National College of Ireland

Sentiment Analysis of Hindi Song Lyrics using a BiLSTM Model with BERT Embeddings Configuration Manual

MSc Research Project
Data Analytics

Jay Milind Kulkarni
Student ID: x21173176

School of Computing
National College of Ireland

Supervisor: Mr. Abdul Shahid

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Jay Milind Kulkarni
Student ID:	x21173176
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Mr. Abdul Shahid
Submission Due Date:	14/08/2023
Project Title:	Sentiment Analysis of Hindi Song Lyrics using a BiLSTM Model with BERT Embeddings Configuration Manual
Word Count:	877
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Jay Milind Kulkarni
Date:	13th August 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Sentiment Analysis of Hindi Song Lyrics using a BiLSTM Model with BERT Embeddings Configuration Manual

Jay Milind Kulkarni
x21173176

1 Introduction

This configuration manual is a guide for the research implementation of "Sentiment Analysis of Hindi Song Lyrics using a BiLSTM Model with BERT Embeddings". The detailed steps for the procedures followed for System and environment setup, Data loading, Data Process, and Modeling are discussed. Also, code snippets are appended as per code file sequences. There are a total of four Python files, the first one is "EDA.ipynb" which contains code logic for Explanatory Data Analysis, and the remaining three files are for each model and the file names are after each model respectively.

2 System Configuration

Hardware and Software requirements for conducting the research are specified in this section

2.1 Hardware Specifications

- Operating System: Windows 11
- HP Envy
- RAM: 16 GB
- 256 GB SSD

2.2 Software Specifications

- Python Version 3.10.12
- Google Colab
- Overleaf

Python was chosen as the programming language for implementing this research project. All the processes such as Data loading, Data Cleaning, Exploratory Data Analysis, Model Building, and evaluation were implemented using Python.

3 Data Collection

The dataset was downloaded from Kaggle and it's called "Hindi Songs Lyrics With Artists" [1]. Below Figure 1 is a snapshot of the website.

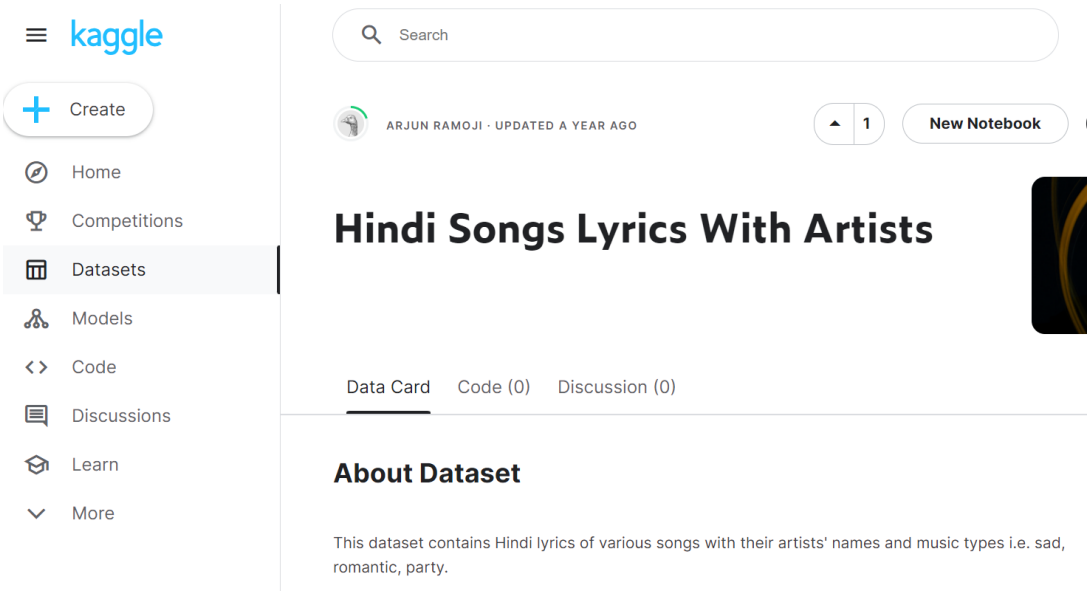


Figure 1: Kaggle Website Dataset Snapshot

4 Research Project Code

There are totally four Python files that were created in this project, the first one is named EDA after Exploratory Data Analysis, and the other files are named after each model.

1. EDA.ipynb: This file consists of code for data pre-processing and EDA visualizations.
2. BERT-BilSTM Model With Stopwords.ipynb: This is the first model where the stopword removal step was skipped to analyze the impact of stopwords on the model.
3. BERT-BilSTM Model Without Stopwords.ipynb: In this code stopword removal along with parameter changes were introduced.
4. BERT-BilSTM Model Without Stopwords and K-Fold Cross Validation.ipynb: This is the final file which is the last model which performed better than the other two models.

5 Python packages and Libraries Used

Following is the list of Python libraries used for this project:

[1] https://www.kaggle.com/datasets/arjunramoji/hindi-songs-lyrics-with-artists

1. Pandas: Library used for data manipulation and analysis of tabular data.
2. Collections: For this project counter package is used from this library for counting elements.
3. Wordcloud: Used for creating a word cloud of Hindi text data.
4. Matplotlib: This library is used for creating visualizations.
5. Advertools: Hindi stopwords lists were extracted from the stopwords package of this library.
6. Seaborn: Used for visualizations
7. Numpy: Library used for numerical computation of arrays or matrices.
8. Sklearn: This is an important library that is used for data processing, model selection, and evaluation.
9. Torch: A machine learning framework that has tensors and dynamic computational resources.
10. Transformers: Library for BERT model for Natural Language processing.

6 Steps for Code Understanding and Execution

In this section steps for code from each python file are discussed.

6.1 EDA Code File

1. First step is to install advertools library using the command "pip install advertools"
2. Next import libraries such as pandas, collections, word cloud, matplotlib, advertools, and seaborn.
3. Google colab is used, so the dataset is stored in the drive and then this drive is mounted and data is accessed.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/songs.csv')
    df
```

	index	Song name	type	artist	lyrics
0	1	नीले नीले अम्बर पर	romantic	किशोर कुमार	नीले नीले अम्बर पर चाँद जब आये प्यार बरसाए हमक...
1	2	अक्कड़ बक्कड़	party	बादशाह	अक्कड़ बक्कड़ बॉम्बे बो 80, 90 पुरे 100 रात के...
2	3	अखियाँ	sad	पोपोन	ओ थक गेया अँखियाँ ओ जग्ग दियाँ अखियाँ माहियाँ ...
3	4	अंग से अंग लगाना	romantic	अलका याग्निक, बिनोद राठौड़, सुदेश भोसले	अरे जो जी में आए.. अरे जो जी में आए.. तुम आज ...
4	5	अगर ज़िन्दगी हो	romantic	आशा भोसले	अगर ज़िन्दगी हो तो तेरे संग हो अगर ज़िन्दगी हो...
...	...	...	...	...	...
788	789	हैप्पी हैप्पी	party	बादशाह	विंटर का महीना उस पर तुझ जैसी हसीना बोलो फिर ...
789	790	हो गया है तुझको	romantic	लता मंगेशकर,उदित नारायण	आई अब की साल दिवाली मुंह पर अपने खून मले आई अ...
790	791	होंठों से छू लो तुम	romantic	जगजीत सिंह	होंठों से छू लो तुम मेरा गीत अमर कर दो होंठों ...
791	792	होली के दिन	party	किशोर कुमार	चलो सहेली, चलो रे साथी, चलो सहेली, चलो रे स...

Figure 2: Code for Mounting Drive and Accessing Dataset

4. In the next step function to remove stopwords from lyrics data is written by using a Hindi stopword list from advertools library.

```
[ ] # Geting Hindi stopwords from advertools
    hindi_text_stopwords = set(adv.stopwords['hindi'])

[ ] # Function to remove hindi stopwords
    def remove_hindi_stopwords(text):
        words = text.split()
        filtered_hindi_words = [word for word in words if word not in hindi_text_stopwords]
        return ' '.join(filtered_hindi_words)

[ ] Final_df['lyrics'] = Final_df['lyrics'].apply(remove_hindi_stopwords)

[ ] sns.countplot(x='type', data=Final_df)
    plt.title('Count of Songs in Each Type')
    plt.show()
```

Figure 3: Stopwords Removal and Plotting Count

5. Once the data cleaning is done, now plots of class count and word counts are created and Figure 3 is the snapshot of code for this.

6. The final step in the EDA file is to create a word cloud for the most common words from the lyrics of each class.

```
# Preprocessing hindi text and logic to find most common words for each class
grouped_text = Final_df_count.groupby('type')['lyrics'].apply(lambda x: ' '.join(x))
hindi_word_frequencies = {}
for class_name, lyrics in grouped_text.items():
    words = lyrics.split()
    most_common = [word for word, count in Counter(words).most_common(300)]
    hindi_word_frequencies[class_name] = most_common
```

```
# Generating word clouds for each class
for class_name, common_words in hindi_word_frequencies.items():
    words_text = ' '.join(common_words)
    wordcloud = WordCloud(
        width=800,
        height=800,
        background_color='white',
        font_path='/content/drive/MyDrive/Colab Notebooks/gargi.ttf',  # Replace with the path to a Hindi font
        contour_color='steelblue'
    ).generate(words_text)

    plt.figure(figsize=(8, 8))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f"Word Cloud for {class_name} Lyrics")
    plt.axis('off')
    plt.show()
```

Figure 4: WordCloud with Most Common Words

6.2 BERT-BilSTM Model With Stopwords File

1. Initial Stages until loading the data is same as EDA file and install “transformers” library.

2. Next is to skip the step of removing stopwords logic and prepare the data for creating BERT Embeddings.

3. Write a function for creating BERT embeddings.

4. Once the data is converted into word embeddings the predictor column “type” is also converted into label using label encoders into values 0,1 and 2.

```python
# Preparing the data - BERT embeddings
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-uncased')
bert_model = BertModel.from_pretrained('bert-base-multilingual-uncased')
```

```
Downloading (…)solve/main/vocab.txt: 100% 872k/872k [00:00<00:00, 10.5MB/s]
Downloading (…)okenizer_config.json: 100% 28.0/28.0 [00:00<00:00, 1.06kB/s]
Downloading (…)lve/main/config.json: 100% 625/625 [00:00<00:00, 27.0kB/s]
Downloading model.safetensors: 100% 672M/672M [00:04<00:00, 214MB/s]
```

```python
def get_bert_embeddings(lyrics):
    inputs = tokenizer(lyrics, return_tensors='pt', padding=True, truncation=True)
    with torch.no_grad():
        outputs = bert_model(**inputs)
    return outputs.last_hidden_state.mean(dim=1)
```

```python
Final_df['bert_embeddings'] = Final_df['lyrics'].apply(get_bert_embeddings)
```

Figure 5: Creating BERT Embeddings

```python
[ ] # Preparing the labels by Encoding 'type' column
    label_encoder = LabelEncoder()
    Final_df['label'] = label_encoder.fit_transform(Final_df['type'])
```

```python
[ ] Final_df
```

	lyrics	type	bert_embeddings	label
0	नीले नीले अम्बर पर चाँद जब आये प्यार बरसाए हमक...	romantic	[[tensor(-0.2260), tensor(-0.4654), tensor(0.3...	1
1	अक्कड़ बक्कड़ बॉम्बे बो 80, 90 पुरे 100 रात के...	party	[[tensor(-0.2822), tensor(-0.5775), tensor(0.2...	0
2	ओ थक गेया अँखियाँ ओ जग्ग दियाँ अखियाँ माहियाँ ...	sad	[[tensor(-0.2493), tensor(-0.3941), tensor(0.3...	2
3	अरे जो जी में आए.. अरे जो जी में आए.. तुम आज ...	romantic	[[tensor(-0.1516), tensor(-0.3722), tensor(0.2...	1
4	अगर ज़िन्दगी हो तो तेरे संग हो अगर ज़िन्दगी हो...	romantic	[[tensor(-0.2647), tensor(-0.3406), tensor(0.1...	1
...	...	...	...	...
788	विंटर का महीना उस पर तुझ जैसी हसीना बोलो फिर ...	party	[[tensor(-0.2065), tensor(-0.2011), tensor(0.2...	0
789	आई अब की साल दिवाली मुंह पर अपने खून मले आई अ...	romantic	[[tensor(-0.1495), tensor(-0.3169), tensor(0.2...	1
790	होंठों से छू लो तुम मेरा गीत अमर कर दो होंठों ...	romantic	[[tensor(-0.0878), tensor(-0.2994), tensor(0.2...	1
791	चलो सहेली.. चलो रे साथी.. चलो सहेली.. चलो रे स...	party	[[tensor(-0.2320), tensor(-0.4417), tensor(0.2...	0
792	हम्म.. आ.. हा हा हा.. होशवालों को खबर क्या बेख...	sad	[[tensor(-0.3214), tensor(-0.4544), tensor(0.3...	2

Figure 6: Creating Label encoders

```python
# Converting lists to PyTorch tensors and create a DataLoader
train_data = TensorDataset(torch.stack(X_train), torch.tensor(y_train))
train_loader = DataLoader(train_data, batch_size=10, shuffle=True)
```

```python
# Build the BERT embeddings-BiLSTM model
class BERTBiLSTMModel(nn.Module):
    def __init__(self, lstm_hidden_size, num_classes):
        super(BERTBiLSTMModel, self).__init__()
        self.bert_embeddings_dim = 768
        self.lstm_hidden_size = lstm_hidden_size
        self.num_classes = num_classes

        self.lstm = nn.LSTM(self.bert_embeddings_dim, lstm_hidden_size, batch_first=True, bidirectional=True)
        self.linear = nn.Linear(2 * lstm_hidden_size, num_classes)

    def forward(self, x):
        lstm_out, _ = self.lstm(x)
        lstm_out = lstm_out[:, -1, :]
        logits = self.linear(lstm_out)
        return logits
```

Figure 7: Buliding a BERT-BiLSTM Model

5. Next step is to divide the data into train and test data and convert to Pytorch tensors and create a data loader and build a BERT-BiLSTM model.

6. Finally, the built model is trained without introducing weights to classes.

```
[ ]  # Training the model on training data
     device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

[ ]  model = BERTBiLSTMModel(lstm_hidden_size=128, num_classes=3).to(device)

[ ]  criterion = nn.CrossEntropyLoss()

[ ]  optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

[ ]  epochs = 50
     for epoch in range(epochs):
         model.train()
         total_loss = 0
         for inputs, labels in train_loader:
             inputs, labels = inputs.to(device), labels.to(device)
             optimizer.zero_grad()
             logits = model(inputs)
             loss = criterion(logits, labels)
             loss.backward()
             optimizer.step()
             total_loss += loss.item()

         print(f'Epoch {epoch + 1}/{epochs}, Loss: {total_loss / len(train_loader):.4f}')
```

Figure 8: BERT-BiLSTM Model Training

```
Epoch 1/50, Loss: 1.0082
Epoch 2/50, Loss: 0.8296
Epoch 3/50, Loss: 0.7283
Epoch 4/50, Loss: 0.7004
Epoch 5/50, Loss: 0.6369
Epoch 6/50, Loss: 0.6407
Epoch 7/50, Loss: 0.5924
Epoch 8/50, Loss: 0.5590
Epoch 9/50, Loss: 0.5336
Epoch 10/50, Loss: 0.5432
Epoch 11/50, Loss: 0.5716
Epoch 12/50, Loss: 0.5092
Epoch 13/50, Loss: 0.4840
Epoch 14/50, Loss: 0.5047
Epoch 15/50, Loss: 0.4490
Epoch 16/50, Loss: 0.4608
Epoch 17/50, Loss: 0.4601
Epoch 18/50, Loss: 0.3906
Epoch 19/50, Loss: 0.3756
Epoch 20/50, Loss: 0.3594
Epoch 21/50, Loss: 0.3399
Epoch 22/50, Loss: 0.3425
Epoch 23/50, Loss: 0.3916
Epoch 24/50, Loss: 0.2973
Epoch 25/50, Loss: 0.2536
Epoch 26/50, Loss: 0.2625
Epoch 27/50, Loss: 0.3524
Epoch 28/50, Loss: 0.2110
```

Figure 9: BERT-BiLSTM Model Training

7. Once the model is built and trained, the next and final step is to evaluate the model. Evaluation parameters are Accuracy, Precision, Recall, and AUC.

```
[ ] # Evaluating the model on the testing data
    model.eval()
    with torch.no_grad():
        X_test_tensor = torch.stack(X_test).to(device)
        y_test_tensor = torch.tensor(y_test).to(device)
        y_pred_probs = torch.softmax(model(X_test_tensor), dim=1)
        y_pred = torch.argmax(y_pred_probs, dim=1).cpu().numpy()

[ ] accuracy = accuracy_score(y_test, y_pred)

[ ] print(f'Accuracy: {accuracy:.4f}')

    Accuracy: 0.5597

[ ] auc = roc_auc_score(y_test, y_pred_probs.cpu().numpy(), multi_class='ovr')

[ ] print(f'AUC: {auc:.4f}')

    AUC: 0.7700

[ ] conf_matrix = confusion_matrix(y_test, y_pred)
```

Figure 10: BERT-BiLSTM Model Evaluation

```
[ ] conf_matrix = confusion_matrix(y_test, y_pred)

[ ] print('Confusion Matrix:')
    print(conf_matrix)

    Confusion Matrix:
    [[21  3  3]
     [14 44 33]
     [ 2 15 24]]

[ ] precision = precision_score(y_test, y_pred, average=None)

[ ] recall = recall_score(y_test, y_pred, average=None)

[ ] def calculate_individual_accuracy(confusion_matrix):
        num_classes = len(confusion_matrix)
        individual_accuracy = []
        for i in range(num_classes):
            individual_accuracy.append(confusion_matrix[i, i] / confusion_matrix[i, :].sum())
        return individual_accuracy
```

Figure 11: BERT-BiLSTM Model Evaluation

```
[ ] def calculate_individual_accuracy(confusion_matrix):
        num_classes = len(confusion_matrix)
        individual_accuracy = []
        for i in range(num_classes):
            individual_accuracy.append(confusion_matrix[i, i] / confusion_matrix[i, :].sum())
        return individual_accuracy

[ ] individual_accuracy = calculate_individual_accuracy(conf_matrix)

[ ] precision

    array([0.56756757, 0.70967742, 0.4       ])

[ ] recall

    array([0.77777778, 0.48351648, 0.58536585])

[ ] individual_accuracy

    [0.7777777777777778, 0.4835164835164835, 0.5853658536585366]
```

Figure 12: BERT-BiLSTM Model Evaluation

6.3 BERT-BilSTM Model Without Stopwords

1. Initial steps are the combination of the EDA file and BERT-BilSTM Model With Stopwords file i.e., data loading, stopwords removal, creating BERT embeddings and tensors.

2. During the model-building stage few modifications are done for improving model performances.

```
# Build the BERT embeddings-BiLSTM model
class BERTBiLSTMModel(nn.Module):
    def __init__(self, lstm_hidden_size, num_classes, dropout_rate=0.2):
        super(BERTBiLSTMModel, self).__init__()
        self.bert_embeddings_dim = 768
        self.lstm_hidden_size = lstm_hidden_size
        self.num_classes = num_classes

        self.lstm = nn.LSTM(self.bert_embeddings_dim, lstm_hidden_size, batch_first=True, bidirectional=True)
        self.dropout = nn.Dropout(dropout_rate)
        self.linear = nn.Linear(2 * lstm_hidden_size, num_classes)

    def forward(self, x):
        lstm_out, _ = self.lstm(x)
        lstm_out = lstm_out[:, -1, :]
        lstm_out = self.dropout(lstm_out)
        logits = self.linear(lstm_out)
        return logits
```

Figure 13: BERT-BiLSTM Model Buliding

3. Now the model is trained and class weights, learning rate, and weight decay are introduced.

```
[ ] # Training the model on training data
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

[ ] model = BERTBiLSTMModel(lstm_hidden_size=200, num_classes=3).to(device)

[ ] class_weights = torch.tensor([1.9, 1, 2.6]).to(device)

[ ] criterion = nn.CrossEntropyLoss(weight=class_weights)

[ ] optimizer = torch.optim.Adam(model.parameters(), lr=0.0001,weight_decay=0.001)

[ ] scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
```

Figure 14: BERT-BiLSTM Model Training

```
epochs = 80
for epoch in range(epochs):
    model.train()
    total_loss = 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        logits = model(inputs)
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    print(f'Epoch {epoch + 1}/{epochs}, Loss: {total_loss / len(train_loader):.4f}')
```

Figure 15: BERT-BiLSTM Model Training

```
# Evaluating the model on the testing data
model.eval()
with torch.no_grad():
    X_test_tensor = torch.stack(X_test).to(device)
    y_test_tensor = torch.tensor(y_test).to(device)
    y_pred_probs = torch.softmax(model(X_test_tensor), dim=1)
    y_pred = torch.argmax(y_pred_probs, dim=1).cpu().numpy()
```

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f'Accuracy: {accuracy:.4f}')
```

```
Accuracy: 0.6639
```

```
loss = criterion(model(X_test_tensor), y_test_tensor).item()
```

```
print(f'Loss: {loss:.4f}')
```

```
Loss: 1.0644
```

```
auc = roc_auc_score(y_test, y_pred_probs.cpu().numpy(), multi_class='ovr')
```

Figure 16: BERT-BiLSTM Model Evaluation

```
print(f'AUC: {auc:.4f}')
```

```
AUC: 0.7658
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
print('Confusion Matrix:')
print(conf_matrix)
```

```
Confusion Matrix:
[[36 12  2]
 [ 7 99 28]
 [ 4 27 23]]
```

```
precision = precision_score(y_test, y_pred, average=None)
```

```
recall = recall_score(y_test, y_pred, average=None)
```

Figure 17: BERT-BiLSTM Model Evaluation

```
def calculate_individual_accuracy(confusion_matrix):
    num_classes = len(confusion_matrix)
    individual_accuracy = []
    for i in range(num_classes):
        individual_accuracy.append(confusion_matrix[i, i] / confusion_matrix[i, :].sum())
    return individual_accuracy
```

```
individual_accuracy = calculate_individual_accuracy(conf_matrix)
```

```
precision
```

```
array([0.76595745, 0.7173913 , 0.43396226])
```

```
recall
```

```
array([0.72      , 0.73880597, 0.42592593])
```

```
individual_accuracy
```

```
[0.72, 0.7388059701492538, 0.42592592592592593]
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix)
```

Figure 18: BERT-BiLSTM Model Evaluation

6.4 BERT-BilSTM Model Without Stopwords and K-Fold Cross Validation

1. First few steps until BERT embeddings are the same as before for this file as well.
2. This is the last model where along with the class weights, k-fold cross-validation is also introduced.

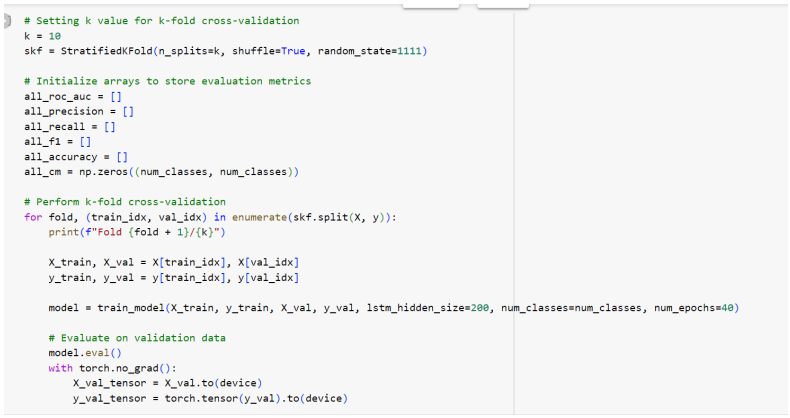
```
# Build the BERT embeddings-BiLSTM model
class BERTBiLSTMModel(nn.Module):
    def __init__(self, lstm_hidden_size, num_classes, dropout_rate=0.2):
        super(BERTBiLSTMModel, self).__init__()
        self.bert_embeddings_dim = 768
        self.lstm_hidden_size = lstm_hidden_size
        self.num_classes = num_classes

        self.lstm = nn.LSTM(self.bert_embeddings_dim, lstm_hidden_size, batch_first=True, bidirectional=True)
        self.dropout = nn.Dropout(dropout_rate)
        self.linear = nn.Linear(2 * lstm_hidden_size, num_classes)

    def forward(self, x):
        lstm_out, _ = self.lstm(x)
        lstm_out = lstm_out[:, -1, :]
        lstm_out = self.dropout(lstm_out)
        logits = self.linear(lstm_out)
        return logits
```

```
X = Final_df['bert_embeddings'].tolist()
X = torch.stack(X)
y = Final_df['label'].values

num_classes = len(np.unique(y))
```

Figure 19: BERT-BiLSTM Model Building

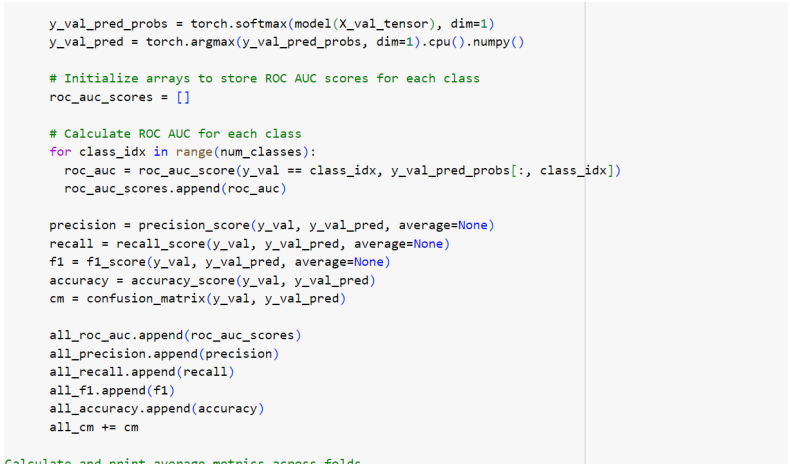
```
def train_model(X_train, y_train, X_val, y_val, lstm_hidden_size, num_classes, num_epochs=40):
    model = BERTBiLSTMModel(lstm_hidden_size=lstm_hidden_size, num_classes=num_classes).to(device)
    class_weights = torch.tensor([1.9, 1.8, 15]).to(device)
    criterion = nn.CrossEntropyLoss(weight=class_weights)
    #optimizer = torch.optim.Adam(model.parameters(), lr=0.0001, weight_decay=0.001)
    #scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
    #scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)

    train_data = TensorDataset(X_train, torch.tensor(y_train))
    train_loader = DataLoader(train_data, batch_size=4, shuffle=True)

    for epoch in range(num_epochs):
        model.train()
        total_loss = 0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            logits = model(inputs)
            loss = criterion(logits, labels)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

        print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {total_loss / len(train_loader):.4f}')

    return model
```

Figure 20: BERT-BiLSTM Model Training



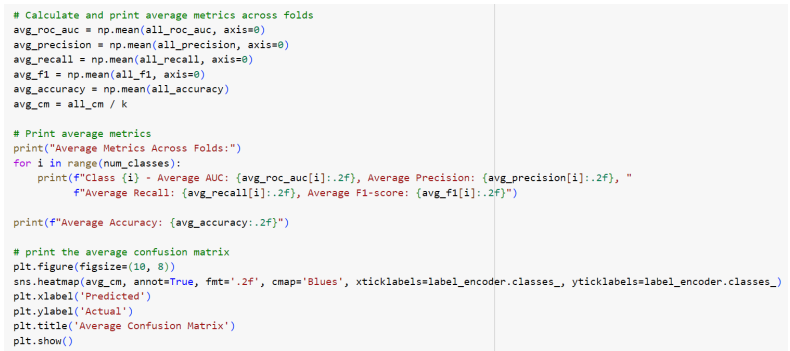
```
# Setting k value for k-fold cross-validation
k = 10
skf = StratifiedKFold(n_splits=k, shuffle=True, random_state=1111)

# Initialize arrays to store evaluation metrics
all_roc_auc = []
all_precision = []
all_recall = []
all_f1 = []
all_accuracy = []
all_cm = np.zeros((num_classes, num_classes))

# Perform k-fold cross-validation
for fold, (train_idx, val_idx) in enumerate(skf.split(X, y)):
    print(f"Fold {fold + 1}/{k}")

    X_train, X_val = X[train_idx], X[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]

    model = train_model(X_train, y_train, X_val, y_val, lstm_hidden_size=200, num_classes=num_classes, num_epochs=40)

    # Evaluate on validation data
    model.eval()
    with torch.no_grad():
        X_val_tensor = X_val.to(device)
        y_val_tensor = torch.tensor(y_val).to(device)
```

Figure 21: BERT-BiLSTM Model Evaluation



```
    y_val_pred_probs = torch.softmax(model(X_val_tensor), dim=1)
    y_val_pred = torch.argmax(y_val_pred_probs, dim=1).cpu().numpy()

    # Initialize arrays to store ROC AUC scores for each class
    roc_auc_scores = []

    # Calculate ROC AUC for each class
    for class_idx in range(num_classes):
      roc_auc = roc_auc_score(y_val == class_idx, y_val_pred_probs[:, class_idx])
      roc_auc_scores.append(roc_auc)

    precision = precision_score(y_val, y_val_pred, average=None)
    recall = recall_score(y_val, y_val_pred, average=None)
    f1 = f1_score(y_val, y_val_pred, average=None)
    accuracy = accuracy_score(y_val, y_val_pred)
    cm = confusion_matrix(y_val, y_val_pred)

    all_roc_auc.append(roc_auc_scores)
    all_precision.append(precision)
    all_recall.append(recall)
    all_f1.append(f1)
    all_accuracy.append(accuracy)
    all_cm += cm

Calculate and print average metrics across folds
```

Figure 22: BERT-BiLSTM Model Evaluation



```
# Calculate and print average metrics across folds
avg_roc_auc = np.mean(all_roc_auc, axis=0)
avg_precision = np.mean(all_precision, axis=0)
avg_recall = np.mean(all_recall, axis=0)
avg_f1 = np.mean(all_f1, axis=0)
avg_accuracy = np.mean(all_accuracy)
avg_cm = all_cm / k

# Print average metrics
print("Average Metrics Across Folds:")
for i in range(num_classes):
    print(f"Class {i} - Average AUC: {avg_roc_auc[i]:.2f}, Average Precision: {avg_precision[i]:.2f}, "
          f"Average Recall: {avg_recall[i]:.2f}, Average F1-score: {avg_f1[i]:.2f}")

print(f"Average Accuracy: {avg_accuracy:.2f}")

# print the average confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(avg_cm, annot=True, fmt='.2f', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Average Confusion Matrix')
plt.show()
```

Figure 23: BERT-BiLSTM Model Evaluation