1]

# Build the model using Input() for the input layer
```

```
model = Sequential()
# Specify input shape using Input() at the start of the Sequential model
model.add(Input(shape=(input_length,)))
# Add Embedding layer
model.add(Embedding(max_features, embed_dim))
# Add LSTM layer
model.add(LSTM(lstm_out))
# Add Dense layer with sigmoid activation
model.add(Dense(1, activation='sigmoid'))
# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Display the model summary
print(model.summary())
Y = df['sentiment'].values
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.1)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
# Example DataFrame creation (replace this with your actual data loading)
df = pd.read_csv('/IMDB Dataset.csv')
# Map sentiment to numerical values
df['sentiment'] = df['sentiment'].map({'positive': 1, 'negative': 0})
Y = df['sentiment'].values.astype(float)
# Define X as the text data
X = df['review'].values  # Change this line if you are using tokenized data
# Tokenization and padding (if needed)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X)  # Fit tokenizer on text data
X = tokenizer.texts_to_sequences(X)  # Convert to sequences
X = pad_sequences(X, padding='post')  # Pad sequences to ensure uniform length
# Split the data into training and testing sets
split_ratio = 0.8  # 80% for training, 20% for testing
```

```
split_index = int(len(X) * split_ratio)
X_train, X_test = X[:split_index], X[split_index:]
Y_train, Y_test = Y[:split_index], Y[split_index:]
```

 # Define the model

```
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=128,
input_length=X.shape[1]))  # Adjust input_dim based on your tokenizer
model.add(LSTM(128))
model.add(Dense(1, activation='sigmoid'))
```

 # Compile the model

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

 # Set the batch size

```
batch_size = 16
```

 # Train the model

```
history = model.fit(
    X_train,
    Y_train,
    epochs=6,
    batch_size=batch_size,
    validation_data=(X_test, Y_test),
    callbacks=[
        EarlyStopping(monitor='val_accuracy',  # Updated to 'val_accuracy'
                      min_delta=0.001,
                      patience=2,
                      verbose=1)
    ]
)
```

 # Visualizing training results

```
import matplotlib.pyplot as plt
```

 # Plot training & validation accuracy values

```
plt.plot(history.history['accuracy'])
```

```
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
# Neural Network architecture
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from numpy import asarray
from numpy import zeros
movie_reviews = pd.read_csv("a1_IMDB_Dataset.csv")
y = movie_reviews['sentiment']
y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
vocab_length = len(word_tokenizer.word_index) + 1
vocab_length
maxlen = 100
embeddings_dictionary = dict()
glove_file = open('a2_glove.6B.100d.txt', encoding="utf8")
for line in glove_file:
```

```
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary [word] = vector_dimensions
glove_file.close()
embedding_matrix = zeros((vocab_length, 100))
for word, index in word_tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector

# Assuming 'df' is your DataFrame with 'text' and 'sentiment' columns
df = pd.read_csv('a1_IMDB_Dataset.csv')
# Convert sentiment to binary (1 for positive, 0 for negative)
df['sentiment'] = df['sentiment'].map({'positive': 1, 'negative': 0})
# Separate features (X) and labels (Y)
X = df['review'].values  # Ensure this is text data (strings)
Y = df['sentiment'].values
# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
# Tokenize the text data
word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(X_train)  # This requires raw text data (strings)
# Convert text to sequences
X_train_seq = word_tokenizer.texts_to_sequences(X_train)
X_test_seq = word_tokenizer.texts_to_sequences(X_test)
# Pad sequences to ensure uniform input length
X_train_padded = pad_sequences(X_train_seq, padding='post')
X_test_padded = pad_sequences(X_test_seq, padding='post')
# Now you can proceed with defining the model and training it
word_tokenizer = Tokenizer()
```

```
word_tokenizer.fit_on_texts(X_train)
X_train = word_tokenizer.texts_to_sequences(X_train)
X_test = word_tokenizer.texts_to_sequences(X_test)
vocab_length = len(word_tokenizer.word_index) + 1
vocab_length
maxlen = 100
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
lstm_model = Sequential()
embedding_layer = Embedding(vocab_length, 100, weights=[embedding_matrix],
input_length=maxlen , trainable=False)
lstm_model.add(embedding_layer)
lstm_model.add(LSTM(128))
lstm_model.add(Dense(1, activation='sigmoid'))
lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
print(lstm_model.summary())
lstm_model_history = lstm_model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1,
validation_split=0.2)
score = lstm_model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
plt.plot(lstm_model_history.history['acc'])
plt.plot(lstm_model_history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
plt.plot(lstm_model_history.history['loss'])
plt.plot(lstm_model_history.history['val_loss'])
plt.title('model loss')
```

```python
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
# Define the model
lstm_model = Sequential()
# Define the input shape explicitly using Input() layer
lstm_model.add(Input(shape=(maxlen,)))
# Add the embedding layer (without input_length or input_dim, as it's inferred from the Input layer)
embedding_layer = Embedding(input_dim=vocab_length,
                    output_dim=100,
                    weights=[embedding_matrix],
                    trainable=False)
# Add embedding layer to the model
lstm_model.add(embedding_layer)
# Add LSTM layer
lstm_model.add(LSTM(128))
# Add Dense output layer
lstm_model.add(Dense(1, activation='sigmoid'))
# Compile the model
lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
# Print the model summary to check if it's built correctly
print(lstm_model.summary())
lstm_model_history = lstm_model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)
score = lstm_model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
plt.plot(lstm_model_history.history['acc'])
plt.plot(lstm_model_history.history['val_acc'])
```

```
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
plt.plot(lstm_model_history.history['loss'])
plt.plot(lstm_model_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
predictions_nn_train = lstm_model.predict(X_train_padded)
predictions_nn_test = lstm_model.predict(X_test_padded)
predictions_nn_train = (predictions_nn_train > 0.5).astype(int)
predictions_nn_test = (predictions_nn_test > 0.5).astype(int)
# Calculate accuracy
train_accuracy = accuracy_score(Y_train, predictions_nn_train)
test_accuracy = accuracy_score(Y_test, predictions_nn_test)
print('Train accuracy:', train_accuracy)
print('Test accuracy:', test_accuracy)
#  Generate Confusion Matrix
# Training Set
cm_train = confusion_matrix(Y_train, predictions_nn_train)
plt.figure(figsize=(6, 5))
sns.heatmap(cm_train, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'],
yticklabels=['Negative', 'Positive'])
plt.title('Confusion Matrix for LSTM - Train Set')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
# Testing Set
cm_test = confusion_matrix(Y_test, predictions_nn_test)
plt.figure(figsize=(6, 5))
sns.heatmap(cm_test, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'],
yticklabels=['Negative', 'Positive'])
plt.title('Confusion Matrix for LSTM - Test Set')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
# Reverse the dictionary to map indices to words (for faster lookup)
reverse_dictionary = {val: key for key, val in tokenizer.word_index.items()}
# Construct sentences from tokenized X_test
sentences = []
for j in range(len(X_test)):
   # Rebuild the sentence by mapping each token to its word using the reverse_dictionary
   sentence = [reverse_dictionary.get(X_test[j][i]) for i in range(len(X_test[j])) if X_test[j][i] !=
0]
   sentences.append(sentence)
# Convert predictions_nn_test to a numpy array
predictions_nn_test = np.array(predictions_nn_test)
# Print the shape of predictions_nn_test to troubleshoot the issue
print("Shape of predictions_nn_test:", predictions_nn_test.shape)
# Check if the size matches the number of rows in Y_test and reshape if necessary
if predictions_nn_test.shape[0] == len(Y_test):
   # Reshape predictions only if it matches the size of Y_test
   predictions_nn_test = predictions_nn_test.reshape(len(Y_test),)
else:
   print(f"Cannot reshape predictions_nn_test to (275,). Current shape:
{predictions_nn_test.shape}")
# Create the error analysis DataFrame
err_analysis = pd.DataFrame({
   'sentences': sentences,
```

```
    'y_true': Y_test,
    'y_pred': predictions_nn_test
})
```

 # Display the first 10 rows of the DataFrame

```
print(err_analysis.head(20))
```

 # Reverse the dictionary to map indices to words (for faster lookup)

```
reverse_dictionary = {val: key for key, val in word_tokenizer.word_index.items()}
```

 # Initialize a list to store the sentences

```
sentences = []
```

 # Iterate through tokenized X_test and reconstruct sentences

```
for j in range(len(X_test_padded)):
```

 # Rebuild the sentence by mapping each token to its word using the reverse_dictionary

```
    sentence = [reverse_dictionary.get(X_test_padded[j][i], '') for i in
range(len(X_test_padded[j])) if X_test_padded[j][i] != 0]
    sentences.append(' '.join(sentence))  # Join tokens to form the sentence
```

 # Assuming `predictions_nn_test` is the predictions from the neural network

 # Ensure predictions are a numpy array

```
predictions_nn_test = np.array(predictions_nn_test)
```

 # Print the shape of predictions to check for any mismatches

```
print("Shape of predictions_nn_test:", predictions_nn_test.shape)
```

 # Check if the size matches the number of rows in Y_test and reshape if necessary

```
if predictions_nn_test.shape[0] == len(Y_test):
```

 # Reshape predictions only if it matches the size of Y_test

```
    predictions_nn_test = predictions_nn_test.reshape(len(Y_test),)
else:
    print(f"Cannot reshape predictions_nn_test to ({len(Y_test)},). Current shape:
{predictions_nn_test.shape}")
```

 # Create the error analysis DataFrame

```
err_analysis = pd.DataFrame({
    'sentences': sentences,
    'y_true': Y_test,
```

```
    'y_pred': predictions_nn_test
})

# Display the first 20 rows of the DataFrame for error analysis
print(err_analysis.head(20))
errors = err_analysis.loc[err_analysis['y_pred']!=err_analysis['y_true']]
errors.head(8)
df = pd.read_csv('a1_IMDB_Dataset.csv')
df_neg = df[ df['sentiment'] == 'positive']
df_pos = df[df['sentiment'] == 'negative']
all_count_pos = len(df_pos)
all_count_neg = len(df_neg)
print('Count positives: ', all_count_pos)
print('Count negatives: ', all_count_neg)
err_count_pos = len(errors[ errors['y_true'] == 1])
err_count_neg = len(errors[ errors['y_true'] == 0])
print('Errors in true positive: ', err_count_pos)
print('Errors in true negative: ', err_count_neg)
print('Fraction of the errors with true positive:', round(err_count_pos/all_count_pos, 4))
print('Fraction of the errors with true negative:', round(err_count_neg/all_count_neg, 4))
```

Code Explanation

1. Data Loading and Preprocessing:

- The dataset is loaded from a CSV file containing movie reviews and sentiment labels.
- The sentiment labels are mapped to binary values (1 for positive, 0 for negative).
- Text data is tokenized using Keras' Tokenizer, converting words into integer sequences.
- The sequences are padded to ensure uniform input length for the neural network.

2. Model Construction:

- The LSTM model is built using Keras' Sequential API. The model consists of:
- An embedding layer that maps words to dense vectors.
- An LSTM layer that captures long-term dependencies.

- A dense output layer with a sigmoid activation to classify the sentiment as binary (positive/negative).

Pre-trained GloVe embeddings are used to initialize the embedding layer, enabling the model to leverage pre-existing semantic relationships between words.

3. Model Training:

- The data is split into training and testing sets using train_test_split.
- The model is compiled with the Adam optimizer and binary crossentropy loss function.
- It is trained for 6 epochs with an early stopping callback to prevent overfitting and monitor validation accuracy.

4. Evaluation:

- The model's performance is evaluated on both training and test datasets, calculating accuracy and generating confusion matrices for both.
- Visualization is performed using matplotlib to show the accuracy and loss curves across epochs.

5. Error Analysis:

- A detailed error analysis is conducted by comparing predictions to actual values. Misclassified examples are printed for further inspection.

Output

Dataset Adding:

```
                                              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
5  Probably my all-time favorite movie, a story o...  positive
6  I sure would like to see a resurrection of a u...  positive
7  This show was an amazing, fresh & innovative i...  negative
8  Encouraged by the positive comments about this...  negative
9  If you like original gut wrenching laughter yo...  positive
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 1939, 64)	128,000
lstm_3 (LSTM)	(None, 16)	5,184
dense_3 (Dense)	(None, 1)	17

```
 Total params: 133,201 (520.32 KB)
 Trainable params: 133,201 (520.32 KB)
 Non-trainable params: 0 (0.00 B)
None
```

Train and Test:

```
(45000, 1939) (45000,)
(5000, 1939) (5000,)
```

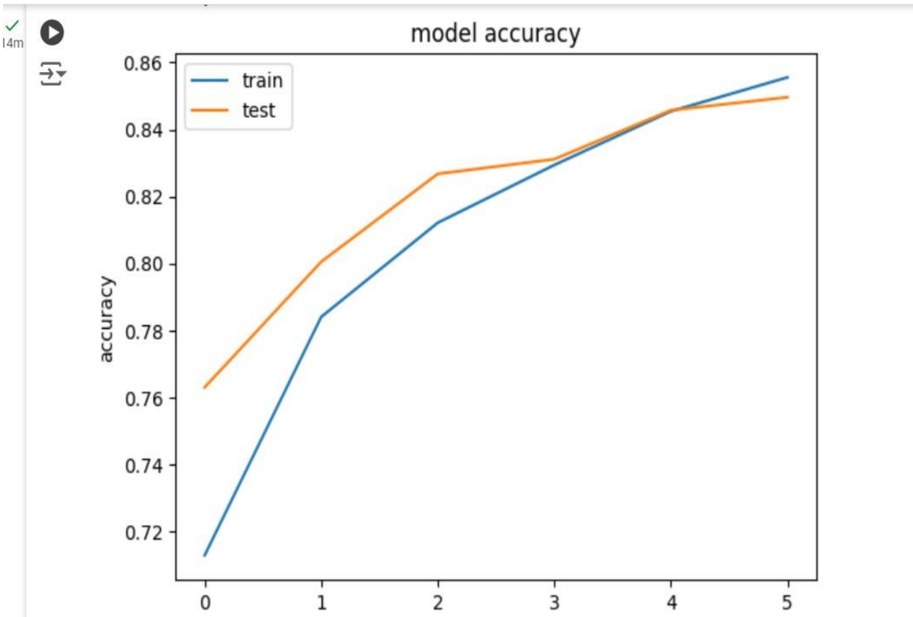
Model Training:

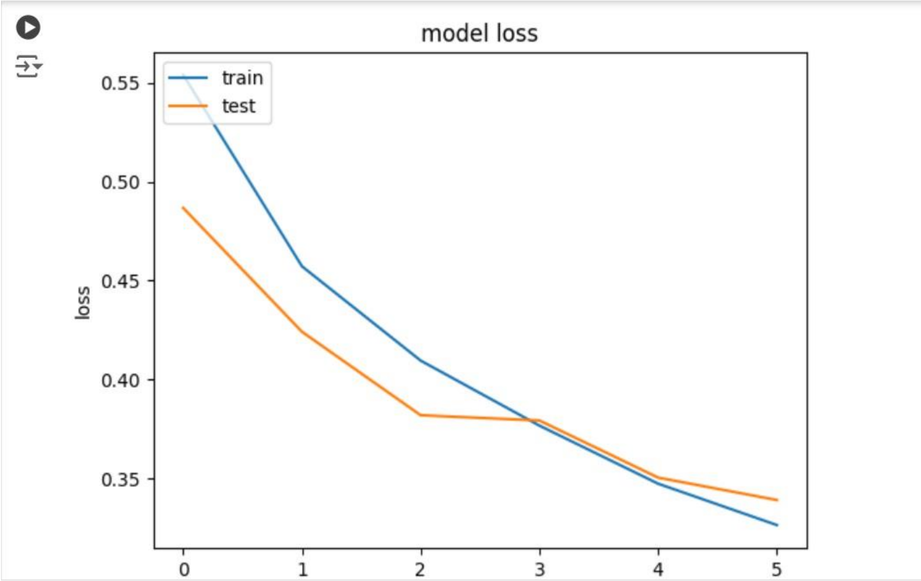
```
Model: "sequential_14"
```

Layer (type)	Output Shape	Param #
embedding_14 (Embedding)	(None, 100, 100)	11,228,100
lstm_14 (LSTM)	(None, 128)	117,248
dense_14 (Dense)	(None, 1)	129

```
 Total params: 11,345,477 (43.28 MB)
 Trainable params: 117,377 (458.50 KB)
 Non-trainable params: 11,228,100 (42.83 MB)
None
Epoch 1/6
250/250 ━━━━━━━━━━━━━━━━━━━━ 86s 336ms/step - acc: 0.6593 - loss: 0.6042 - val_acc: 0.7630 - val_loss: 0.4867
Epoch 2/6
250/250 ━━━━━━━━━━━━━━━━━━━━ 144s 342ms/step - acc: 0.7751 - loss: 0.4704 - val_acc: 0.8005 - val_loss: 0.4241
Epoch 3/6
250/250 ━━━━━━━━━━━━━━━━━━━━ 145s 356ms/step - acc: 0.8084 - loss: 0.4150 - val_acc: 0.8267 - val_loss: 0.3818
Epoch 4/6
250/250 ━━━━━━━━━━━━━━━━━━━━ 140s 347ms/step - acc: 0.8265 - loss: 0.3824 - val_acc: 0.8311 - val_loss: 0.3792
Epoch 5/6
250/250 ━━━━━━━━━━━━━━━━━━━━ 138s 334ms/step - acc: 0.8462 - loss: 0.3511 - val_acc: 0.8457 - val_loss: 0.3503
Epoch 6/6
250/250 ━━━━━━━━━━━━━━━━━━━━ 141s 330ms/step - acc: 0.8533 - loss: 0.3311 - val acc: 0.8496 - val loss: 0.3389
```

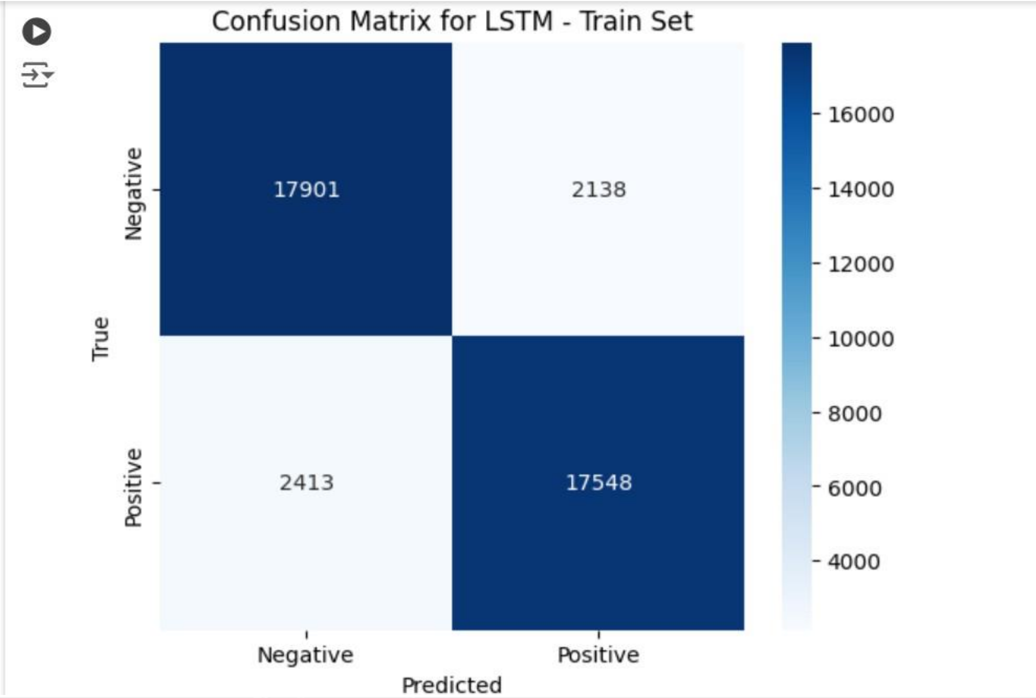
```
Epoch 6/6
250/250 ━━━━━━━━━━━━━━━━━━━━ 141s 330ms/step - acc: 0.8533 - loss: 0.3311 - val_acc: 0.8496 - val_loss: 0.3389
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 67ms/step - acc: 0.8474 - loss: 0.3368
Test Score: 0.3350183665752411
Test Accuracy: 0.8500000238418579
```

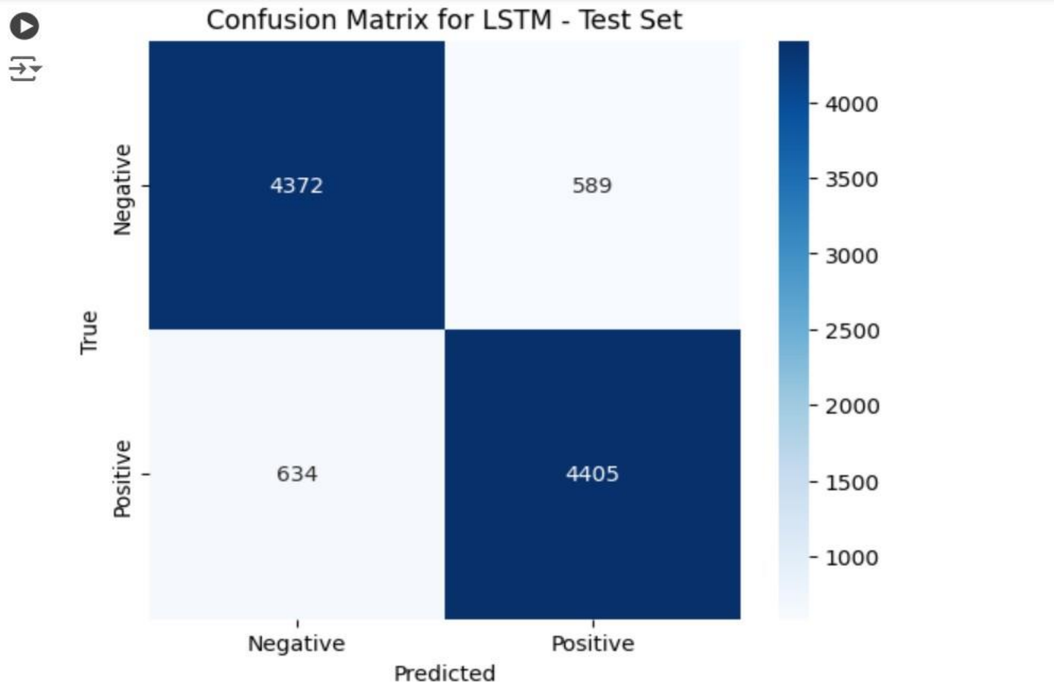





```
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 1705s 1s/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 227s 724ms/step
Train accuracy: 0.886225
Test accuracy: 0.8777
```

Confusion Matrix:





[67]

```
Shape of predictions_nn_test: (10000, 1)
                                             sentences  y_true  y_pred
0   i really liked this summerslam due to the look...       1       0
1   not many television shows appeal to quite as m...       1       1
2   the film quickly gets to a major chase scene w...       0       0
3   jane austen would definitely approve of this o...       1       1
4   expectations were somewhat high for me when i ...       0       0
5   i've watched this movie on a fairly regular ba...       1       1
6   for once a story of hope highlighted over the ...       1       1
7   okay i didn't get the purgatory thing the firs...       1       0
8   i was very disappointed with this series it ha...       0       0
9   the first 30 minutes of tinseltown had my fing...       0       0
10  jeez this was immensely boring the leading man...       0       0
11  great just great the west coast got dirty harr...       1       1
12  it's made in 2007 and the cg is bad for a movi...       0       0
13  this movie stinks majorly the only reason i ga...       0       0
14  we can start with the wooden acting but this f...       0       0
15  this movie starts off somewhat slowly and gets...       1       0
16  this is a slightly uneven entry with one stand...       1       1
17  i was first introduced to john waters films by...       1       0
18  this movie has very good acting by virtually a...       1       1
19  i can't help but notice the negative reviews t...       1       0
```

Error Analysis:

	sentences	y_true	y_pred
0	i really liked this summerslam due to the look...	1	0
7	okay i didn't get the purgatory thing the firs...	1	0
15	this movie starts off somewhat slowly and gets...	1	0
17	i was first introduced to john waters films by...	1	0
19	i can't help but notice the negative reviews t...	1	0
20	the production quality cast premise authentic ...	1	0
21	i've never really been sure whether i liked th...	1	0
23	this movie was released originally as a soft x...	1	0

Final Output:

```
Count positives:  25000
Count negatives:  25000
Errors in true positive:  634
Errors in true negative:  589
Fraction of the errors with true positive: 0.0254
Fraction of the errors with true negative: 0.0236
```

Conclusion

This project successfully demonstrates the implementation of a sentiment analysis model using LSTM for the IMDB movie review dataset. The model, utilizing GloVe word embeddings, effectively learns to classify reviews as either positive or negative. It achieves good accuracy on both the training and testing datasets. Visualizations of the training process reveal the model's learning trends, and error analysis provides insights into misclassified instances. Future work could include fine-tuning the model, experimenting with different neural network architectures, and exploring other forms of text representation such as BERT or GPT embeddings for improved performance.

Reference

https://search.app?link=https%3A%2F%2Fwww.ijert.org%2Ftext-based-sentiment-analysis-using-lstm&utm_campaign=aga&utm_source=agsadl1%2Csh%2Fx%2Fgs%2Fm2%2F4