emotion-classification-using-ltsm

June 18, 2024

1 Background

The primary goal of this project is to build a machine learning model capable of accurately detecting and classifying the emotions expressed in short text messages. The dataset used in this notebook is derived from an emotion detection dataset, consisting of sentences labeled with emotion classes such as sadness, joy, love, anger, fear, and surprise. The project involves data processing, feature extraction, model training, and performance evaluation to develop an efficient emotion classifier. This model will be useful in applications where understanding human emotions is important, such as mental health monitoring, customer feedback analysis, and sentiment detection.

2 Code Structure and Objectives

The code presented here is designed to build a machine learning model for classifying emotions in text. It starts by importing the necessary libraries, including those for data manipulation (like Pandas and Numpy), visualization (such as Matplotlib and Seaborn), text processing, and machine learning model building using TensorFlow's Keras API.

The first step involves loading the data from CSV files that represent the training, validation, and test datasets. It ensures that the validation and test datasets are balanced by splitting the test data. After verifying the shape of each dataset, it proceeds with an exploratory analysis, mapping numeric labels to their corresponding emotions like sadness, joy, and anger. This mapping is used to generate visualizations, providing insight into the distribution of emotions across the training data.

In the data preprocessing stage, a tokenizer is created using all text data across the datasets. The text data is processed using stemming (via the PorterStemmer) and is tokenized into sequences that will be fed to the neural network model. All sequences are padded to ensure uniform input lengths, crucial for efficient model training. The labels are then one-hot encoded, allowing them to be used with categorical cross-entropy loss.

The model itself is a Sequential Keras model composed of an Embedding layer for mapping word indices to dense vectors, a Bidirectional LSTM layer for capturing long-term dependencies from both directions, and a final Dense layer to classify the text into one of the six emotion classes using a softmax activation function. An Adam optimizer is used to compile the model with categorical cross-entropy as the loss function. The training phase uses the prepared training and validation datasets and visualizes training/validation accuracy and loss for each epoch.

The final step involves evaluating the model on the test data, calculating a confusion matrix to visualize prediction performance across different emotions. This structure, encompassing data preprocessing, model training, and evaluation, ultimately aims to provide a robust solution for classifying emotions in text-based communication.

3 Import Libraries and Setup

[1]:
```python
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from keras.utils import to_categorical
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
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, Bidirectional,␣
 ↪Dropout
from tensorflow.keras.optimizers import Adam
```

```
2024-06-18 08:12:33.806343: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-06-18 08:12:33.806482: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-06-18 08:12:33.949229: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
```

4 Load and Prepare Data

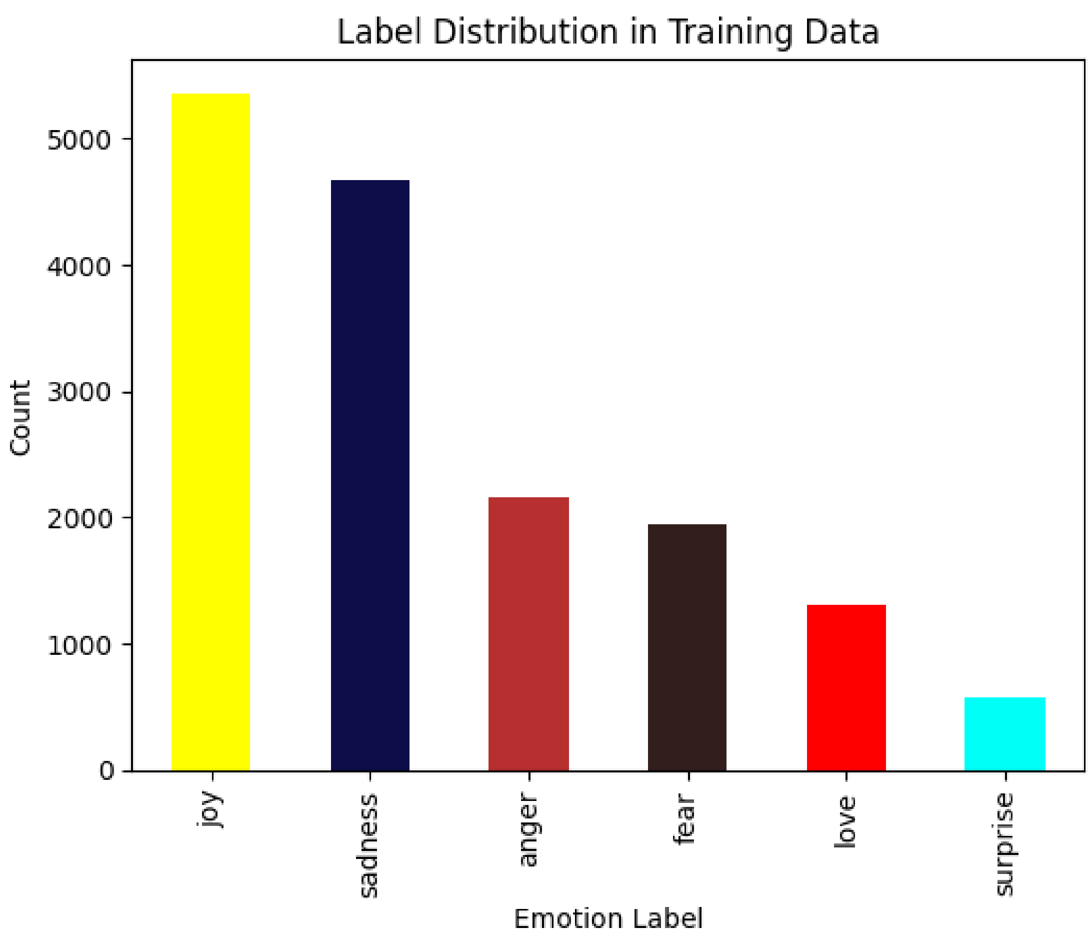
[2]:
```python
# Load datasets
val_data = pd.read_csv('/kaggle/input/emotion-dataset/validation.csv')
train_data = pd.read_csv('/kaggle/input/emotion-dataset/training.csv')
test_data = pd.read_csv('/kaggle/input/emotion-dataset/test.csv')

# Print dataset shapes
print(f"Validation data: {val_data.shape}")
print(f"Training data: {train_data.shape}")
print(f"Test data: {test_data.shape}")
```

```
# Balance validation and test data
half_test_data = test_data.iloc[1000:]
test_data = test_data.iloc[:1000]
val_data = pd.concat([val_data, half_test_data], axis=0)

print(f"New Validation data: {val_data.shape}")
print(f"New Test data: {test_data.shape}")

# Label mapping dictionary
labels_dict = {0: 'sadness', 1: 'joy', 2: 'love', 3: 'anger', 4: 'fear', 5:␣
↪'surprise'}
train_data['label_name'] = train_data['label'].map(labels_dict)
```

```
Validation data: (2000, 2)
Training data: (16000, 2)
Test data: (2000, 2)
New Validation data: (3000, 2)
New Test data: (1000, 2)
```

5 Data Exploration and Visualization

[3]:
```
# Plot the label distribution in training data
train_data['label_name'].value_counts().plot(kind='bar', color=['yellow',␣
↪'#0c0d49', '#b82f2f', '#331e1e', 'red', '#00fff7'])
plt.title('Label Distribution in Training Data')
plt.ylabel('Count')
plt.xlabel('Emotion Label')
plt.show()

# Check for missing values
print(train_data.isnull().sum())
print(val_data.isnull().sum())
print(test_data.isnull().sum())
```



```
text          0
label         0
label_name    0
dtype: int64
text     0
label    0
dtype: int64
text     0
label    0
dtype: int64
```

6 Data Preprocessing

[4]:
```
# Initialize PorterStemmer and stopwords
stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))
```

```
# Function to preprocess data with stemming and stopword removal
def preprocess_text(text):
    tokens = [stemmer.stem(word) for word in text.split() if word not in␣
 ↪stop_words]
    return " ".join(tokens)

# Apply preprocessing to datasets
train_data['clean_text'] = train_data['text'].apply(preprocess_text)
val_data['clean_text'] = val_data['text'].apply(preprocess_text)
test_data['clean_text'] = test_data['text'].apply(preprocess_text)

# Create a tokenizer with combined training, validation, and test data
all_texts = train_data['clean_text'].tolist() + test_data['clean_text'].
 ↪tolist() + val_data['clean_text'].tolist()
tokenizer = Tokenizer(num_words=16000)
tokenizer.fit_on_texts(all_texts)
word_index = tokenizer.word_index
print(f"Number of words without Stemming: {len(word_index)}")

# Function to preprocess data with stemming and tokenization
def preprocess_data(data):
    processed_data = []
    for _, row in data.iterrows():
        sequence = tokenizer.texts_to_sequences([row['clean_text'].split()])[0]
        processed_data.append([sequence, row['label']])
    return processed_data
```

```
Number of words without Stemming: 11598
```

7 Build and Train Model

[5]:
```
# Preprocess training and validation datasets
train_data_processed = preprocess_data(train_data)
val_data_processed = preprocess_data(val_data)

# Separate features and labels, and pad sequences
max_seq_length = max(len(seq[0]) for seq in train_data_processed)
train_X = pad_sequences([row[0] for row in train_data_processed],␣
 ↪maxlen=max_seq_length, padding='post')
train_y = np.array([row[1] for row in train_data_processed])

val_X = pad_sequences([row[0] for row in val_data_processed],␣
 ↪maxlen=max_seq_length, padding='post')
val_y = np.array([row[1] for row in val_data_processed])

# Convert labels to one-hot encoding
```

```python
num_classes = len(labels_dict)
train_y_one_hot = to_categorical(train_y, num_classes=num_classes)
val_y_one_hot = to_categorical(val_y, num_classes=num_classes)

print(f"Training set shape: {train_X.shape}, {train_y.shape}")
print(f"Validation set shape: {val_X.shape}, {val_y.shape}")

# Build an optimized bidirectional LSTM model
model = Sequential([
    Embedding(input_dim=16000, output_dim=100, input_length=max_seq_length),
    Bidirectional(LSTM(128, return_sequences=True)),
    Dropout(0.3),
    LSTM(64),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
])

# Compile the model
optimizer = Adam(learning_rate=0.003)
model.compile(loss='categorical_crossentropy', optimizer=optimizer,␣
 ↪metrics=['accuracy'])

# Train the model
history = model.fit(train_X, train_y_one_hot, epochs=25,␣
 ↪validation_data=(val_X, val_y_one_hot), verbose=1)
```

```
Training set shape: (16000, 35), (16000,)
Validation set shape: (3000, 35), (3000,)
Epoch 1/25
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/core/embedding.py:86:
UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

```
500/500               47s 83ms/step -
accuracy: 0.3152 - loss: 1.6045 - val_accuracy: 0.3540 - val_loss: 1.5729
Epoch 2/25
500/500               41s 82ms/step -
accuracy: 0.3386 - loss: 1.5747 - val_accuracy: 0.3540 - val_loss: 1.5735
Epoch 3/25
500/500               82s 82ms/step -
accuracy: 0.3366 - loss: 1.5109 - val_accuracy: 0.5357 - val_loss: 1.0287
Epoch 4/25
500/500               82s 83ms/step -
accuracy: 0.5875 - loss: 0.9012 - val_accuracy: 0.8767 - val_loss: 0.4218
Epoch 5/25
500/500               41s 82ms/step -
accuracy: 0.8988 - loss: 0.3389 - val_accuracy: 0.9013 - val_loss: 0.3169
```

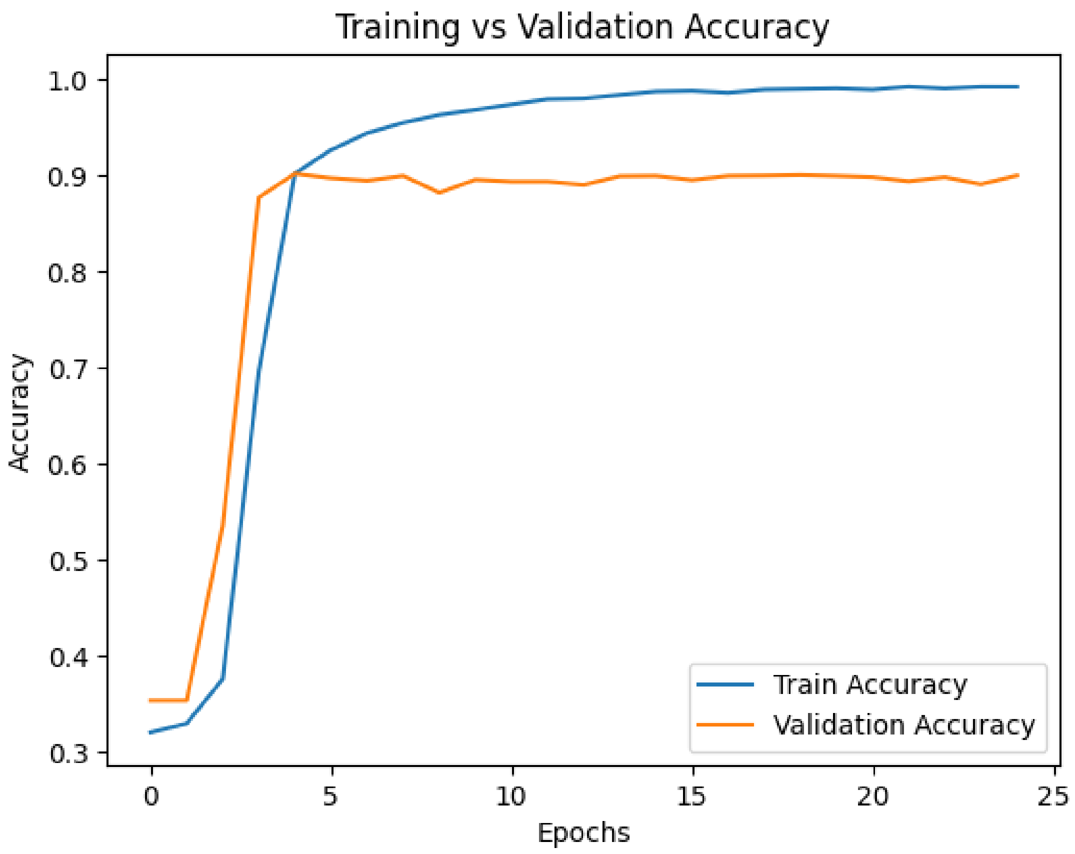
```
Epoch 6/25
500/500             82s 82ms/step -
accuracy: 0.9259 - loss: 0.2337 - val_accuracy: 0.8967 - val_loss: 0.2932
Epoch 7/25
500/500             41s 81ms/step -
accuracy: 0.9417 - loss: 0.1661 - val_accuracy: 0.8940 - val_loss: 0.3108
Epoch 8/25
500/500             40s 81ms/step -
accuracy: 0.9553 - loss: 0.1247 - val_accuracy: 0.8990 - val_loss: 0.3390
Epoch 9/25
500/500             41s 82ms/step -
accuracy: 0.9666 - loss: 0.0931 - val_accuracy: 0.8817 - val_loss: 0.3666
Epoch 10/25
500/500             41s 82ms/step -
accuracy: 0.9667 - loss: 0.0909 - val_accuracy: 0.8950 - val_loss: 0.3954
Epoch 11/25
500/500             81s 82ms/step -
accuracy: 0.9738 - loss: 0.0727 - val_accuracy: 0.8930 - val_loss: 0.4329
Epoch 12/25
500/500             41s 81ms/step -
accuracy: 0.9798 - loss: 0.0538 - val_accuracy: 0.8930 - val_loss: 0.4324
Epoch 13/25
500/500             41s 82ms/step -
accuracy: 0.9810 - loss: 0.0518 - val_accuracy: 0.8897 - val_loss: 0.4380
Epoch 14/25
500/500             41s 81ms/step -
accuracy: 0.9842 - loss: 0.0427 - val_accuracy: 0.8987 - val_loss: 0.4414
Epoch 15/25
500/500             41s 81ms/step -
accuracy: 0.9883 - loss: 0.0348 - val_accuracy: 0.8990 - val_loss: 0.4644
Epoch 16/25
500/500             41s 82ms/step -
accuracy: 0.9899 - loss: 0.0322 - val_accuracy: 0.8947 - val_loss: 0.4842
Epoch 17/25
500/500             41s 81ms/step -
accuracy: 0.9836 - loss: 0.0442 - val_accuracy: 0.8990 - val_loss: 0.4973
Epoch 18/25
500/500             41s 82ms/step -
accuracy: 0.9902 - loss: 0.0296 - val_accuracy: 0.8993 - val_loss: 0.5127
Epoch 19/25
500/500             40s 81ms/step -
accuracy: 0.9907 - loss: 0.0258 - val_accuracy: 0.9000 - val_loss: 0.5104
Epoch 20/25
500/500             41s 82ms/step -
accuracy: 0.9911 - loss: 0.0272 - val_accuracy: 0.8990 - val_loss: 0.5493
Epoch 21/25
500/500             81s 80ms/step -
accuracy: 0.9882 - loss: 0.0355 - val_accuracy: 0.8977 - val_loss: 0.5231
```

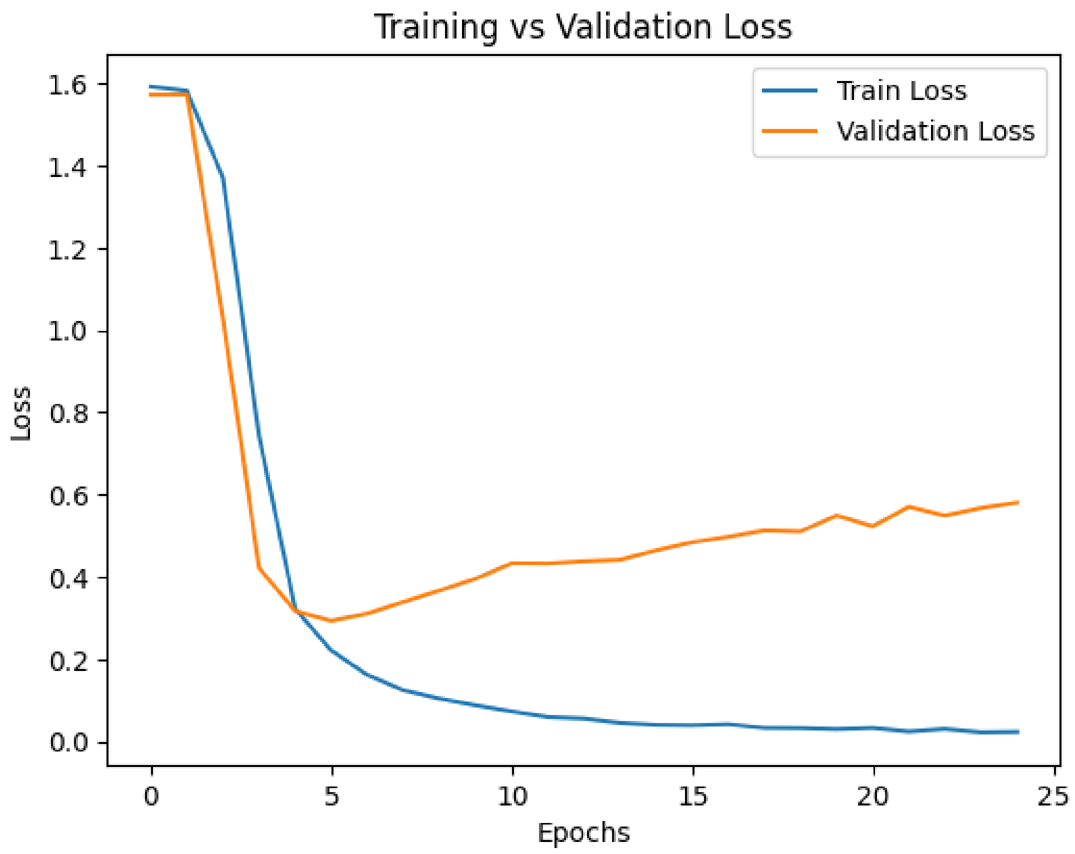
```
Epoch 22/25
500/500          40s 80ms/step -
accuracy: 0.9934 - loss: 0.0199 - val_accuracy: 0.8933 - val_loss: 0.5701
Epoch 23/25
500/500          41s 83ms/step -
accuracy: 0.9915 - loss: 0.0265 - val_accuracy: 0.8977 - val_loss: 0.5490
Epoch 24/25
500/500          41s 81ms/step -
accuracy: 0.9932 - loss: 0.0185 - val_accuracy: 0.8903 - val_loss: 0.5674
Epoch 25/25
500/500          41s 81ms/step -
accuracy: 0.9917 - loss: 0.0260 - val_accuracy: 0.8993 - val_loss: 0.5803
```

8 Evaluation and Prediction

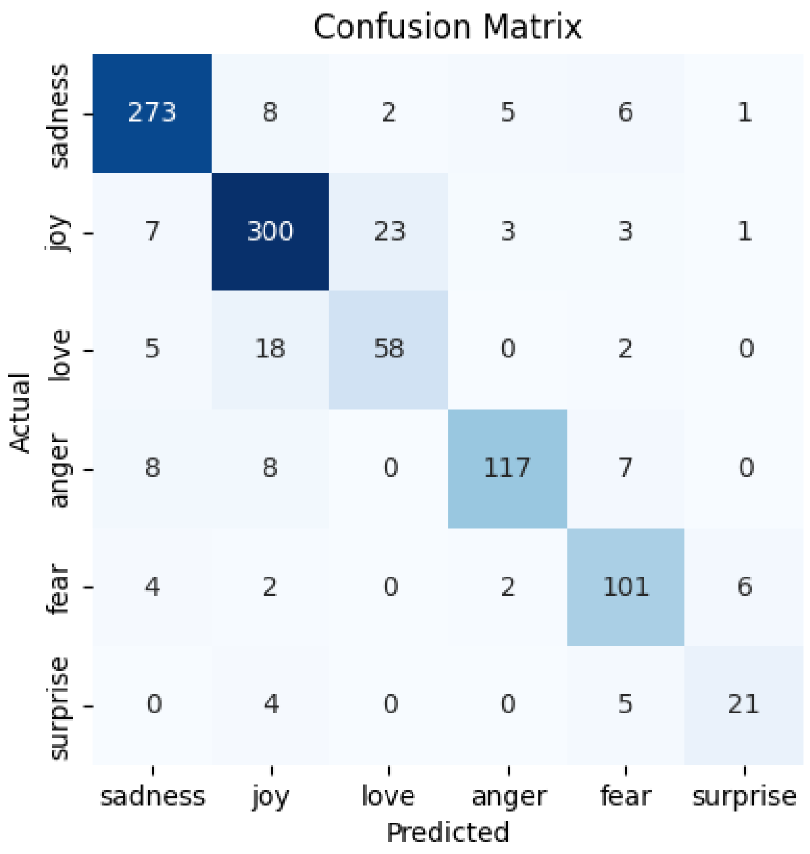
[6]:

```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training vs Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Function to predict on test data
def predict_text(text):
    sequence = tokenizer.texts_to_sequences([preprocess_text(text).split()])[0]
    sequence_padded = pad_sequences([sequence], maxlen=max_seq_length,␣
 ↪padding='post')
    prediction = model.predict(sequence_padded)
    return np.argmax(prediction)

# Random predictions for testing
for _ in range(5):
    index = random.randint(0, len(test_data) - 1)
    predicted_class = predict_text(test_data['text'][index])
    actual_class = test_data['label'][index]
    print(f"\nPredicted: {labels_dict[predicted_class]}, Actual:␣
↪{labels_dict[actual_class]}")

# Evaluate on entire test set
test_data_processed = preprocess_data(test_data)
test_X = pad_sequences([row[0] for row in test_data_processed],␣
↪maxlen=max_seq_length, padding='post')
test_y = np.array([row[1] for row in test_data_processed])

test_y_one_hot = to_categorical(test_y, num_classes=num_classes)

# Predict classes
y_pred = model.predict(test_X)
y_pred_classes = np.argmax(y_pred, axis=1)

# Confusion matrix
cm = confusion_matrix(test_y, y_pred_classes)
df_cm = pd.DataFrame(cm, index=labels_dict.values(), columns=labels_dict.
↪values())
ax = sns.heatmap(df_cm, annot=True, fmt='d', square=True, cbar=False,␣
↪cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```


Training vs Validation Accuracy
Accuracy
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
5
10
15
20
25
Train Accuracy
Validation Accuracy



```
1/1              0s 464ms/step

Predicted: sadness, Actual: sadness
1/1              0s 31ms/step

Predicted: love, Actual: joy
1/1              0s 30ms/step

Predicted: love, Actual: joy
1/1              0s 30ms/step

Predicted: joy, Actual: joy
1/1              0s 28ms/step

Predicted: sadness, Actual: sadness
32/32              1s 27ms/step
```



Actual \ Predicted	sadness	joy	love	anger	fear	surprise
sadness	273	8	2	5	6	1
joy	7	300	23	3	3	1
love	5	18	58	0	2	0
anger	8	8	0	117	7	0
fear	4	2	0	2	101	6
surprise	0	4	0	0	5	21