A_LSTM, BILSTM_Multi_classification

March 11, 2025

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer  # Updated import
from tensorflow.keras.preprocessing.sequence import pad_sequences  # Updated␣
↪import
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

from sklearn.metrics import roc_curve, auc
from itertools import cycle
```

[3]:
```
#import pandas as pd

# Load the dataset
#df = pd.read_csv('datasetforclassification_updatedd_updated_updated_updated1.
↪csv', encoding='latin1')

# Drop rows with nan values in 'Base_Reviews' and 'My_Labels' columns
#df = df.dropna(subset=['Base_Reviews', 'My_Labels'])

# Save the cleaned dataset
#df.to_csv('datasetforclassification_updatedd_updated_updated_updated12.csv',␣
↪index=False, encoding='latin1')
```

[5]:
```
# Load the dataset
#df = pd.read_csv('Amazon_Dataset_LD.csv', encoding='latin1')
df = pd.read_csv('accessibilityissues_multi.csv', encoding='latin1')
```

[7]:
```
import pandas as pd
import re
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Define a function to clean the text
def clean_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove special characters and punctuation
    text = re.sub(r'[^\w\s]', '', text)
    # Remove digits
    text = re.sub(r'\d+', '', text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = text.split()
    filtered_tokens = [token for token in tokens if token not in stop_words]
    text = ' '.join(filtered_tokens)
    # Lemmatize the words
    lemmatizer = WordNetLemmatizer()
    tokens = text.split()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]
    text = ' '.join(lemmatized_tokens)
    return text

# Assuming df is your DataFrame
# Drop rows where either 'comment_Text' or 'Sarcasm_Type' is nan
df = df.dropna(subset=['Review', 'Assessability Issue Type'])

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

# Now handle the 'Sarcasm_Type' column for label encoding
y_dict = {'Navigation and Interaction Problems (NAV)': 0, 'Input and Control␣
↪Issues (INPUT)': 1, 'Compatibility with Assistive Technologies (CAT)': 2,␣
↪'UI Accessibility Issues (UI)': 3, 'Audio and Visual Accessibility issues␣
↪(AUDIOVISUAL)': 4}
df['Sarcasm_Type_Encoded'] = df['Assessability Issue Type'].map(y_dict)
```

[9]:
```
# Split the dataset into X and y
X = df['Review'].values
y = df['Assessability Issue Type'].values
```

[11]:
```
print(y)
```

```
['Navigation and Interaction Problems (NAV)'
 'Navigation and Interaction Problems (NAV)'
 'Navigation and Interaction Problems (NAV)' …
```

```
'UI Accessibility Issues (UI)' 'UI Accessibility Issues (UI)'
'Input and Control Issues (INPUT)']
```

[13]:
```
# Label encoding and one-hot encoding
y_dict = {'Navigation and Interaction Problems (NAV)': 0, 'Input and Control␣
↪Issues (INPUT)': 1, 'Compatibility with Assistive Technologies (CAT)': 2,␣
↪'UI Accessibility Issues (UI)': 3, 'Audio and Visual Accessibility issues␣
↪(AUDIOVISUAL)': 4}  # Update as needed
y = [y_dict[item] for item in y]
```

[15]:
```
# Convert the labels to categorical variables
num_classes = len(np.unique(y))
y = keras.utils.to_categorical(y, num_classes)
```

[17]:
```
# Tokenize the data
max_features = 5000
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(X)
X = tokenizer.texts_to_sequences(X)
```

[19]:
```
# Pad the sequences
maxlen = 150
X = pad_sequences(X, maxlen=maxlen)
```

[21]:
```
# Apply oversampling to balance the classes
oversampler = RandomOverSampler(random_state=42)
X_resampled, y_resampled = oversampler.fit_resample(X, y)
```

1 LSTM Model

[24]:
```
#Define the LSTM model
lstm_model = keras.models.Sequential()
lstm_model.add(keras.layers.Embedding(max_features, 128, input_length=maxlen))
lstm_model.add(keras.layers.LSTM(128, dropout=0.5, recurrent_dropout=0.5))
lstm_model.add(keras.layers.Dense(64, activation='relu'))
lstm_model.add(keras.layers.Dense(num_classes, activation='softmax'))
lstm_model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.
↪Adam(), metrics=['accuracy'])
lstm_model.summary()
```

```
/opt/anaconda3/lib/python3.12/site-
packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(
```

Model: "sequential"

2 BiLSTM Model

[27]:
```
# Define the BiLSTM model
bilstm_model = keras.models.Sequential()
bilstm_model.add(keras.layers.Embedding(max_features, 128, input_length=maxlen))
bilstm_model.add(keras.layers.Bidirectional(keras.layers.LSTM(128, dropout=0.2,␣
↪recurrent_dropout=0.2)))
bilstm_model.add(keras.layers.Dense(64, activation='relu'))
bilstm_model.add(keras.layers.Dense(num_classes, activation='softmax'))
bilstm_model.compile(loss='categorical_crossentropy', optimizer=keras.
↪optimizers.Adam(), metrics=['accuracy'])
bilstm_model.summary()
```

```
/opt/anaconda3/lib/python3.12/site-
packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(
```

Model: "sequential_1"

[29]:
```
# Perform k-fold cross-validation
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

lstm_acc_scores = []
bilstm_acc_scores = []
```

[31]:
```
for train_index, test_index in kf.split(X_resampled):
    X_train, X_test = X_resampled[train_index], X_resampled[test_index]
    y_train, y_test = y_resampled[train_index], y_resampled[test_index]
```

[33]:
```
# Train the LSTM model on the current fold
lstm_history = lstm_model.fit(X_train, y_train, validation_data=(X_test,
 ↪y_test), epochs=10, batch_size=128, verbose=0)
lstm_loss, lstm_acc = lstm_model.evaluate(X_test, y_test, verbose=0)
lstm_acc_scores.append(lstm_acc)
```

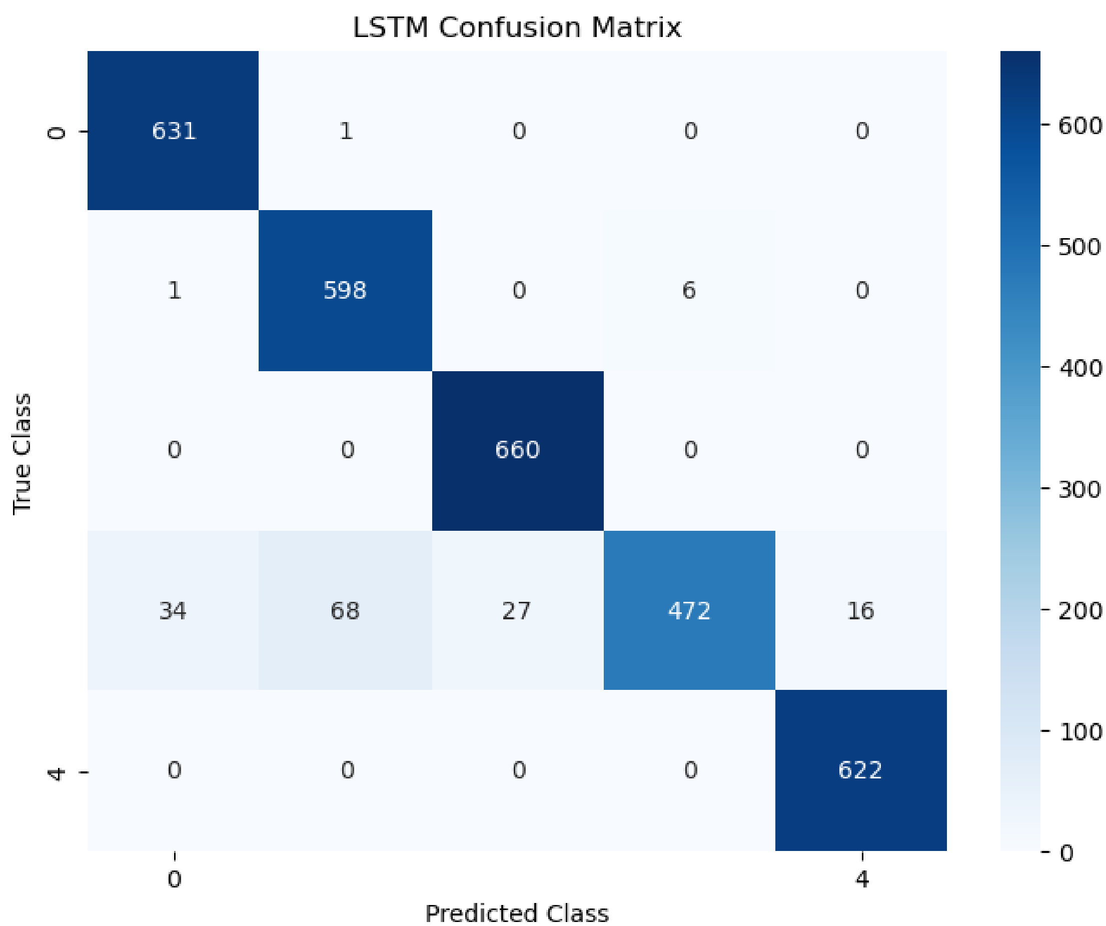
[34]:
```
# Train the BiLSTM model on the current fold
bilstm_history = bilstm_model.fit(X_train, y_train, validation_data=(X_test,
 ↪y_test), epochs=10, batch_size=128, verbose=0)
bilstm_loss, bilstm_acc = bilstm_model.evaluate(X_test, y_test, verbose=0)
bilstm_acc_scores.append(bilstm_acc)

# Calculate the average accuracy scores across the k-fold cross-validation
avg_lstm_acc = np.mean(lstm_acc_scores)
avg_bilstm_acc = np.mean(bilstm_acc_scores)
```

[37]:
```
print("LSTM: Average Accuracy = {:.2f}".format(avg_lstm_acc))
print("BiLSTM: Average Accuracy = {:.2f}".format(avg_bilstm_acc))

# Confusion Matrix
```

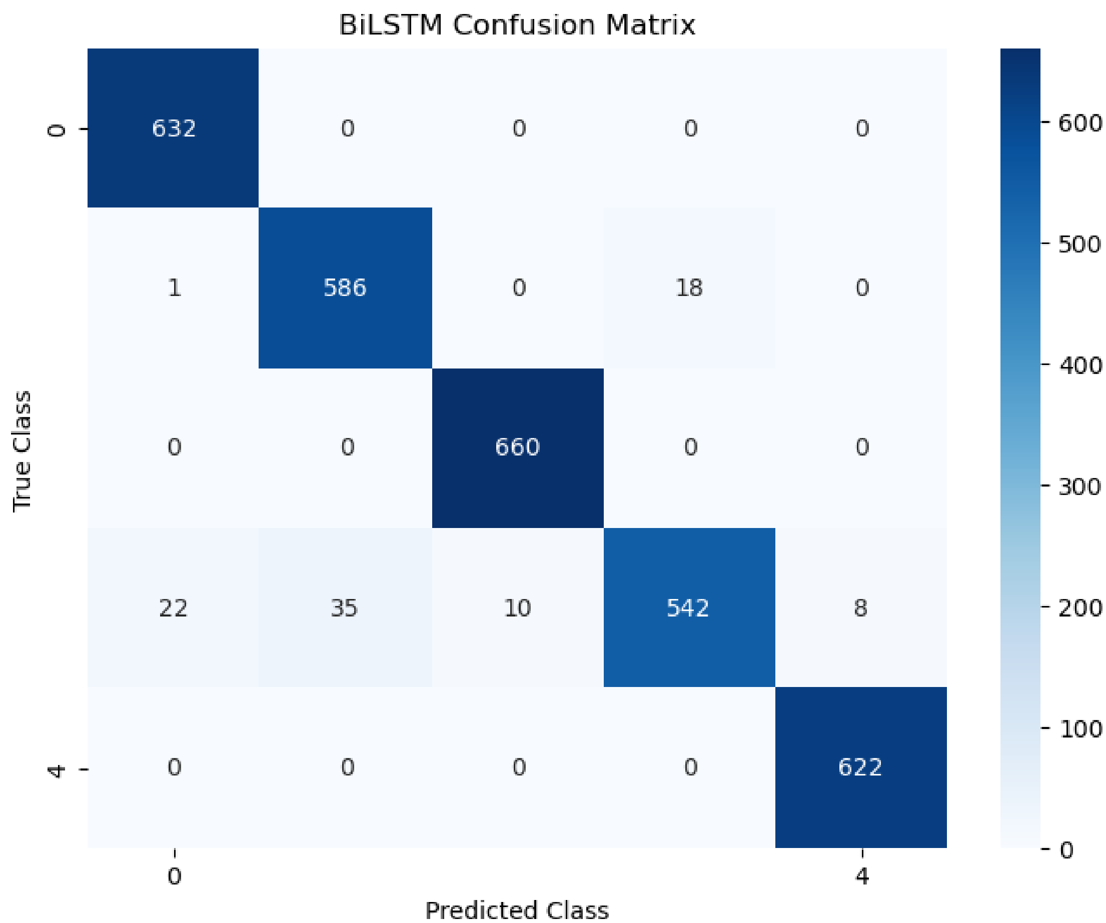
```
lstm_pred = np.argmax(lstm_model.predict(X_test), axis=-1)
bilstm_pred = np.argmax(bilstm_model.predict(X_test), axis=-1)

lstm_cm = confusion_matrix(np.argmax(y_test, axis=-1), lstm_pred)
bilstm_cm = confusion_matrix(np.argmax(y_test, axis=-1), bilstm_pred)
```

```
LSTM: Average Accuracy = 0.95
BiLSTM: Average Accuracy = 0.97
98/98          2s 24ms/step
98/98          3s 30ms/step
```

[39]:
```
# Confusion Matrix Visualization
plt.figure(figsize=(8, 6))
sns.heatmap(lstm_cm, annot=True, fmt='d', cmap='Blues', xticklabels=4,␣
↪yticklabels=4)
plt.title('LSTM Confusion Matrix')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()
```

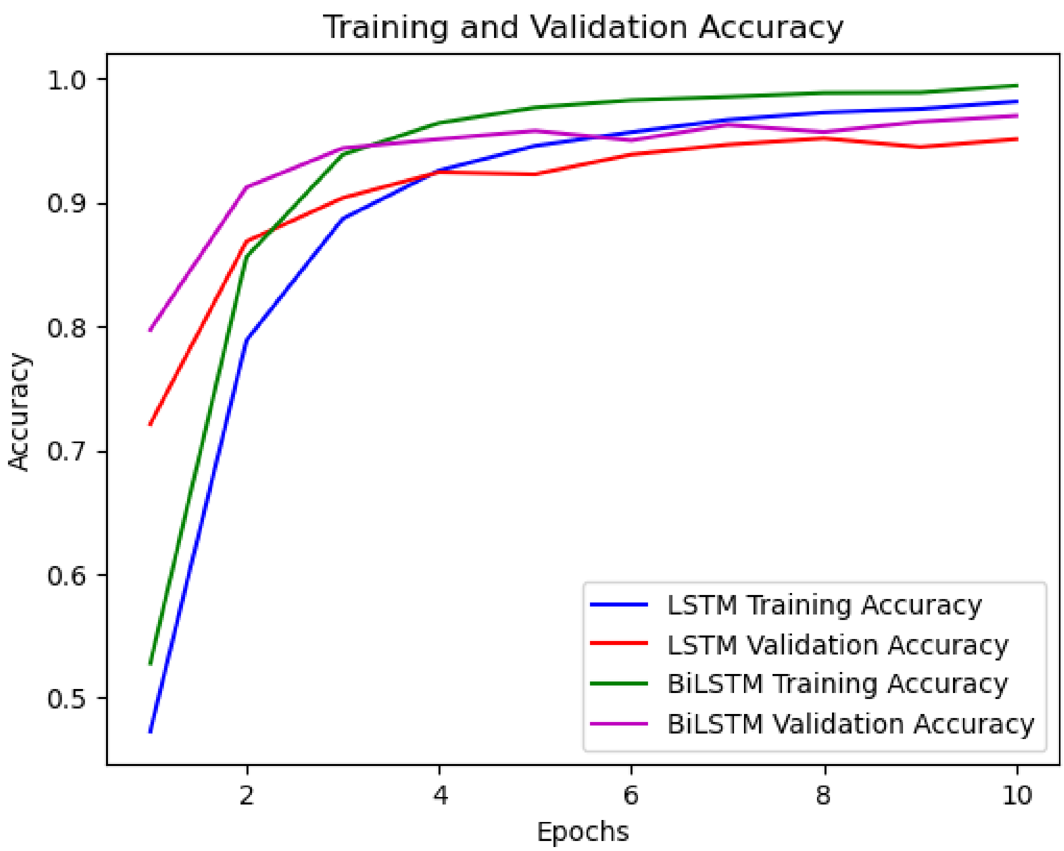


[41]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(bilstm_cm, annot=True, fmt='d', cmap='Blues', xticklabels=4,␣
  ↪yticklabels=4)
plt.title('BiLSTM Confusion Matrix')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()
```



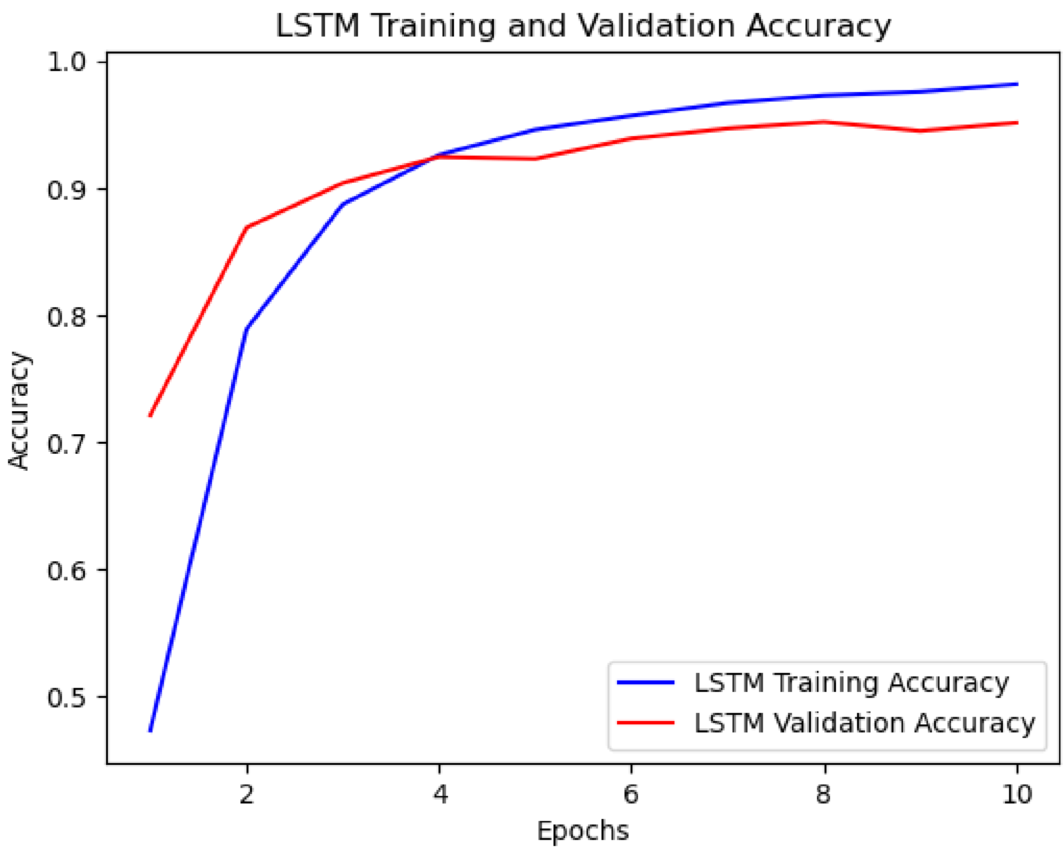
[43]:
```
# Accuracy Graph
epochs = range(1, len(lstm_history.history['accuracy']) + 1)

plt.plot(epochs, lstm_history.history['accuracy'], 'b', label='LSTM Training␣
  ↪Accuracy')
plt.plot(epochs, lstm_history.history['val_accuracy'], 'r', label='LSTM␣
  ↪Validation Accuracy')
```

```
plt.plot(epochs, bilstm_history.history['accuracy'], 'g', label='BiLSTM␣
↪Training Accuracy')
plt.plot(epochs, bilstm_history.history['val_accuracy'], 'm', label='BiLSTM␣
↪Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

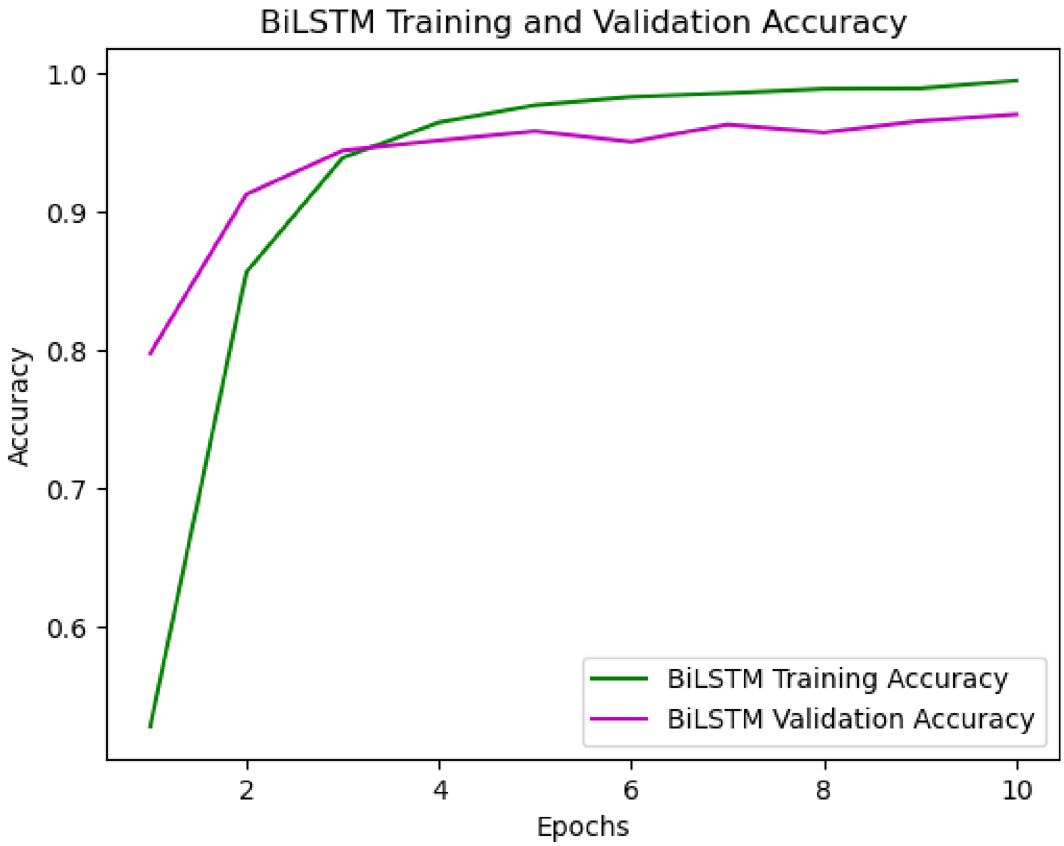


[45]:
```
# LSTM Accuracy Graph
epochs = range(1, len(lstm_history.history['accuracy']) + 1)

plt.plot(epochs, lstm_history.history['accuracy'], 'b', label='LSTM Training␣
↪Accuracy')
plt.plot(epochs, lstm_history.history['val_accuracy'], 'r', label='LSTM␣
↪Validation Accuracy')
plt.title('LSTM Training and Validation Accuracy')
plt.xlabel('Epochs')
```

```
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[47]:
```
# BiLSTM Accuracy Graph
epochs = range(1, len(bilstm_history.history['accuracy']) + 1)

plt.plot(epochs, bilstm_history.history['accuracy'], 'g', label='BiLSTM␣
↪Training Accuracy')
plt.plot(epochs, bilstm_history.history['val_accuracy'], 'm', label='BiLSTM␣
↪Validation Accuracy')
plt.title('BiLSTM Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[49]:
```
# Calculate precision, recall, and F1-score for LSTM
lstm_pred = np.argmax(lstm_model.predict(X_test), axis=-1)
lstm_report = classification_report(np.argmax(y_test, axis=-1), lstm_pred)
print("LSTM Classification Report:")
print(lstm_report)
```

```
98/98              2s 25ms/step
LSTM Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       632
           1       0.90      0.99      0.94       605
           2       0.96      1.00      0.98       660
           3       0.99      0.76      0.86       617
           4       0.97      1.00      0.99       622

    accuracy                           0.95      3136
   macro avg       0.95      0.95      0.95      3136
weighted avg       0.95      0.95      0.95      3136
```

[51]:
```python
# Calculate precision, recall, and F1-score for BiLSTM
bilstm_pred = np.argmax(bilstm_model.predict(X_test), axis=-1)
bilstm_report = classification_report(np.argmax(y_test, axis=-1), bilstm_pred)
print("BiLSTM Classification Report:")
print(bilstm_report)
```

```
98/98            5s 51ms/step
BiLSTM Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       632
           1       0.94      0.97      0.96       605
           2       0.99      1.00      0.99       660
           3       0.97      0.88      0.92       617
           4       0.99      1.00      0.99       622

    accuracy                           0.97      3136
   macro avg       0.97      0.97      0.97      3136
weighted avg       0.97      0.97      0.97      3136
```

[53]:
```python

    # Train the LSTM model on the current fold
    lstm_model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
 ↪epochs=10, batch_size=128, verbose=0)
    lstm_loss, lstm_acc = lstm_model.evaluate(X_test, y_test, verbose=0)
    lstm_acc_scores.append(lstm_acc)
```

[55]:
```python

    # Train the BiLSTM model on the current fold
    bilstm_model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
 ↪epochs=10, batch_size=128, verbose=0)
    bilstm_loss, bilstm_acc = bilstm_model.evaluate(X_test, y_test, verbose=0)
    bilstm_acc_scores.append(bilstm_acc)
```

[57]:
```python
# for train_index, test_index in kf.split(X_resampled):
#     X_train, X_test = X_resampled[train_index], X_resampled[test_index]
#     y_train, y_test = y_resampled[train_index], y_resampled[test_index]

#     # Train the LSTM model on the current fold
#     lstm_model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
 ↪epochs=10, batch_size=128, verbose=0)
#     lstm_loss, lstm_acc = lstm_model.evaluate(X_test, y_test, verbose=0)
#     lstm_acc_scores.append(lstm_acc)
```

```
#     # Train the BiLSTM model on the current fold
#     bilstm_model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
↪epochs=10, batch_size=128, verbose=0)
#     bilstm_loss, bilstm_acc = bilstm_model.evaluate(X_test, y_test, verbose=0)
#     bilstm_acc_scores.append(bilstm_acc)
```

[59]:
```
# Calculate the average accuracy scores across the k-fold cross-validation
avg_lstm_acc = np.mean(lstm_acc_scores)
avg_bilstm_acc = np.mean(bilstm_acc_scores)
```

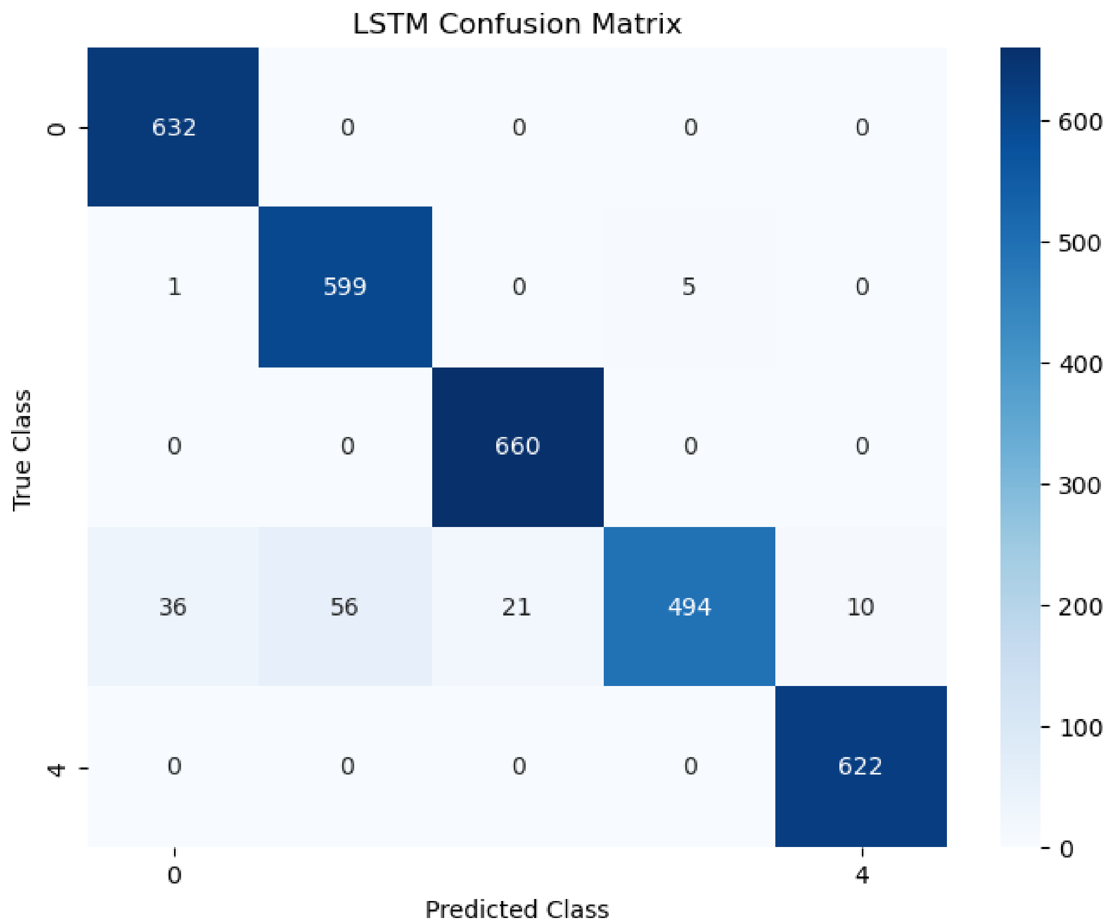
[61]:
```
print("LSTM: Average Accuracy = {:.2f}".format(avg_lstm_acc))
print("BiLSTM: Average Accuracy = {:.2f}".format(avg_bilstm_acc))
```

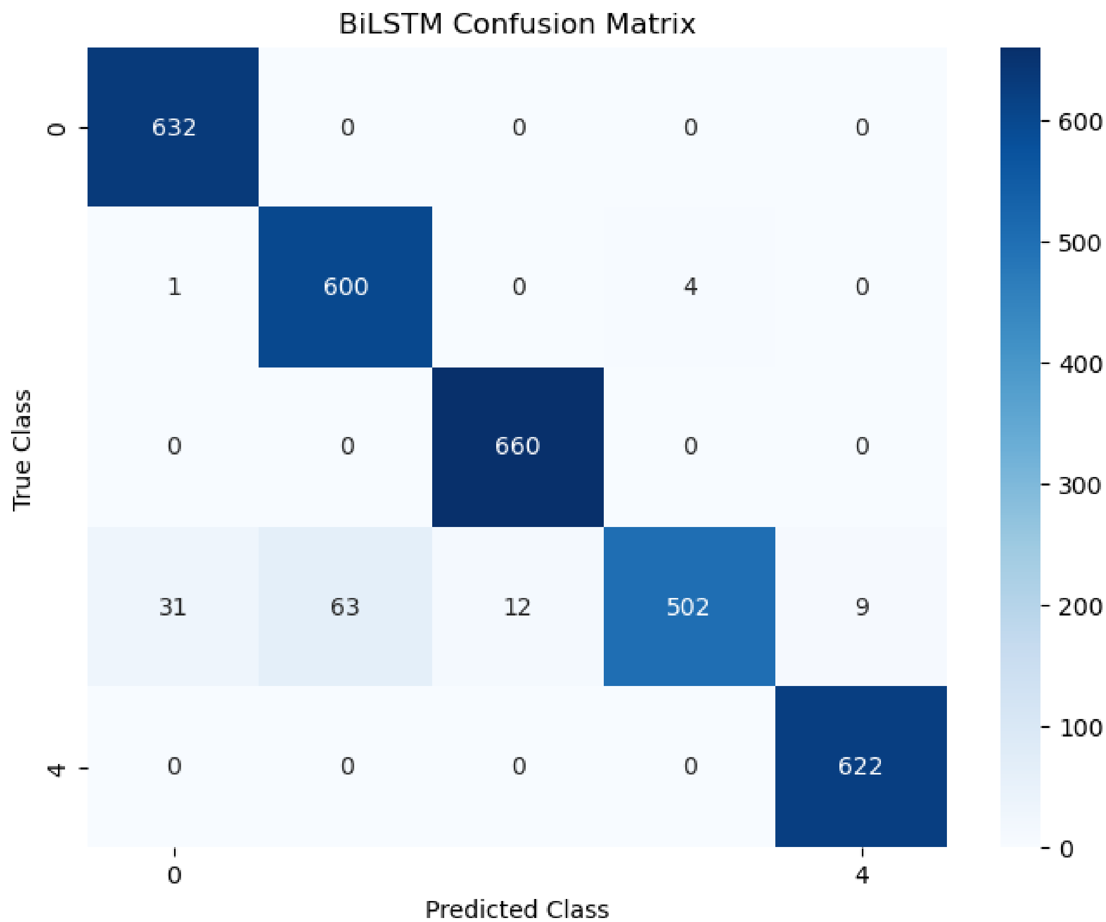
```
LSTM: Average Accuracy = 0.96
BiLSTM: Average Accuracy = 0.97
```

[63]:
```
# Confusion Matrix
lstm_pred = np.argmax(lstm_model.predict(X_test), axis=-1)
bilstm_pred = np.argmax(bilstm_model.predict(X_test), axis=-1)

lstm_cm = confusion_matrix(np.argmax(y_test, axis=-1), lstm_pred)
bilstm_cm = confusion_matrix(np.argmax(y_test, axis=-1), bilstm_pred)
```

```
98/98                 2s 25ms/step
98/98                 5s 51ms/step
```

[65]:
```
# Confusion Matrix Visualization
plt.figure(figsize=(8, 6))
sns.heatmap(lstm_cm, annot=True, fmt='d', cmap='Blues', xticklabels=4,␣
↪yticklabels=4)
plt.title('LSTM Confusion Matrix')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()

plt.figure(figsize=(8, 6))
sns.heatmap(bilstm_cm, annot=True, fmt='d', cmap='Blues', xticklabels=4,␣
↪yticklabels=4)
plt.title('BiLSTM Confusion Matrix')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()
```


LSTM Confusion Matrix
True Class
Predicted Class
632
0
1
599
5
660
36
56
21
494
10
622
4
600
500
400
300
200
100



[67]:

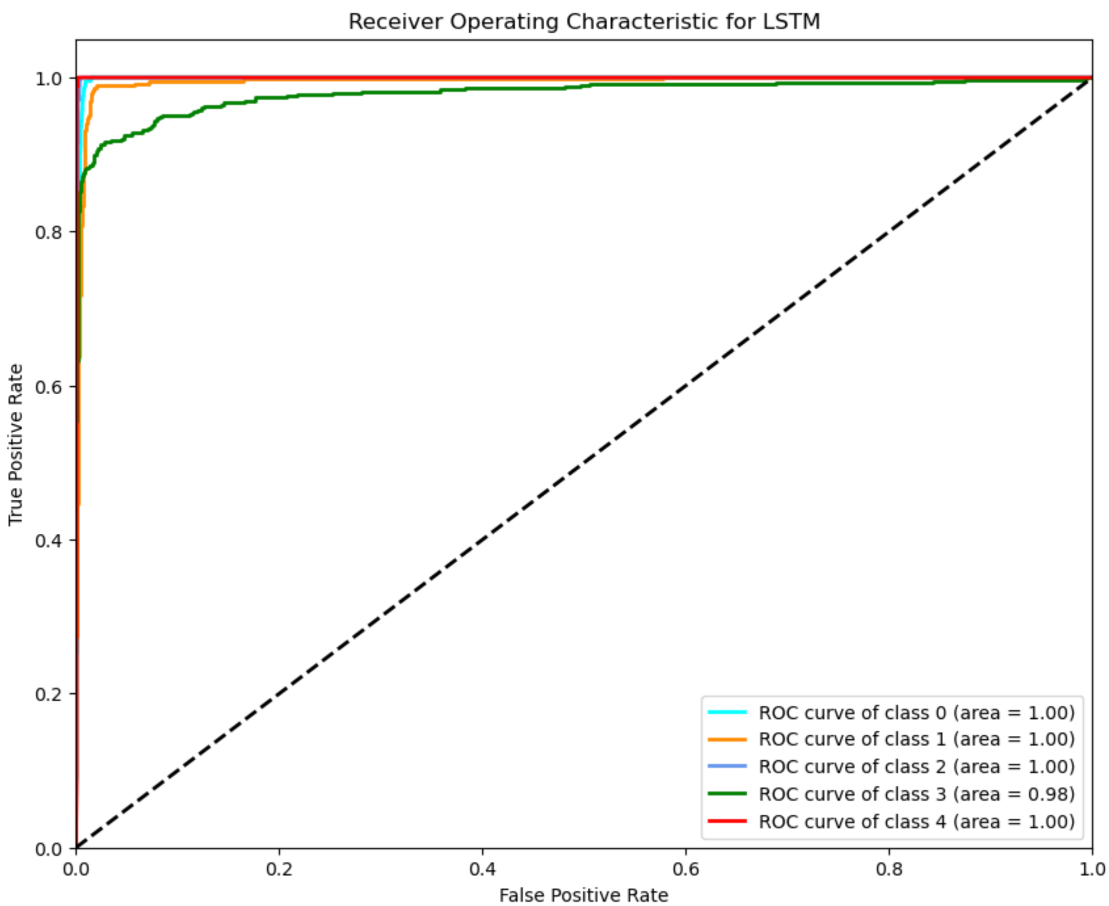
```
# Calculate the ROC curve and AUC for each class
n_classes = y_test.shape[1]

# For LSTM model
lstm_y_score = lstm_model.predict(X_test)
lstm_fpr = dict()
lstm_tpr = dict()
lstm_roc_auc = dict()

# For BiLSTM model
bilstm_y_score = bilstm_model.predict(X_test)
bilstm_fpr = dict()
bilstm_tpr = dict()
bilstm_roc_auc = dict()

for i in range(n_classes):
    lstm_fpr[i], lstm_tpr[i], _ = roc_curve(y_test[:, i], lstm_y_score[:, i])
    lstm_roc_auc[i] = auc(lstm_fpr[i], lstm_tpr[i])
```

```
    bilstm_fpr[i], bilstm_tpr[i], _ = roc_curve(y_test[:, i], bilstm_y_score[:,␣
↪i])
    bilstm_roc_auc[i] = auc(bilstm_fpr[i], bilstm_tpr[i])
```

```
98/98              2s 25ms/step
98/98              5s 51ms/step
```

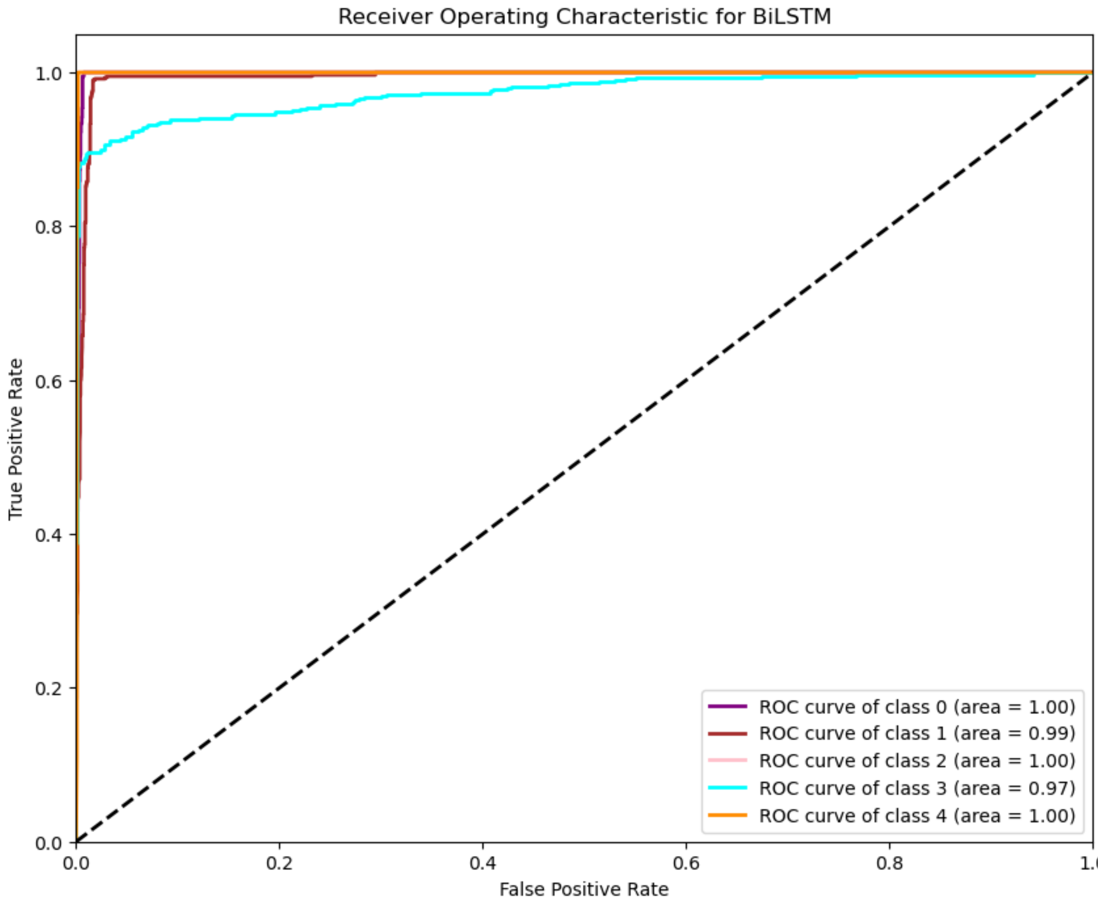
[69]:

```
# Plot the ROC curve for LSTM model
plt.figure(figsize=(10, 8))
colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'green', 'red',␣
↪'purple', 'brown', 'pink'])
for i, color in zip(range(n_classes), colors):
    plt.plot(lstm_fpr[i], lstm_tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, lstm_roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
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 for LSTM')
plt.legend(loc="lower right")
plt.show()
```



[71]:

```
# Plot the ROC curve for BiLSTM model
plt.figure(figsize=(10, 8))
for i, color in zip(range(n_classes), colors):
    plt.plot(bilstm_fpr[i], bilstm_tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, bilstm_roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
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 for BiLSTM')
plt.legend(loc="lower right")
plt.show()
```



Receiver Operating Characteristic for BiLSTM
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 1.00)
ROC curve of class 1 (area = 0.99)
ROC curve of class 2 (area = 1.00)
ROC curve of class 3 (area = 0.97)
ROC curve of class 4 (area = 1.00)

[ ]: