shap-and-lime-for-models

May 15, 2024

```
[ ]: !pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-
packages (4.35.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.19.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.15.0)
Requirement already satisfied: safetensors>=0.3.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.4.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-
packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.16.4->transformers) (2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.16.4->transformers) (4.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->transformers) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
```

(2023.11.17)

[ ]:
```
import numpy as np
import re
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, TFBertForSequenceClassification
from transformers import TFBertForSequenceClassification
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Bidirectional, GRU, Dropout, Dense
from tensorflow.keras.layers import Embedding
%matplotlib inline
```

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

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

[ ]:
```
df = pd.read_excel('/content/drive/MyDrive/Datasets/Bengali_comment.xlsx')
df.head()
```

[ ]:
```
                                              comment    Category   Gender  \
0                               …        Actor  Female
1                     ?          …      Singer    Male
2                            ,          ????       Actor   Female
3                                                  Sports     Male
4                                                     Politician     Male

   comment react number      label
0                   1.0     sexual
1                   2.0  not bully
2                   2.0  not bully
3                   0.0  not bully
4                   0.0      troll
```

[ ]:
```
df.isnull().sum()
```

[ ]:
```
comment                 0
Category                0
Gender                  0
comment react number    3
label                   0
```

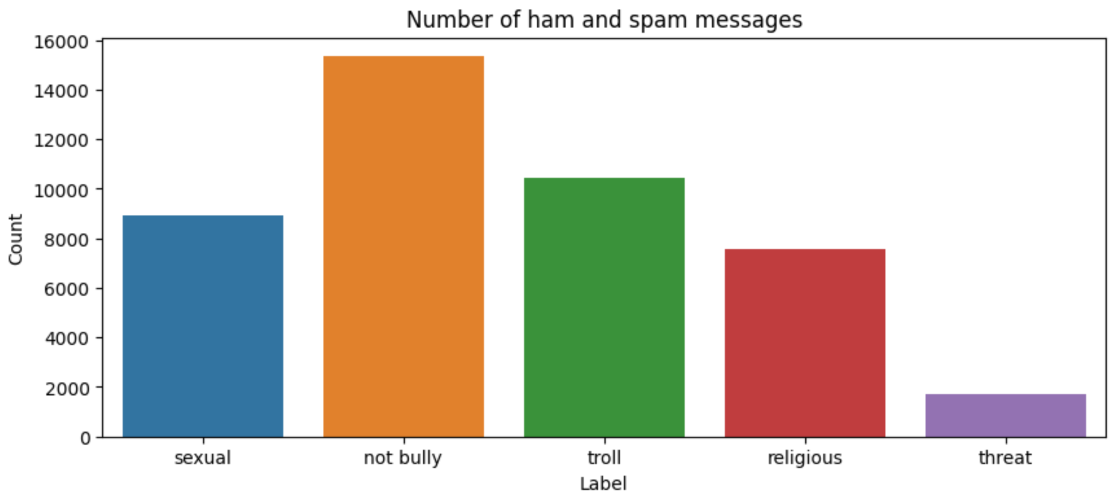
```
dtype: int64
```

[ ]:
```
df.dropna(inplace=True)
```

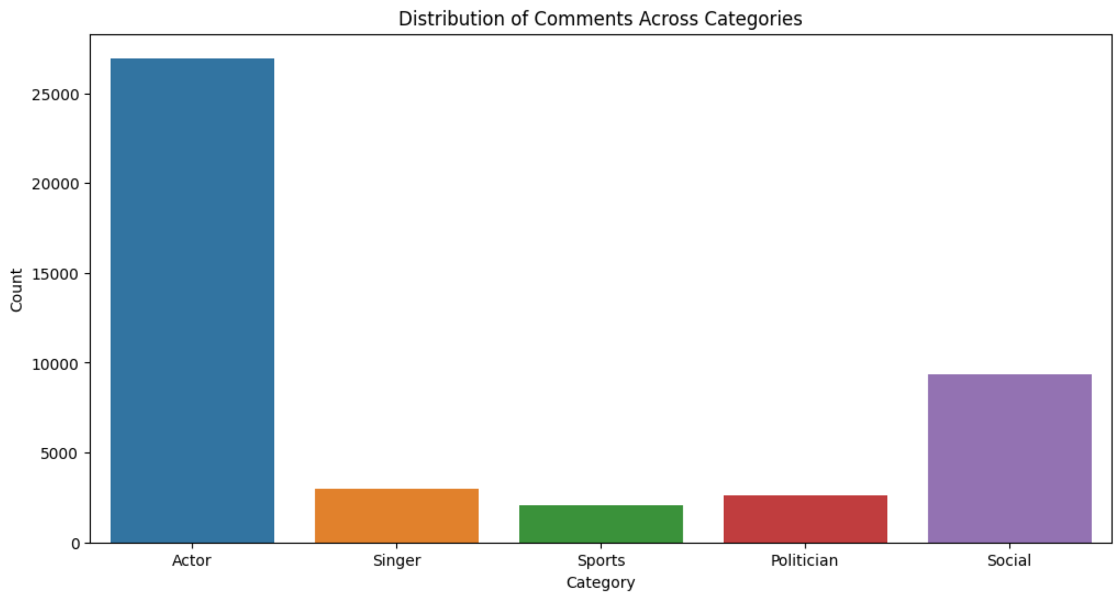
[ ]:
```
df['label'].value_counts()
```

[ ]:
```
not bully    15339
troll        10462
sexual        8928
religious     7575
threat        1694
Name: label, dtype: int64
```

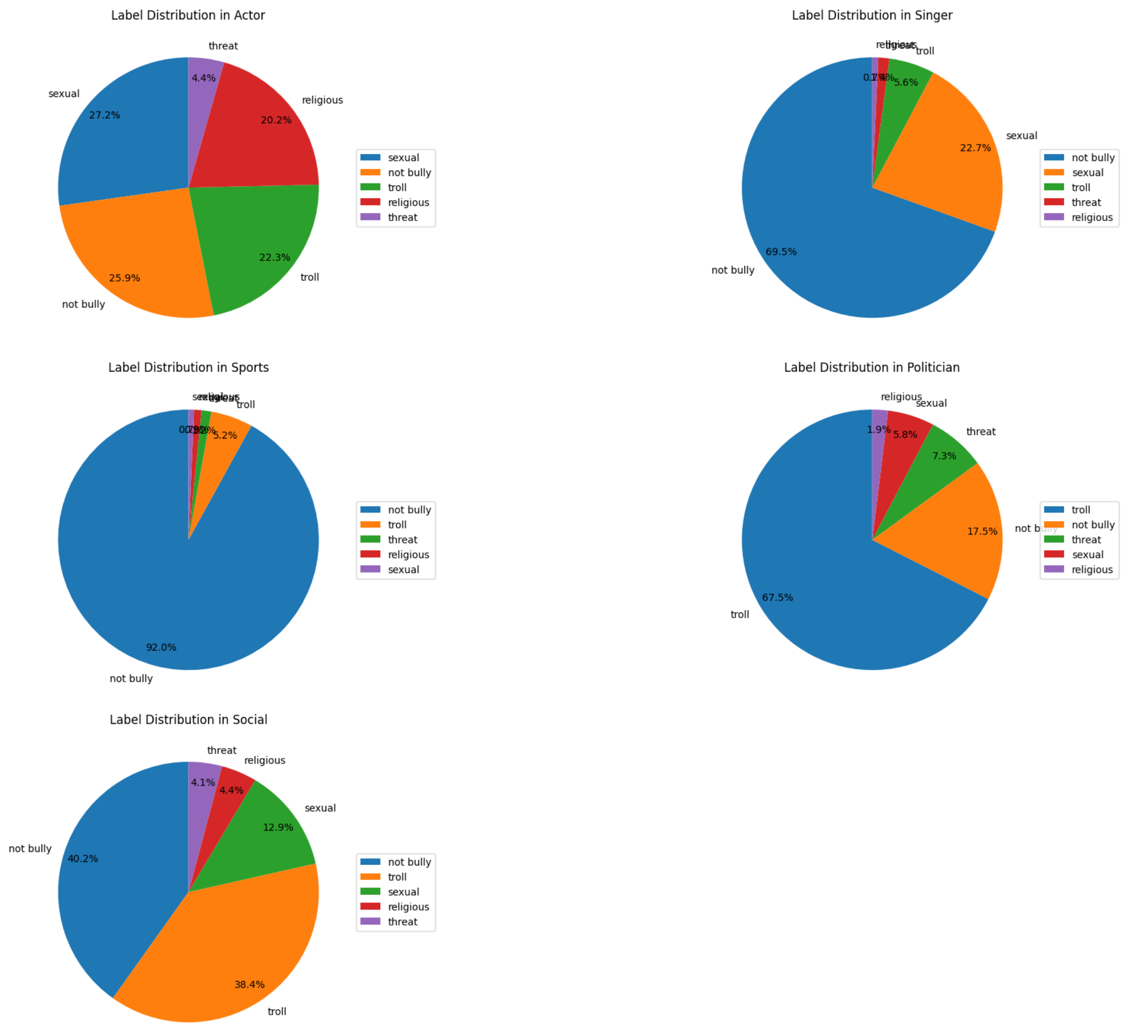
[ ]:
```
plt.figure(figsize=(10,4))
sns.countplot(x='label',data=df)
plt.xlabel('Label')
plt.ylabel('Count')
plt.title('Number of ham and spam messages')
plt.show()
```



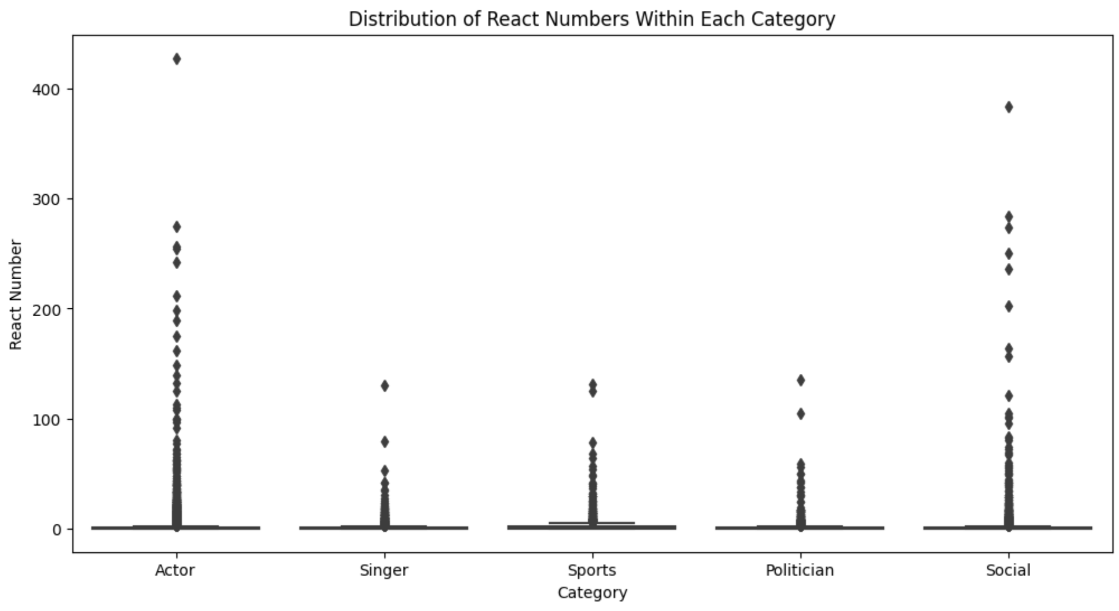
[ ]:
```
plt.figure(figsize=(12, 6))
sns.countplot(x='Category', data=df)
plt.title('Distribution of Comments Across Categories')
plt.xlabel('Category')
plt.ylabel('Count')
plt.show()
```



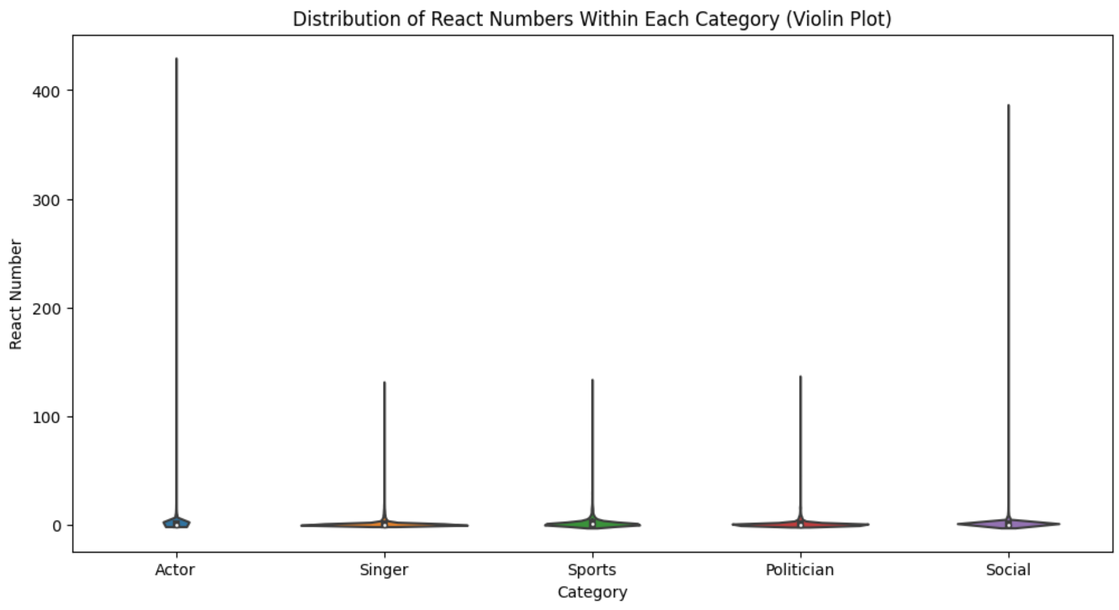
```
[ ]: # Step 2: Pie Charts for percentage distribution of labels within each category
plt.figure(figsize=(20, 15))  # Increase the figure size
categories = df['Category'].unique()
num_categories = len(categories)

for i, category in enumerate(categories):
    plt.subplot((num_categories // 2) + 1, 2, i+1)
    category_df = df[df['Category'] == category]
    labels = category_df['label'].unique()
    pie = plt.pie(category_df['label'].value_counts(), labels=labels,␣
 ↪autopct='%1.1f%%', startangle=90, pctdistance=0.85)
    plt.title(f'Label Distribution in {category}')
    plt.legend(labels, loc="center left", bbox_to_anchor=(1, 0, 0.5, 1))
plt.tight_layout()
plt.show()
```



```
[ ]: plt.figure(figsize=(12, 6))
     sns.boxplot(x='Category', y='comment react number', data=df)
     plt.title('Distribution of React Numbers Within Each Category')
     plt.xlabel('Category')
     plt.ylabel('React Number')
     plt.show()
```



[ ]:
```
plt.figure(figsize=(12, 6))
sns.violinplot(x='Category', y='comment react number', data=df)
plt.title('Distribution of React Numbers Within Each Category (Violin Plot)')
plt.xlabel('Category')
plt.ylabel('React Number')
plt.show()
```



```
[ ]: import nltk
     nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

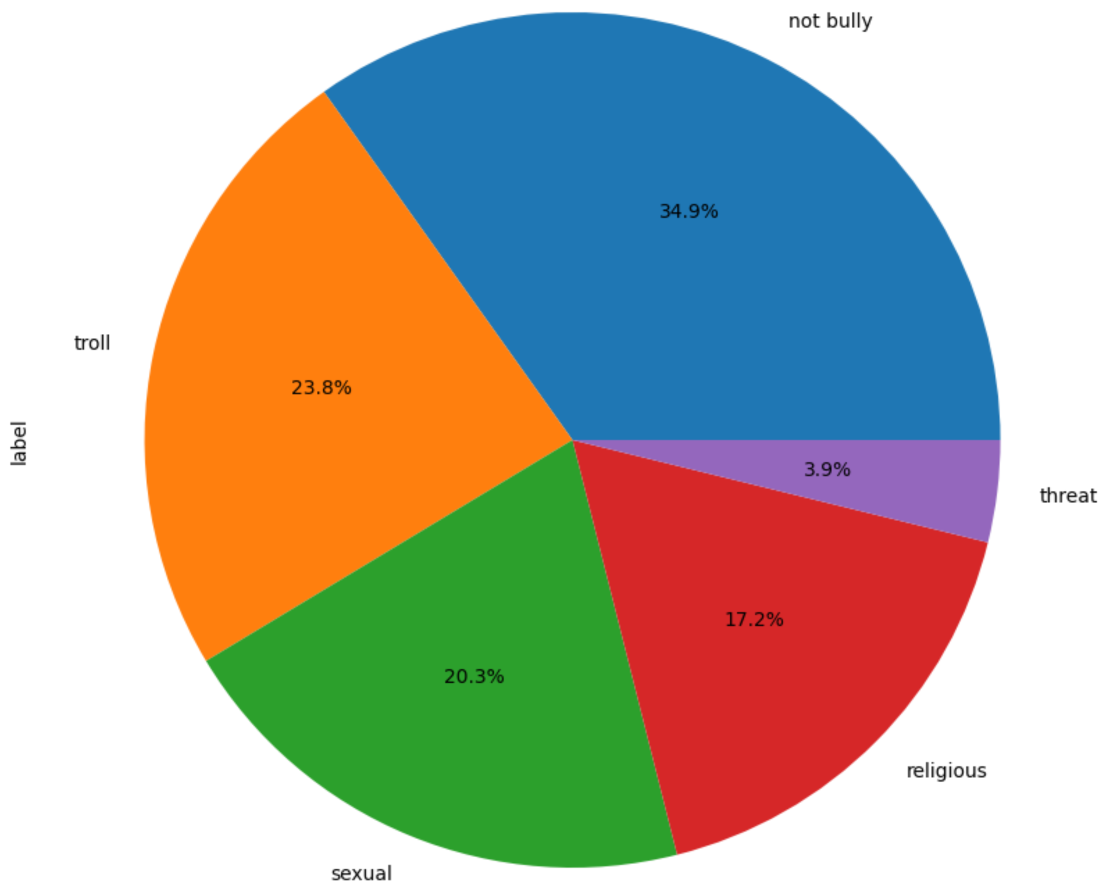
```
[ ]: True
```

```
[ ]: import nltk
     from nltk.corpus import stopwords
     nltk.download('punkt')
     nltk.download('stopwords')

     bengali_stopwords = set(stopwords.words('bengali'))
     print(bengali_stopwords)
     df['comment'] = df['comment'].apply(lambda x: ' '.join([word for word in x.
      ↪split() if word not in bengali_stopwords]))
```

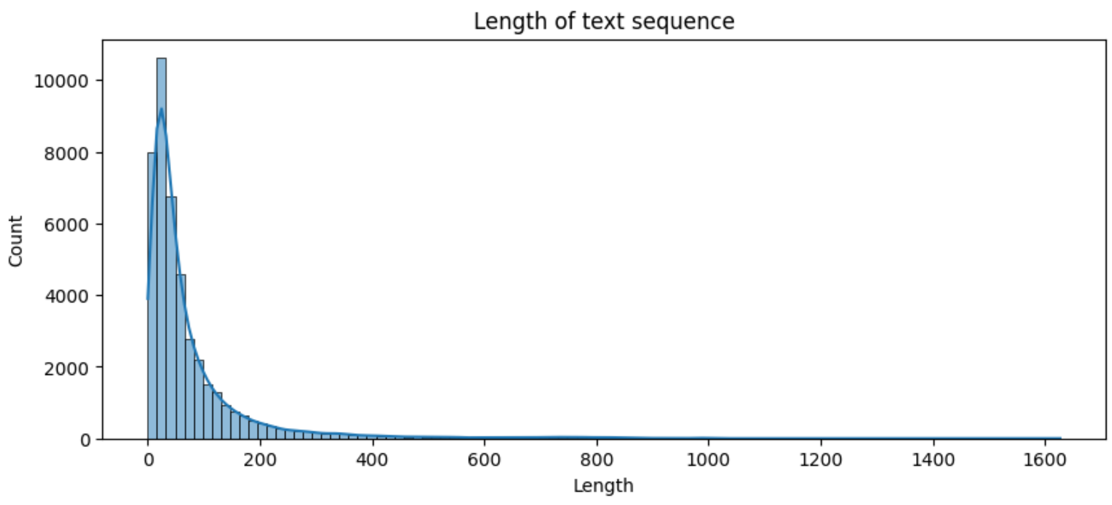
```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
{'  ', '   ', '  ', '   ', ' ', '   ', '   ', '    ', '   ',
'  ', '  ', '  ', '  ', '   ', '   ', '    ', '  ', '    ',
'  ', ' ', '   ', '   ', '  ', '   ', '   ', '   ', ' ', '  ',
'   ', '   ', '   ', '  ', '  ', '   ', '   ', ' ', '   ',
'  ', '  ', '   ', '   ', '   ', '   ', '   ', '  ', '    ',
'   ', ' ', '   ', '    ', '  ', '   ', '   ', '    ', '   ',
'   ', ' ', '   ', '   ', '    ', '   ', '  ', '  ', '  ',
'     ', '    ', '  ', '  ', '  ', ' ', ' ', '   ', '  ', '  ',
'   ', '   ', ' ', '  ', ' ', '  ', '    ', '   ', '   ', '  ',
'   ', '   ', '   ', '   ', ' ', '    ', ' ', '    ', '  ',
'   ', ' ', '    ', '  ', '  ', '   ', '   ', '    ', ' ',
'  ', '   ', '   ', '   ', '   ', '   ', '   ', '  ', '  ', ' ',
'    ', '   ', '    ', '   ', '   ', '     ', '    ', '   ', '   ',
'  ', '  ', '  ', '    ', '    ', '   ', '    ', '   ', '  ',
' ', '   ', '    ', '   ', '    ', ' ', '   ', '   ', '   ',
'  ', '   ', '  ', '  ', '  ', '  ', '  ', '  ', '    ',
'  ', '    ', '   ', ' ', '    ', '   ', '  ', ' ', '   ', '  ',
'    ', '  ', ' ', ' ', ' ', '  ', '   ', '   ', '   ', '  ',
'   ', '    ', '   ', ' ', '   ', '   ', '     ', '   ', '  ',
' ', '   ', '   ', '   ', '   ', ' ', '   ', ' ', '   ', '   ',
'    ', '  ', '   ', '  ', '   ', '    ', '   ', ' ', '   ', '  ',
'    ', '    ', ' ', '  ', '  ', '     ', '    ', '   ',
'   ', ' ', '   ', '  ', '    ', '  ', '  ', '  ', '   ', '   ',
'', '  ', ' ', '  ', '', '   ', '   ', '    ', '   ',
'   ', '  ', '   ', '    ', '   ', '   ', '  ', '  ', '  ',
'  ', ' ', '   ', '  ', '  ', '  ', '   ', ' ', ' ', '  ',
'   ', '   ', '    ', '   ', ' ', '    ', '   ', '   ', '    ',
'   ', '   ', '   ', ' ', '    ', '  ', ' ', ' ', '  ', '    ',
'  ', ' ', '  ', ' ', '   ', '   ', '     ', '   ', ' ', '  ',
'   ', '  ', '   ', '  ', '   ', ' ', '    ', '    ', '   ', '  ',
'    ', ' ', '   ', ' ', '   ', '   ', '    ', '  ', '   ', '   ',
'     ', '   ', '   ', '    ', '   ', '    ', '   ', '   ',
'  ', '   ', '   ', '  ', '   ', '   ', '  ', '   ', '    ',
'   ', '     ', '  ', '   ', '   ', '   ', '  ', '   ', '   ',
'    ', '   ', '  ', ' ', '   ', '  ', '  ', '    ', '   ', '    ',
'   ', '   ', '    ', ' ', '    ', '    ', '   ', '    ', ' ',
'    ', '   ', '  ', '   ', '   ', '    ', ' ', '    ', '   ',
'  ', '    ', '  ', '   ', '   ', '  ', '   ', ' ', '    ',
'    ', '    ', '   ', '  ', '    ', '   ', '   ', ' ', ' ',
'    ', '   ', ' ', ' ', '    ', '    ', '   ', '  ', '   ', '  ',
' ', '   ', '    ', ' ', '   ', '  ', '  ', '    ', '   ', ' ',
'  ', '  ', ' ', '   ', '   ', '  ', ' ', '    ', ' ', '    ',
'   ', '   ', '  '}
```

[ ]:
```python
# pie chart of the labels not bully troll sexual religious threat
plt.figure(figsize=(10, 10))
df['label'].value_counts().plot.pie(autopct='%1.1f%%')
plt.show()
```



```
[ ]: # text sequence length
     plt.figure(figsize=(10,4))
     df['length'] = df['comment'].apply(len)
     sns.histplot(df['length'],kde=True,bins=100)
     plt.xlabel('Length')
     plt.ylabel('Count')
     plt.title('Length of text sequence')
     plt.show()
```



```
[ ]: # explore the datasets
     def explore_data(data):
         for i in range(5):
             print("Sample Comment:-\n",data['comment'][i])
             print("------------------------------------------------------------")
             print("Sample Label:-\n",data['label'][i])
             print("------------------------------------------------------------")

         # analyse the length of text
         text_len = [len(text) for text in data['comment']]
         print("Average length of text:-",np.mean(text_len))
         print("Max length of text:-",np.max(text_len))
         print("Min length of text:-",np.min(text_len))
         print("Standard deviation of length of text:-",np.std(text_len))
         print("Median length of text:-",np.median(text_len))
         print("25 percentile of length of text:-",np.percentile(text_len,25))
         print("75 percentile of length of text:-",np.percentile(text_len,75))
         print("------------------------------------------------------------")
```

```
[ ]: explore_data(df)
```

```
Sample Comment:-
                                                ***
 ****   safa
------------------------------------------------------------
Sample Label:-
 sexual
------------------------------------------------------------
Sample Comment:-
            ?          ?
```

```
------------------------------------------------------------
Sample Label:-
 not bully
------------------------------------------------------------
Sample Comment:-
      ,     ????
------------------------------------------------------------
Sample Label:-
 not bully
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 not bully
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 troll
------------------------------------------------------------
Average length of text:- 75.51056866221192
Max length of text:- 1627
Min length of text:- 0
Standard deviation of length of text:- 109.54130843722012
Median length of text:- 40.0
25 percentile of length of text:- 20.0
75 percentile of length of text:- 83.0
------------------------------------------------------------
```

```
[ ]: # remove punctuation
     remove_punctuations = [
         "/::\)","/::","(-_-)","(*_*)","(>_<)",":)",";)",":
      ↪P","xD","-_-","#","(>_<)","...",".",",",";",":","!","?","'"," ", "   ", " ","?
      ↪",
         "\"","-","_","/
      ↪","\\","|","{","}","[","]","(",")","<",">","@","#","$","%","^","&","*","~","`","+","=","0",
        ␣
      ↪" "," "," "," "," "," "," "," "," ","\n","\t","\r","\f","\v","\u00C0-\u017F","\u2000-\u206F","\
        ␣
      ↪"\uFB00-\uFB4F","\uFE00-\uFE0F","\uFE30-\uFE4F","\u1F600-\u1F64F","\u1F300-\u1F5FF","\u1F68
        ␣
      ↪"\u1F300-\u1F5FF","\u1F900-\u1F9FF","\u1F600-\u1F64F","\u1F680-\u1F6FF","\u1F1E0-\u1F1FF","
     ]
     # reset index of the dataframe
     df.reset_index(inplace=True)
```

[ ]:
```python
for i in range(len(df)):
    text = df.loc[i,'comment']
    for punctuation in remove_punctuations:
        text = text.replace(punctuation,' ')
    df.loc[i,'comment'] = text
```

[ ]:
```python
# remove emoji
def remove_emoji(text):
    emoji_pattern = re.compile(
        "["u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        "]+",
        flags=re.UNICODE,
    )
    return emoji_pattern.sub(r"", text)
```

[ ]:
```python
# remove emoji
for i in range(len(df)):
    text = df.loc[i,'comment']
    text = remove_emoji(text)
    df.loc[i,'comment'] = text
```

[ ]:
```python
# remove english character
def remove_english_character(text):
    english_character = re.compile("[a-zA-Z]+")
    return english_character.sub(r"", text)
```

[ ]:
```python
# remove english character
for i in range(len(df)):
    text = df.loc[i,'comment']
    text = remove_english_character(text)
    df.loc[i,'comment'] = text
```

[ ]:
```python
# remove extra space
def remove_extra_space(text):
    extra_space = re.compile("\s+")
    return extra_space.sub(r" ", text)
```

[ ]:
```python
def remove_single_bengali_character(text):
    # Regular expression pattern to match single Bengali characters
    single_character = re.compile(r'\s[ - ]\s')
    return single_character.sub(" ", text)
```

```
# Identify data to check if the remove_single_bengali_character function works
for i in range(5):
    print("Original data:-\n", df['comment'][i])
    print("Processed data:-\n",␣
 ↪remove_single_bengali_character(df['comment'][i]))
    print("-----------------------------------------------------------")
```

```
Original data:-


Processed data:-


-----------------------------------------------------------
Original data:-

Processed data:-

-----------------------------------------------------------
Original data:-

Processed data:-

-----------------------------------------------------------
Original data:-

Processed data:-

-----------------------------------------------------------
Original data:-

Processed data:-

-----------------------------------------------------------
```

[ ]:
```
df['comment'] = df['comment'].apply(remove_single_bengali_character)
df.head()
```

[ ]:
```
   index                                              comment    Category  \
0      0                                     …       Actor
1      1                                              Singer
2      2                                                        Actor
3      3                                                   Sports
4      4                                                     Politician

   Gender  comment react number      label  length
0  Female                   1.0     sexual     140
```

```
1    Male                   2.0  not bully      38
2  Female                   2.0  not bully      21
3    Male                   0.0  not bully      21
4    Male                   0.0      troll       8
```

[ ]:
```
explore_data(df)
```

```
Sample Comment:-



------------------------------------------------------------
Sample Label:-
 sexual
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 not bully
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 not bully
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 not bully
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 troll
------------------------------------------------------------
Average length of text:- 74.27898995408883
Max length of text:- 1319
Min length of text:- 0
Standard deviation of length of text:- 107.8391821371824
Median length of text:- 39.0
25 percentile of length of text:- 20.0
75 percentile of length of text:- 82.0
------------------------------------------------------------
```

[ ]:
```
# remove extra space
for i in range(len(df)):
    text = df.loc[i,'comment']
    text = remove_extra_space(text)
    df.loc[i,'comment'] = text
```

[ ]:
```
explore_data(df)
```

```
Sample Comment:-



------------------------------------------------------------
Sample Label:-
 sexual
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 not bully
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 not bully
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 not bully
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 troll
------------------------------------------------------------
Average length of text:- 71.9247011227783
Max length of text:- 1195
Min length of text:- 0
Standard deviation of length of text:- 103.40000622663459
Median length of text:- 38.0
25 percentile of length of text:- 20.0
75 percentile of length of text:- 79.0
------------------------------------------------------------
```

```
[ ]: ## Create a list of stopwords
     ##stop_words_list = [
     #    ' ', '  ', ' ', ' ', ' ', ' ', ' ', '  ', '  ', '  ', '  ', '  ', '  ', '  ',␣
       ↪' ', ' ', '  ', '  ', ' ', ' ', '  ', ' ', ' ', '  ', '  ',
     #    ' ', '  ', ' ', ' ', '  ', ' ', ' ', '   ', '  ', ' ', '  ', '   ', '  ',␣
       ↪' ', ' ', '  ', '  ', ' ', '   ', ' ', ' ', ' ', '  ', '  ',
     #    '  ', ' ', '   ', ' ', ' ', '  ', ' ', '   ', '   ', '   ', '  ', '  ',␣
       ↪' ', '  ', ' ', '   ', '  ', ' ', '   ', ' ', '  ', ' ', '  ',
     #    '   ', '  ', '  ', '  ', '  ', '  ', '  ', '   ', '  ', '  ', '   ', '  ', ' ',␣
       ↪' ', ' ', '  ', '  ', '  ', '  ', ' ', '  ', ' ', '  ', '  ',
     #    ' ', '   ', '   ', '  ', ' ', ' ', ' ', ' ', ' ', '  ', '  ', '  ', '   ',␣
       ↪' ', '   ', ' ', ' ', '  ', '  ', ' ', '  ', '   ',
     #    '  ', '   ', '  ', '  ', '   ', '   ', ' ', '   ', '  ', '  ', '  ', '    ',␣
       ↪'  ', ' ', '  ', '  ', ' ', '  ', ' ', '  ', '   ',
     #    '  ', '  ', '  ', '  ', '  ', '  ', '  ', '   ', ' ', '   ', '  ', '  ', '   ',␣
       ↪' ', '   ', '   ', ' ', '  ', '  ', ' ', '  ', '  ',
     #    '  ', ' ', ' ', ' ', ' ', '   ', '  ', ' ', '  ', '  ', '  ', '  ', ' ',␣
       ↪'   ', '  ', '  ', '  ', '  ', ' ', ' ', '  ', '  ', ' ',
     #    ' ', ' ', '  ', '  ', ' ', '  ', '  ', ' ', '   ', ' ', '   ', '  ', '  ',␣
       ↪' ', ' ', ' ', '   ', ' ', '   ', '  ', '   ', '  ', ' ',
     #    '  ', ' ', '   ', ' ', ' ', '  ', ' ', '  ', '  ', '  ', '  ', '  ', ' ',␣
       ↪' ', '  ', '  ', '  ', '  ', '  ', ' ', '   ', ' ', ' ',
     #    '  ', '    ', '  ', '  ', '  ', '  ', '    ', ' ', '  ', '   ', ' ', ' ',␣
       ↪' ', ' ', '    ', '    ', ' ', '  ', ' ', ' ', ' ', '  ',
     #    ' ', ' ', ' ', '  ', '  ', '  ', ' ', '     ', ' ', '  ', '   ', '  ',␣
       ↪'  ', '  ', '   ', '   ', '  ', '  ', ' ', ' ', '  ',
     #    '  ', ' ', ' ', '  ', '  ', '  ', ' ', ' ', ' ', '   ', '   ', '  ', '   ',␣
       ↪'  ', ' ', ' ', '  ', '  ', ' ', ' ', ' ', '  ', ' ',
     #    '  ', '  ', '    ', '   ', ' ', '  ', ' ', ' ', '  ', ' ', '   ', '  ',␣
       ↪' ', '   ', ' ', ' ', ' ', ' ', '   ', ' ', ' ', '   ', ' ',
     #    '  ', '   ', ' ', '    ', '   ', '   ', ' ', '  ', '  ', ' ', ' ', '  ', '   ',␣
       ↪' ', '  ', '  ', '    ', '    ', '   ', '   ', ' ', ' ', ' ', ' ', ' ',
     #    '  ', ' ', '   ', '  ', '  ', '  ', '  ', '  ', '   ', ' ', '  ', ' ', '  ',␣
       ↪' ', ' ', ' ', '  ', ' ', ' ', '  ', '  ', '  ', '   ', '   ', ' ', ' ', ' ',
     #    ' ', ' ', '  ', '  ', ' ', '    ', ' ', ' ', '  ', '  ', '   ', ' ', ' ',␣
       ↪' ', '  ', ' ', '  ', '   ', '   ', '  ', '  ', ' ', '  ', '  ', '  ', '   ',␣
       ↪' ',
     #    '  ', '  ', '  ', '  ', '  ', '  ', ' ', '  ', '  ', ' ', '  ', '  ', '  ',␣
       ↪'  ', ' ', '  ', '  ', '  ', '  ', '  ', ' ', '  ', ' ', '  ', '  ', '   ',␣
       ↪' ',
     #    '  ', ' ', ' ', '  ', '  ', ' ', ' ', '  ', ' ', ' ', '  ', '  ', '  ',␣
       ↪' ', '   ', ' ', '  ', ' ', ' ', '  ', ' ', '  ', ' ', '  ', '  ', '  ',␣
       ↪' ',
     #    '  ', '  ', '  ', '  ', ' ', ' ', ' ', ' ', '  ', '  ', ' ', ' ', '   ',␣
       ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', '  ', '  ', ' ', '  ', '  ', ' ', '  ',␣
       ↪' ',
#     ' ', '  ', ' ', '  ', '  ', '  ', '  ', '  ', '  ', ' ', '  ', '  ', ' ', '  ',␣
  ↪'  ', '  ', '  ', ' ', ' ', '  ', '  ', '  ', '  ', ' ', '  ', '  ', '  ', '  ',
#     ' ', ' ', '  ', '  ', '  ', '  ', '  ', '  ', ' ', '  ', '   ', '  ', '  ',␣
  ↪'  ', '  ', '  ', '  ', '  ', '  ', '  ', '   ', '   ', '  ', '   ', '   ',
#     '  ', '  ', '  ', '   ', '  ', '   ', '   ', '   ', '  ', '  ', ' ', ' ',␣
  ↪'  ', ' ', ' ', ' ', '  ', ' ', ' ', '  ', '  ', ' ', '  ', '  ', '  ', ' ',␣
  ↪'  ',
#     ' ', '  ', ' ', '  ', ' ', '  ', '  ', '  ', '  ', ' ', '  ', '  ', ' ',␣
  ↪' ', '  ', ' ', ' ', '  ', '   ', '  ', '  ', '  ', '  ', '  ', '  ', '   ',␣
  ↪'  ',
#     '  ', '  ', '  ', '   ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ',␣
  ↪'   ', '   ', '   ', '  ', '   ', '  ', ' ', '  ', ' ', '  ', '  ', '  ', '   ',
#     '  ', '  ', '  ', '   ', '   ', '   ', '   ', '   ', '   ', '  ', '  ', '  ',␣
  ↪'  ', '  ', '  ', '   ', '   ', '  ', '  ', '   ', '  ', '  ', '  ', '  ',
#     '  ', '  ', '  ', '   ', '  ', '  ', '  ', '  ', '  ', '   ', '  ', '  ',␣
  ↪'  ', '  ', '  ', '  ', ' ', ' ', ' ', '  ', '  ', '  ', ' ', '  ', '  ',
#     ' ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', ' ', '  ', '  ', ' ', '  ',␣
  ↪'  ', '  ', '  ', '  ', ' ', ' ', '  ', ' ', ' ', '  ', ' ', '  ', '  ', '   ',
#     '  ', '  ', '  ', ' ', '  ', '  ', ' ', '  ', '   ', '   ', '  ', '  ',␣
  ↪'  ', '   ', '  ', '  ', ' ', ' ', '  ', ' ', ' ', '  ', '  ', '   ', '  ',␣
  ↪'  ',
#     '  ', ' ', '  ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '  ', '  ',␣
  ↪'  ', ' ', '  ', '  ', '  ', '  ', '  ', '  ', '   ', ' ', '  ', ' ', '  ',␣
  ↪'  ', ' ',
#     '   ', '  ', ' ', ' ', '  ', ' ', ' ', '  ', ' ', ' ', ' ', '  ', ' ', ' ',␣
  ↪' ', '   ', '  ', '  ', ' ', '  ', '  ', '    ', ' ', ' ', '  ', '  ', ' ',␣
  ↪' ',
```

```
#   '  ', ' ', '  ', ' ', ' ', ' ', '  ', '    ', '  ', ' ', '  ', '  ', ' ',␣
  ↪'  ', '  ', ' ', '  ', '  ', '  ', '  ', ' ', '  ', ' ', ' ', ' ', '  ', ' ',␣
  ↪'  ', ' ',
#   ' ', ' ', '  ', '  ', '  ', ' ', ' ', '  ', '  ', ' ', ' ', '  ', ' ',␣
  ↪'  ', ' ', ' ', ' ', ' ', '  ', ' ', ' ', '  ', ' ', '  ', '  ', ' ', ' ',␣
  ↪' ', '  ', ' ',
#   ' ', '    ', ' ', '  ', '  ', '  ', '  ', ' ', ' ', '  ', ' ', '  ',␣
  ↪'  ', '  ', '  ', ' ', ' ', ' ', ' ', '    ', '  ', '   ', '  ', ' ',
#   ' ', ' ', ' ', ' ', '   ', '  ', '  ', ' ', ' ', '   ', '  ', '  ',␣
  ↪'  ', ' ', '  ', '  ', ' ', ' ', ' ', '  ', ' ', '  ', '  ', ' ', '  ',␣
  ↪'  ', '  ',
#   '  ', '  ', '  ', ' ', '  ', ' ', '  ', '  ', ' ', '   ', '  ', ' ', '  ',␣
  ↪' ', ' ', '  ', '  ', ' ', ' ', ' ', '    ', ' ', '  ', '  ', '    ', '  ',␣
  ↪' ', ' ',
#   ' ', '  ', ' ', ' ', ' ', '  ', '  ', '  ', ' ', '  ', '  ', ' ', '  ',␣
  ↪'  ', '  ', '  ', ' ', '  ', ' ', ' ', '    ', '  ', ' ', '  ', '  ', '  ',␣
  ↪'  ', '  ',
#   ' ', ' ', '  ', '  ', '  ', '  ', '  ', '  ', ' ', ' ', '  ', ' ', '  ',␣
  ↪'  ', '  ', '  ', '  ', ' ', ' ', '  ', ' ', '  ', '  ', '  ', '  ', '  ',␣
  ↪' ', '  ', ' ',
#   ' ', ' ', '  ', '  ', ' ', '  ', '  ', '  ', ' ', '   ', '  ', '  ', '  ',␣
  ↪'  ', ' ', '  ', '  ', ' ', '  ', ' ', '  ', ' ', '  ', ' ', '  ', '  ', '  ',␣
  ↪'  ', '  ',
#   ' ', '  ', '  ', '  ', ' ', '  ', '  ', '  ', ' ', '  ', '  ', '  ', '   ',␣
  ↪'  ', '  ', ' ', ' ', '  ', '  ', '  ', '  ', '   ', '  ', '  ', '  ', '  ', '  ',␣
  ↪'  ',
#   '  ', '   ', '  ', '  ', '   ', '  ', '  ', ' ', ' ', '  ', ' ', '  ', '  ',␣
  ↪'  ', '  ', '   ', ' ', ' ', ' ', '  ', '  ', ' ', '  ', '  ', '  ', '  ', ' ',
#   ' ', ' ', '  ', '  ', '  ', ' ', ' ', '   ', ' ', ' ', '  ', '  ', '    ',␣
  ↪'  ', '  ', ' ', '  ', '  ', '  ', ' ', '   ', ' ', ' ', '  ', '  ', '  ', ' ',
#   '  ', '    ', '  ', '  ', '  ', ' ', ' ', ' ', '  ', ' ', ' ', '  ', ' ',␣
  ↪' ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', ' ', '  ', '  ', '  ', '  ',␣
  ↪' ',
#   ' ', ' ', '  ', ' ', ' ', ' ', '  ', ' ', ' ', ' ', ' ', ' ', ' ', '  ',␣
  ↪'  ', '  ', ' ', '  ', ' ', '  ', ' ', '  ', ' ', '  ', ' ', '  ', ' ', ' ',␣
  ↪' ', ' ',
#   ' ', ' ', ' ', '  ', '  ', ' ', '  ', '  ', '  ', ' ', ' ', ' ', ' ', '  ',␣
  ↪'  ', '   ', ' ', '  ', '  ', ' ', '   ', '  ', '   ', '  ', ' ', '  ', '  ',␣
  ↪' ', ' ',
#   ' ', ' ', ' ', '  ', '  ', ' ', '  ', ' ', '  ', ' ', '  ', '  ', '  ', ' ',␣
  ↪'  ', ' ', ' ', '  ', ' ', ' ', '  ', '  ', '  ', '  ', ' ', '  ', '  ', ' ',␣
  ↪'  ', ' ', '  ',
#   '  ', '  ', '  ', ' ', '  ', '  ', '  ', '  ', ' ', '  ', '  ', '  ', '   ',␣
  ↪'  ', '  ', ' ', '  ', '  ', '  ', '  ', ' ', '  ', '  ', '  ', '  ', ' ', '  ',
#    '  ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '  ', ' ', ' ', '  ', '  ', '  ',␣
  ↪' ', ' ', ' ', '  ', '  ', ' ', ' ', ' ', '  ', '  ', '  ', ' ', '  ', ' ',␣
  ↪'  ',
#    '  ', '  ', ' ', '  ', '  ', '  ', '  ', '  ', '  ', ' ', '  ', '  ', '  ',␣
  ↪'  ', '  ', ' ', ' ', '  ', '   ', '   ', '   ', ' ', ' ', '  ', '  ', '  ',␣
  ↪'   ',
#    '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', ' ', '  ', ' ',␣
  ↪'  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', ' ', '  ', '  ', '   ',␣
  ↪'  ',
#    ' ', '  ', '  ', '  ', ' ', ' ', '  ', '  ', '  ', '  ', '  ', '  ', '  ',␣
  ↪'   ', '   ', '  ', '  ', '  ', ' ', '  ', '  ', '  ', '  ', ' ', '  ', '   ',␣
  ↪'  ',
#    '  ', ' ', set, '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ',␣
  ↪'  ', '  ', '  ', '  ', ' ', '  ', '  ', '  ', '   ', ' ', '  ', '  ', '  ',␣
  ↪'  ', '  ',
#    '  ', ' ', '  ', '  ', '  ', '  ', '  ', ' ', ' ', ' ', '  ', ' ', '  ',␣
  ↪'  ', ' ', ' ', ' ', '  ', '  ', '  ', '  ', ' ', ' ', '  ', '  ', '  ', '  ',␣
  ↪'  ', ' ',
#    ' ', ' ', '  ', '  ', '   ', ' ', ' ', '  ', ' ', '  ', ' ', '  ', '  ',␣
  ↪'  ', '  ', '  ', '  ', '  ', '  ', ' ', '   ', '  ', '   ', '  ', '  ', '  ',␣
  ↪' ', ' ',
#    '  ', ' ', '  ', ' ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', ' ', '  ',␣
  ↪'  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ',␣
  ↪'  ', '  ',
#    ' ', ' ', ' ', ' ', ' ', ' ', ' ', '  ', '  ', '  ', ' ', ' ', '  ', ' ',␣
  ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', '  ', ' ', '  ', ' ', '  ', '  ', '  ',␣
  ↪'  ',
#    '  ', '  ', ' ', '  ', '  ', '  ', '  ', ' ', '  ', ' ', '  ', '  ', '  ',␣
  ↪'  ', '  ', '  ', ' ', ' ', ' ', ' ', '  ', ' ', ' ', '  ', '  ', '  ',␣
  ↪'  ',
#    '  ', ' ', ' ', '  ', ' ', '  ', '  ', ' ', ' ', '  ', '  ', '  ', ' ',␣
  ↪' ', '  ', ' ', ' ', '  ', ' ', '  ', ' ', '  ', '  ', ' ', '  ']
```

[ ]:
```
## remove stop words
#def remove_stop_words(text):
#    words = text.split()
#    filtered_words = [word for word in words if word not in stop_words_list]
#    return ' '.join(filtered_words)
#
#df['comment'] = df['comment'].apply(remove_stop_words)
```

[ ]:
```
# number unique words
unique_words = set()
for comment in df['comment']:
    for word in comment.split():
        unique_words.add(word)
```

```
print(len(unique_words))
```

```
56199
```

[ ]:
```
# total number of words
total_words = [word for comment in df['comment'] for word in comment.split()]
print(len(total_words))
```

```
535585
```

[ ]:
```
df = df[['comment', 'label']]
```

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

[ ]:
```
                                                comment      label
0                                     …     sexual
1                                              not bully
2                                                        not bully
3                                                   not bully
4                                                                 troll
```

[ ]:
```
explore_data(df)
```

```
Sample Comment:-


------------------------------------------------------------
Sample Label:-
 sexual
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 not bully
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 not bully
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 not bully
```

```
------------------------------------------------------------
Sample Comment:-

------------------------------------------------------------
Sample Label:-
 troll
------------------------------------------------------------
Average length of text:- 71.9247011227783
Max length of text:- 1195
Min length of text:- 0
Standard deviation of length of text:- 103.40000622663459
Median length of text:- 38.0
25 percentile of length of text:- 20.0
75 percentile of length of text:- 79.0
------------------------------------------------------------
```

[ ]:
```python
le = LabelEncoder()
df['label'] = le.fit_transform(df['label'])

labels = to_categorical(df['label'], num_classes=5)

df.head()
```

[ ]:
```
                                              comment  label
0                             …       2
1                                          0
2                                                   0
3                                               0
4                                                     4
```

[ ]:
```python
train_texts, test_texts, train_labels, test_labels =␣
 ↪train_test_split(df['comment'].tolist(), df['label'].tolist(), test_size=0.2)
```

[ ]:
```python
from transformers import TFBertModel
import tensorflow as tf
```

[ ]:
```python
tokenizer = BertTokenizer.from_pretrained("sagorsarker/bangla-bert-base")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:72:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
vocab.txt:   0%|          | 0.00/2.24M [00:00<?, ?B/s]

config.json:   0%|          | 0.00/491 [00:00<?, ?B/s]
```

```
[ ]: max_length = 128
     train_encodings = tokenizer(train_texts, truncation=True, padding=True,␣
      ↪max_length=max_length, return_tensors="tf")
     test_encodings = tokenizer(test_texts, truncation=True, padding=True,␣
      ↪max_length=max_length, return_tensors="tf")
```

```
[ ]: num_labels = len(df['label'].unique())
```

```
[ ]: # Assuming train_encodings and test_encodings contain input_ids
     train_input_ids = train_encodings['input_ids']
     test_input_ids = test_encodings['input_ids']

     # Trim or pad sequences to the desired length (128)
     max_length = 128
     train_input_ids = tf.keras.preprocessing.sequence.
      ↪pad_sequences(train_input_ids, maxlen=max_length, padding='post')
     test_input_ids = tf.keras.preprocessing.sequence.pad_sequences(test_input_ids,␣
      ↪maxlen=max_length, padding='post')
```

```
[ ]: # Create datasets
     train_dataset = tf.data.Dataset.from_tensor_slices((
         {
             'sequences': train_input_ids,
             'attention_mask': train_encodings['attention_mask']
         },
         tf.keras.utils.to_categorical(train_labels, num_labels)
     ))

     test_dataset = tf.data.Dataset.from_tensor_slices((
         {
             'sequences': test_input_ids,
             'attention_mask': test_encodings['attention_mask']
         },
         tf.keras.utils.to_categorical(test_labels, num_labels)
     ))
```

```
[ ]: # One-hot encode labels
     train_labels_onehot = to_categorical(train_labels, num_labels)
     test_labels_onehot = to_categorical(test_labels, num_labels)
```

1 GRU Model

[ ]:
```
from keras.layers import Input, Embedding, Bidirectional, GRU, Dense, Dropout,␣
 ↪Attention, Reshape
from keras.models import Model
from keras.layers import GlobalAveragePooling1D
from keras.layers import Input, Embedding, Bidirectional, GRU, Dense, Dropout,␣
 ↪Attention, Reshape, Flatten
from keras.models import Model
```

[ ]:
```
vocab_size = tokenizer.vocab_size
embedding_dim = 300
```

[ ]:
```
def GRUmodel(vocab_size, embedding_dim=128, sequence_length=128):
    sequences = Input(shape=(sequence_length,), dtype='int32', name='sequences')

    embedded_sequences = Embedding(vocab_size, embedding_dim,␣
 ↪input_length=sequence_length)(sequences)

    # Enhanced GRU layers
    x = Bidirectional(GRU(256, return_sequences=True))(embedded_sequences)
    x = Bidirectional(GRU(128))(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.2)(x)
    # Attention mechanism
    attention = Attention()([x, x])

    # Flatten and additional dense layers
    x = Flatten()(attention)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.1)(x)
    x = Dense(32, activation='relu')(x)

    num_classes = 5
    output = Dense(num_classes, activation='softmax')(x)

    return Model(inputs=sequences, outputs=output)
```

[ ]:
```
vocab_size = tokenizer.vocab_size
model = GRUmodel(vocab_size, sequence_length=128)
model.summary()
```

```
Model: "model_3"
________________________________________________________________________________
__________________
```

```
 Layer (type)                Output Shape                 Param #   Connected to
================================================================================
==================
 sequences (InputLayer)      [(None, 128)]                0         []

 embedding_3 (Embedding)     (None, 128, 128)             1305280
['sequences[0][0]']
                                                          0

 bidirectional_5 (Bidirecti  (None, 128, 512)             592896
['embedding_3[0][0]']
 onal)

 bidirectional_6 (Bidirecti  (None, 256)                  493056
['bidirectional_5[0][0]']
 onal)

 dense_10 (Dense)            (None, 64)                   16448
['bidirectional_6[0][0]']

 dropout_5 (Dropout)         (None, 64)                   0
['dense_10[0][0]']

 attention_3 (Attention)     (None, 64)                   0
['dropout_5[0][0]',
'dropout_5[0][0]']

 flatten_3 (Flatten)         (None, 64)                   0
['attention_3[0][0]']

 dense_11 (Dense)            (None, 128)                  8320
['flatten_3[0][0]']

 dropout_6 (Dropout)         (None, 128)                  0
['dense_11[0][0]']

 dense_12 (Dense)            (None, 64)                   8256
['dropout_6[0][0]']

 dropout_7 (Dropout)         (None, 64)                   0
['dense_12[0][0]']

 dense_13 (Dense)            (None, 32)                   2080
['dropout_7[0][0]']

 dense_14 (Dense)            (None, 5)                    165
['dense_13[0][0]']
```

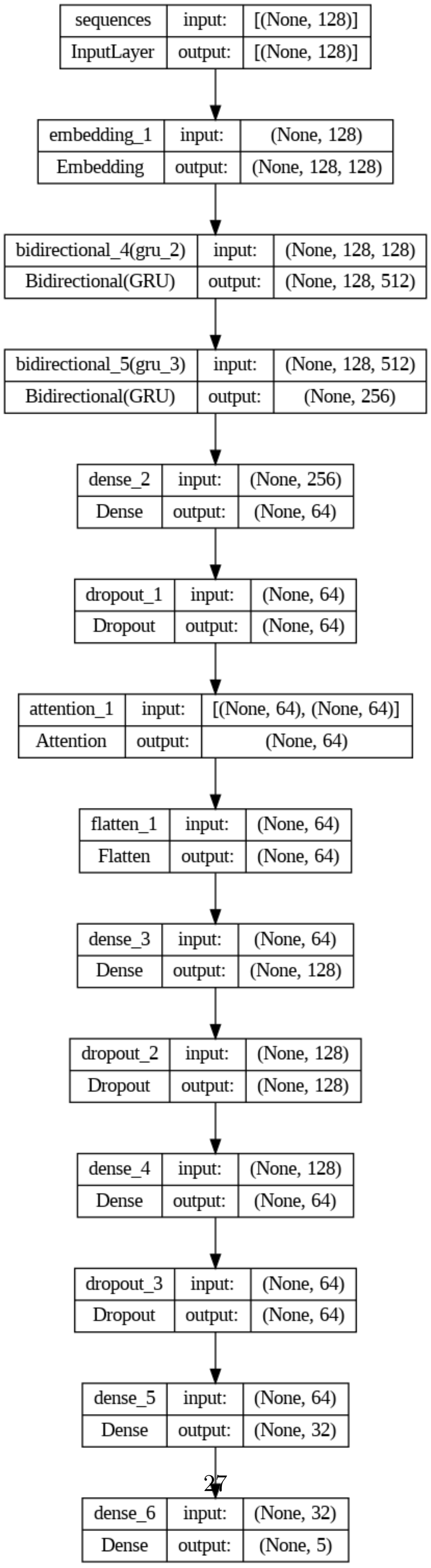
```
================================================================================
==================
Total params: 14174021 (54.07 MB)
Trainable params: 14174021 (54.07 MB)
Non-trainable params: 0 (0.00 Byte)
--------------------------------------------------------------------------------
------------------
```

[ ]:
```
!pip install pydot graphviz
```

```
Requirement already satisfied: pydot in /usr/local/lib/python3.10/dist-packages
(1.4.2)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-
packages (0.20.1)
Requirement already satisfied: pyparsing>=2.1.4 in
/usr/local/lib/python3.10/dist-packages (from pydot) (3.1.1)
```

[ ]:
```
from tensorflow.keras.utils import plot_model

# Save the model summary as an image file
plot_model(model, to_file='hybrid_model_summary.png', show_shapes=True,␣
 ↪show_layer_names=True)
```

[ ]:


sequences | input: | [(None, 128)]
InputLayer | output: | [(None, 128)]
embedding_1 | input: | (None, 128)
Embedding | output: | (None, 128, 128)
bidirectional_4(gru_2) | input: | (None, 128, 128)
Bidirectional(GRU) | output: | (None, 128, 512)
bidirectional_5(gru_3) | input: | (None, 128, 512)
Bidirectional(GRU) | output: | (None, 256)
dense_2 | input: | (None, 256)
Dense | output: | (None, 64)
dropout_1 | input: | (None, 64)
Dropout | output: | (None, 64)
attention_1 | input: | [(None, 64), (None, 64)]
Attention | output: | (None, 64)
flatten_1 | input: | (None, 64)
Flatten | output: | (None, 64)
dense_3 | input: | (None, 64)
Dense | output: | (None, 128)
dropout_2 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_4 | input: | (None, 128)
Dense | output: | (None, 64)
dropout_3 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_5 | input: | (None, 64)
Dense | output: | (None, 32)
dense_6 | input: | (None, 32)
Dense | output: | (None, 5)

```
[ ]: from keras.optimizers import Adam

     optimizer = Adam(learning_rate=0.001)  # Adjust the learning rate if needed
     model.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
      ↪metrics=['accuracy'])
```

```
[ ]: train_labels_onehot = tf.keras.utils.to_categorical(train_labels,␣
      ↪num_classes=len(set(train_labels)))
     test_labels_onehot = tf.keras.utils.to_categorical(test_labels,␣
      ↪num_classes=len(set(test_labels)))
```

```
[ ]: from tensorflow.keras.callbacks import EarlyStopping
```

```
[ ]: # Early Stopping
     early_stopping = EarlyStopping(monitor='val_loss', patience=3,␣
      ↪restore_best_weights=True)
```

```
[ ]: # Train the model
     history = model.fit(
         train_encodings['input_ids'],
         train_labels_onehot,
         validation_data=(test_encodings['input_ids'],
                          test_labels_onehot),
         epochs=10,
         batch_size=32,
         callbacks=[early_stopping]
     )
```

```
Epoch 1/10
1100/1100 [==============================] - 192s 175ms/step - loss: 0.6115 -
accuracy: 0.8265 - val_loss: 0.7033 - val_accuracy: 0.7886
Epoch 2/10
1100/1100 [==============================] - 192s 174ms/step - loss: 0.5562 -
accuracy: 0.8418 - val_loss: 0.6532 - val_accuracy: 0.7965
Epoch 3/10
1100/1100 [==============================] - 193s 175ms/step - loss: 0.5390 -
accuracy: 0.8481 - val_loss: 0.6810 - val_accuracy: 0.7973
Epoch 4/10
1100/1100 [==============================] - 184s 168ms/step - loss: 0.5042 -
accuracy: 0.8592 - val_loss: 0.6760 - val_accuracy: 0.7969
Epoch 5/10
1100/1100 [==============================] - 183s 167ms/step - loss: 0.4872 -
accuracy: 0.8646 - val_loss: 0.6531 - val_accuracy: 0.7994
Epoch 6/10
1100/1100 [==============================] - 186s 169ms/step - loss: 0.4632 -
```

```
accuracy: 0.8703 - val_loss: 0.6715 - val_accuracy: 0.7973
Epoch 7/10
1100/1100 [==============================] - 192s 175ms/step - loss: 0.4593 -
accuracy: 0.8715 - val_loss: 0.6453 - val_accuracy: 0.7995
Epoch 8/10
1100/1100 [==============================] - 192s 175ms/step - loss: 0.4601 -
accuracy: 0.8752 - val_loss: 0.6489 - val_accuracy: 0.8070
Epoch 9/10
1100/1100 [==============================] - 184s 167ms/step - loss: 0.4187 -
accuracy: 0.8863 - val_loss: 0.6570 - val_accuracy: 0.8073
Epoch 10/10
1100/1100 [==============================] - 184s 168ms/step - loss: 0.3990 -
accuracy: 0.8916 - val_loss: 0.6624 - val_accuracy: 0.8031
```

[ ]:
```python
# Evaluate the model on test data
loss, accuracy = model.evaluate(test_dataset.batch(32))
print(f'Test Accuracy: {accuracy * 100:.2f}%')
print(f'Test Loss: {loss:.4f}')

# Get predictions for test data
predictions = model.predict(test_dataset.batch(32))
predicted_classes = predictions.argmax(axis=1)
true_classes = test_labels
```

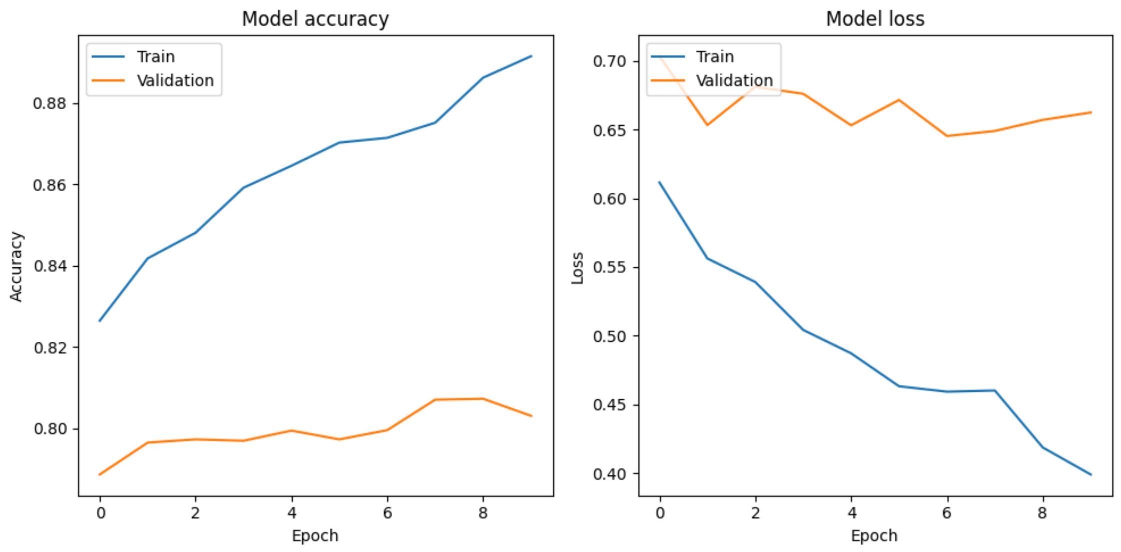
```
  4/275 […] - ETA: 13s - loss: 0.8858 - accuracy:
0.7188
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/functional.py:642:
UserWarning: Input dict contained keys ['attention_mask'] which did not match
any model input. They will be ignored by the model.
  inputs = self._flatten_to_reference_inputs(inputs)
```

```
275/275 [==============================] - 14s 51ms/step - loss: 0.6453 -
accuracy: 0.7995
Test Accuracy: 79.95%
Test Loss: 0.6453
275/275 [==============================] - 10s 36ms/step
```

[ ]:
```python
# Plot training & validation accuracy values
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
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 left')
```

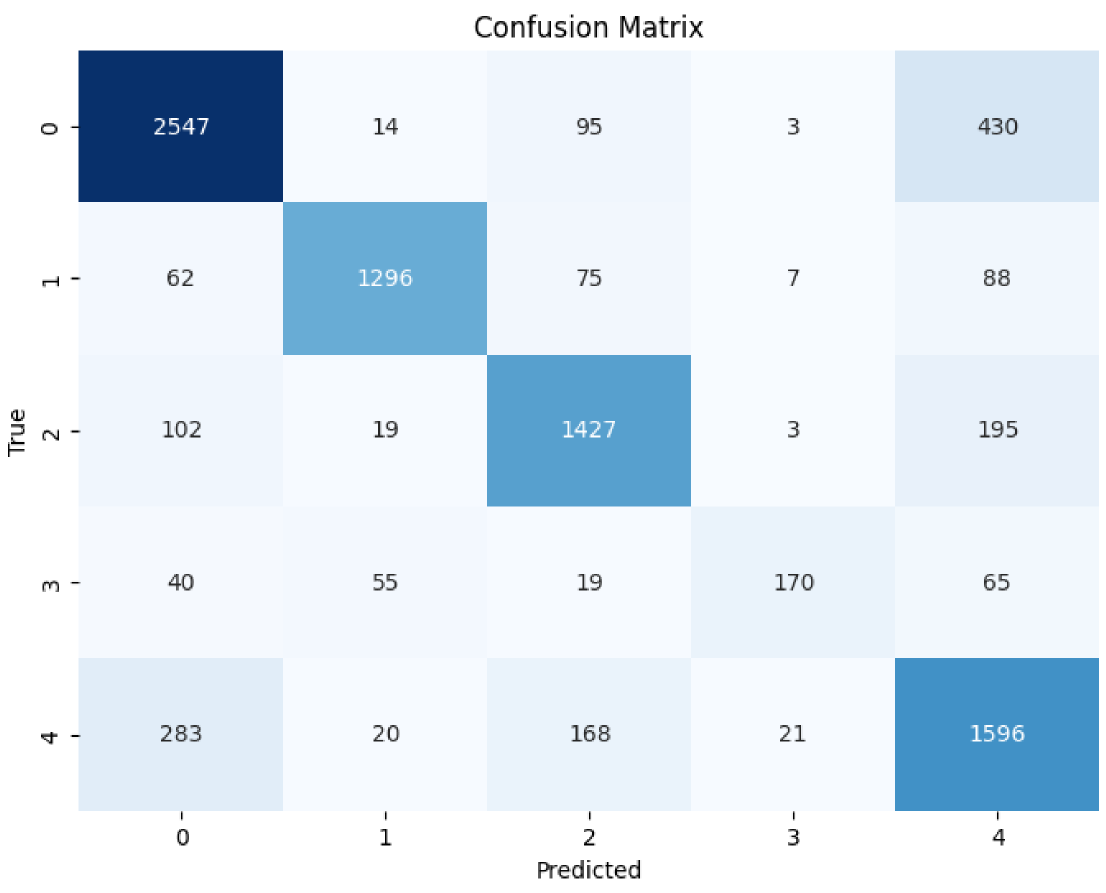
```
# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.tight_layout()
plt.show()
```



[ ]:
```
from sklearn.metrics import classification_report, confusion_matrix,␣
↪precision_recall_fscore_support
```

[ ]:
```
# Classification Report
class_report = classification_report(true_classes, predicted_classes)
print("Classification Report:\n", class_report)

# Confusion Matrix
conf_matrix = confusion_matrix(true_classes, predicted_classes)
print("Confusion Matrix:\n", conf_matrix)

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Precision, Recall, F1-score
precision, recall, f1_score, _ = precision_recall_fscore_support(true_classes,␣
 ↪predicted_classes, average='weighted')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1_score:.4f}')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.82      0.83      3089
           1       0.92      0.85      0.88      1528
           2       0.80      0.82      0.81      1746
           3       0.83      0.49      0.61       349
           4       0.67      0.76      0.72      2088

    accuracy                           0.80      8800
   macro avg       0.81      0.75      0.77      8800
weighted avg       0.81      0.80      0.80      8800

Confusion Matrix:
 [[2547   14   95    3  430]
 [  62 1296   75    7   88]
 [ 102   19 1427    3  195]
 [  40   55   19  170   65]
 [ 283   20  168   21 1596]]
```



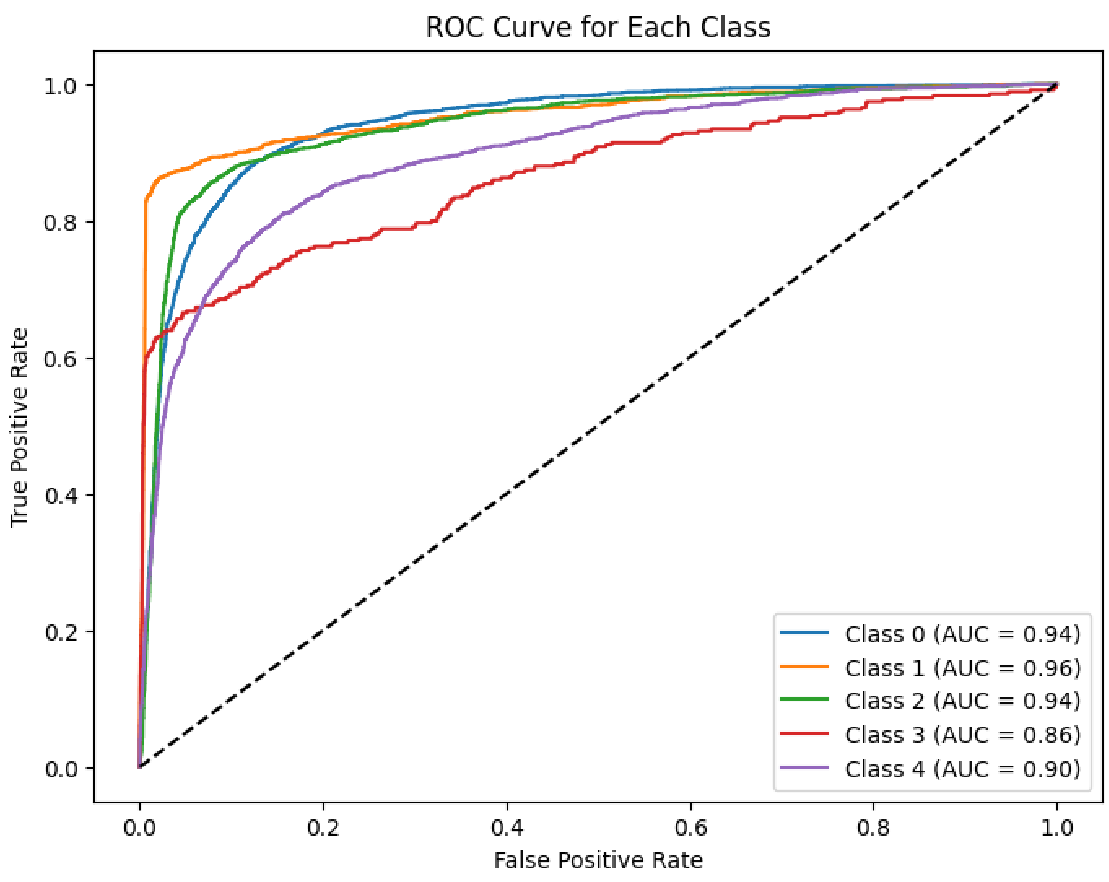
```
Precision: 0.8062
Recall: 0.7995
F1-score: 0.8001
```

[ ]:
```
# Extract TP, TN, FP, FN from confusion matrix
TP = conf_matrix[1, 1]  # True Positives
TN = conf_matrix[0, 0]  # True Negatives
FP = conf_matrix[0, 1]  # False Positives
FN = conf_matrix[1, 0]  # False Negatives

print("True Positives:", TP)
print("True Negatives:", TN)
print("False Positives:", FP)
print("False Negatives:", FN)
```

```
True Positives: 1296
True Negatives: 2547
False Positives: 14
False Negatives: 62
```

```
[ ]: from sklearn.preprocessing import LabelBinarizer
     from sklearn.metrics import roc_curve, roc_auc_score

     # Get predictions for test data
     predictions = model.predict(test_dataset.batch(32))
     predicted_classes = predictions.argmax(axis=1)
     true_classes = test_labels

     # Transform true labels to binary format
     label_binarizer = LabelBinarizer()
     true_labels_bin = label_binarizer.fit_transform(true_classes)

     # Calculate ROC curve and AUC for each class
     fpr = dict()
     tpr = dict()
     roc_auc = dict()
     num_labels = len(label_binarizer.classes_)

     for i in range(num_labels):
         fpr[i], tpr[i], _ = roc_curve(true_labels_bin[:, i], predictions[:, i])
         roc_auc[i] = roc_auc_score(true_labels_bin[:, i], predictions[:, i])

     # Plot ROC curve for each class
     plt.figure(figsize=(8, 6))

     for i in range(num_labels):
         plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

     plt.plot([0, 1], [0, 1], 'k--')  # Diagonal reference line
     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.title('ROC Curve for Each Class')
     plt.legend(loc="lower right")
     plt.show()
```

```
  3/275 […] - ETA: 14s

/usr/local/lib/python3.10/dist-packages/keras/src/engine/functional.py:642:
UserWarning: Input dict contained keys ['attention_mask'] which did not match
any model input. They will be ignored by the model.
  inputs = self._flatten_to_reference_inputs(inputs)

275/275 [==============================] - 11s 38ms/step
```



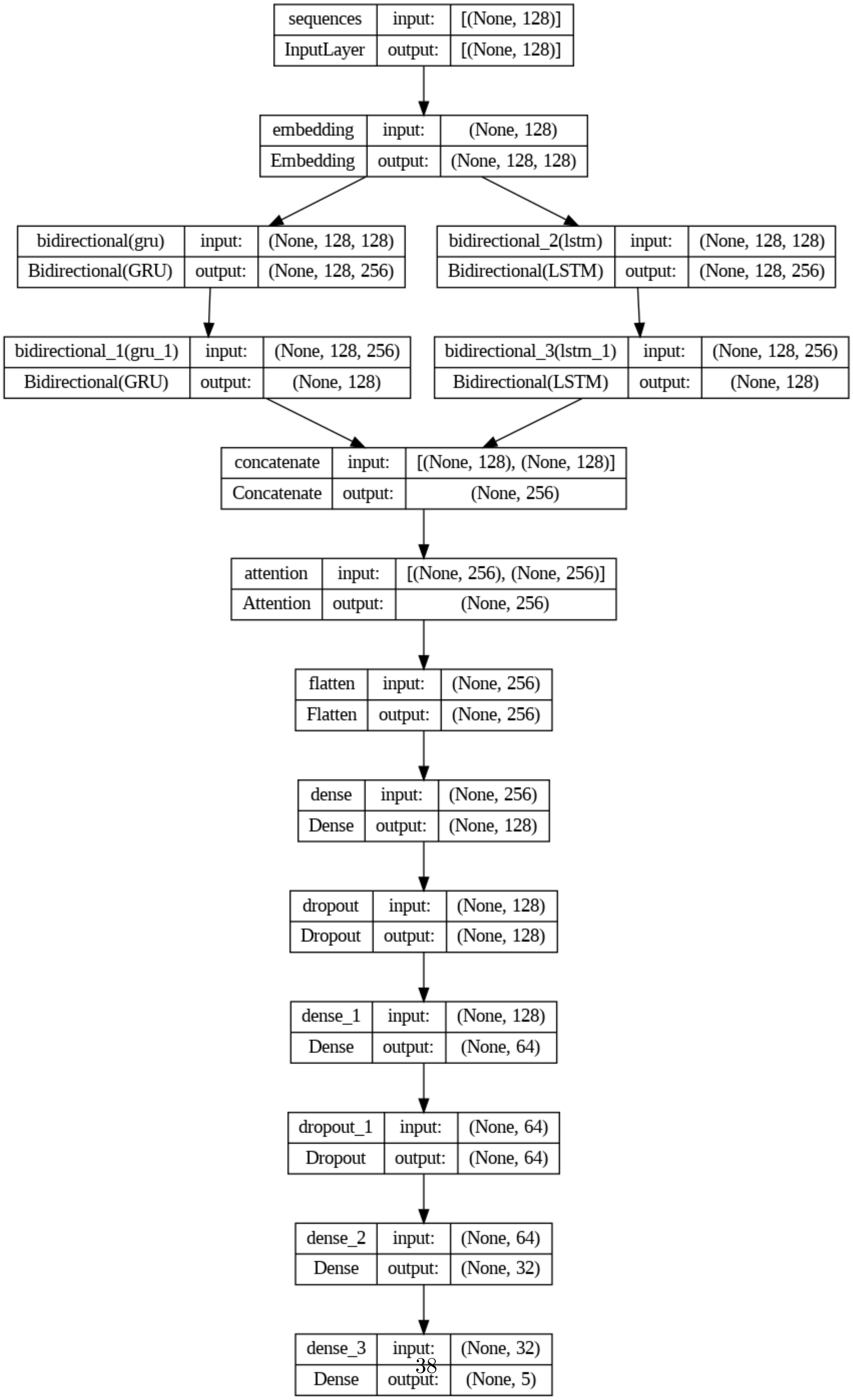
[ ]:
```
from keras.layers import Input, Embedding, Bidirectional, GRU, LSTM, Dense,␣
  ↪Flatten, Dropout, Attention, Concatenate
from keras.models import Model
from keras.optimizers import Adam

def HybridModel(vocab_size, embedding_dim=128, sequence_length=128):
    sequences = Input(shape=(sequence_length,), dtype='int32', name='sequences')

    embedded_sequences = Embedding(vocab_size, embedding_dim,␣
  ↪input_length=sequence_length)(sequences)

    # GRU layers
    gru_out = Bidirectional(GRU(128, return_sequences=True))(embedded_sequences)
    gru_out = Bidirectional(GRU(64))(gru_out)

    # LSTM layers
    lstm_out = Bidirectional(LSTM(128,␣
  ↪return_sequences=True))(embedded_sequences)
    lstm_out = Bidirectional(LSTM(64))(lstm_out)
    # Concatenate GRU and LSTM outputs
    concatenated = Concatenate()([gru_out, lstm_out])

    # Attention mechanism
    attention = Attention()([concatenated, concatenated])

    # Flatten and additional dense layers
    x = Flatten()(attention)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.3)(x)
    x = Dense(32, activation='relu')(x)

    num_classes = 5
    output = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=sequences, outputs=output)
    optimizer = Adam(learning_rate=0.001)  # Adjust the learning rate if needed
    model.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

    return model
```

[ ]:
```
vocab_size = tokenizer.vocab_size
model = HybridModel(vocab_size, sequence_length=128)
model.summary()
```

```
Model: "model"

________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #   Connected to
================================================================================
==================
 sequences (InputLayer)      [(None, 128)]                0         []

 embedding (Embedding)       (None, 128, 128)             1305280
['sequences[0][0]']
                                                          0

 bidirectional (Bidirection  (None, 128, 256)             198144
['embedding[0][0]']
 al)

 bidirectional_2 (Bidirecti  (None, 128, 256)             263168
['embedding[0][0]']
```

```
 onal)

 bidirectional_1 (Bidirecti  (None, 128)                  123648
['bidirectional[0][0]']
 onal)

 bidirectional_3 (Bidirecti  (None, 128)                  164352
['bidirectional_2[0][0]']
 onal)

 concatenate (Concatenate)   (None, 256)                  0
['bidirectional_1[0][0]',
'bidirectional_3[0][0]']

 attention (Attention)       (None, 256)                  0
['concatenate[0][0]',
'concatenate[0][0]']

 flatten (Flatten)           (None, 256)                  0
['attention[0][0]']

 dense (Dense)               (None, 128)                  32896
['flatten[0][0]']

 dropout (Dropout)           (None, 128)                  0
['dense[0][0]']

 dense_1 (Dense)             (None, 64)                   8256
['dropout[0][0]']

 dropout_1 (Dropout)         (None, 64)                   0
['dense_1[0][0]']

 dense_2 (Dense)             (None, 32)                   2080
['dropout_1[0][0]']

 dense_3 (Dense)             (None, 5)                    165
['dense_2[0][0]']

================================================================================
==================
Total params: 13845509 (52.82 MB)
Trainable params: 13845509 (52.82 MB)
Non-trainable params: 0 (0.00 Byte)
________________________________________________________________________________
__________________
```

```
[ ]: from tensorflow.keras.utils import plot_model

     # Save the model summary as an image file
     plot_model(model, to_file='hybrid_model_summary.png', show_shapes=True,␣
      ↪show_layer_names=True)

[ ]:
```


sequences | input: | [(None, 128)]
InputLayer | output: | [(None, 128)]
embedding | input: | (None, 128)
Embedding | output: | (None, 128, 128)
bidirectional(gru) | input: | (None, 128, 128)
Bidirectional(GRU) | output: | (None, 128, 256)
bidirectional_2(lstm) | input: | (None, 128, 128)
Bidirectional(LSTM) | output: | (None, 128, 256)
bidirectional_1(gru_1) | input: | (None, 128, 256)
Bidirectional(GRU) | output: | (None, 128)
bidirectional_3(lstm_1) | input: | (None, 128, 256)
Bidirectional(LSTM) | output: | (None, 128)
concatenate | input: | [(None, 128), (None, 128)]
Concatenate | output: | (None, 256)
attention | input: | [(None, 256), (None, 256)]
Attention | output: | (None, 256)
flatten | input: | (None, 256)
Flatten | output: | (None, 256)
dense | input: | (None, 256)
Dense | output: | (None, 128)
dropout | input: | (None, 128)
Dropout | output: | (None, 128)
dense_1 | input: | (None, 128)
Dense | output: | (None, 64)
dropout_1 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_2 | input: | (None, 64)
Dense | output: | (None, 32)
dense_3 | input: | (None, 32)
Dense | output: | (None, 5)

[ ]:
```
# Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True)
```

[ ]:
```
# Train the model
history = model.fit(
    train_encodings['input_ids'],
    train_labels_onehot,
    validation_data=(test_encodings['input_ids'],
                     test_labels_onehot),
    epochs=20,
    batch_size=32,
    callbacks=[early_stopping]
)
```

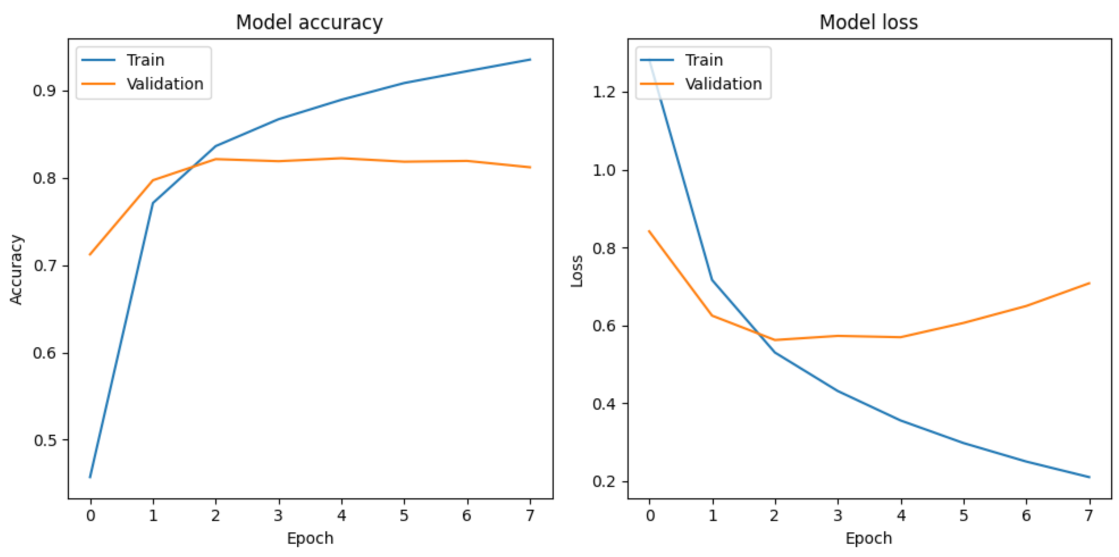
```
Epoch 1/20
1100/1100 [==============================] - 100s 76ms/step - loss: 1.2824 -
accuracy: 0.4571 - val_loss: 0.8415 - val_accuracy: 0.7123
Epoch 2/20
1100/1100 [==============================] - 56s 51ms/step - loss: 0.7167 -
accuracy: 0.7710 - val_loss: 0.6251 - val_accuracy: 0.7972
Epoch 3/20
1100/1100 [==============================] - 51s 46ms/step - loss: 0.5304 -
accuracy: 0.8363 - val_loss: 0.5625 - val_accuracy: 0.8214
Epoch 4/20
1100/1100 [==============================] - 52s 47ms/step - loss: 0.4316 -
accuracy: 0.8671 - val_loss: 0.5731 - val_accuracy: 0.8190
Epoch 5/20
1100/1100 [==============================] - 50s 46ms/step - loss: 0.3559 -
accuracy: 0.8894 - val_loss: 0.5697 - val_accuracy: 0.8224
Epoch 6/20
1100/1100 [==============================] - 52s 47ms/step - loss: 0.2980 -
accuracy: 0.9085 - val_loss: 0.6063 - val_accuracy: 0.8184
Epoch 7/20
1100/1100 [==============================] - 52s 47ms/step - loss: 0.2504 -
accuracy: 0.9221 - val_loss: 0.6499 - val_accuracy: 0.8193
Epoch 8/20
1100/1100 [==============================] - 51s 46ms/step - loss: 0.2105 -
accuracy: 0.9354 - val_loss: 0.7081 - val_accuracy: 0.8120
```

[ ]:
```
# Save the entire model to a HDF5 file
model.save('/content/drive/MyDrive/Bully/GRU_Model')
```

[ ]:
```python
# Evaluate the model on test data
loss, accuracy = model.evaluate(test_dataset.batch(32))
print(f'Test Accuracy: {accuracy * 100:.2f}%')
print(f'Test Loss: {loss:.4f}')

# Get predictions for test data
predictions = model.predict(test_dataset.batch(32))
predicted_classes = predictions.argmax(axis=1)
true_classes = test_labels
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/functional.py:642:
UserWarning: Input dict contained keys ['attention_mask'] which did not match
any model input. They will be ignored by the model.
  inputs = self._flatten_to_reference_inputs(inputs)

275/275 [==============================] - 7s 17ms/step - loss: 0.5625 -
accuracy: 0.8214
Test Accuracy: 82.14%
Test Loss: 0.5625
275/275 [==============================] - 7s 15ms/step
```

[ ]:
```python
# Plot training & validation accuracy values
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
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 left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
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 left')
plt.tight_layout()
plt.show()
```

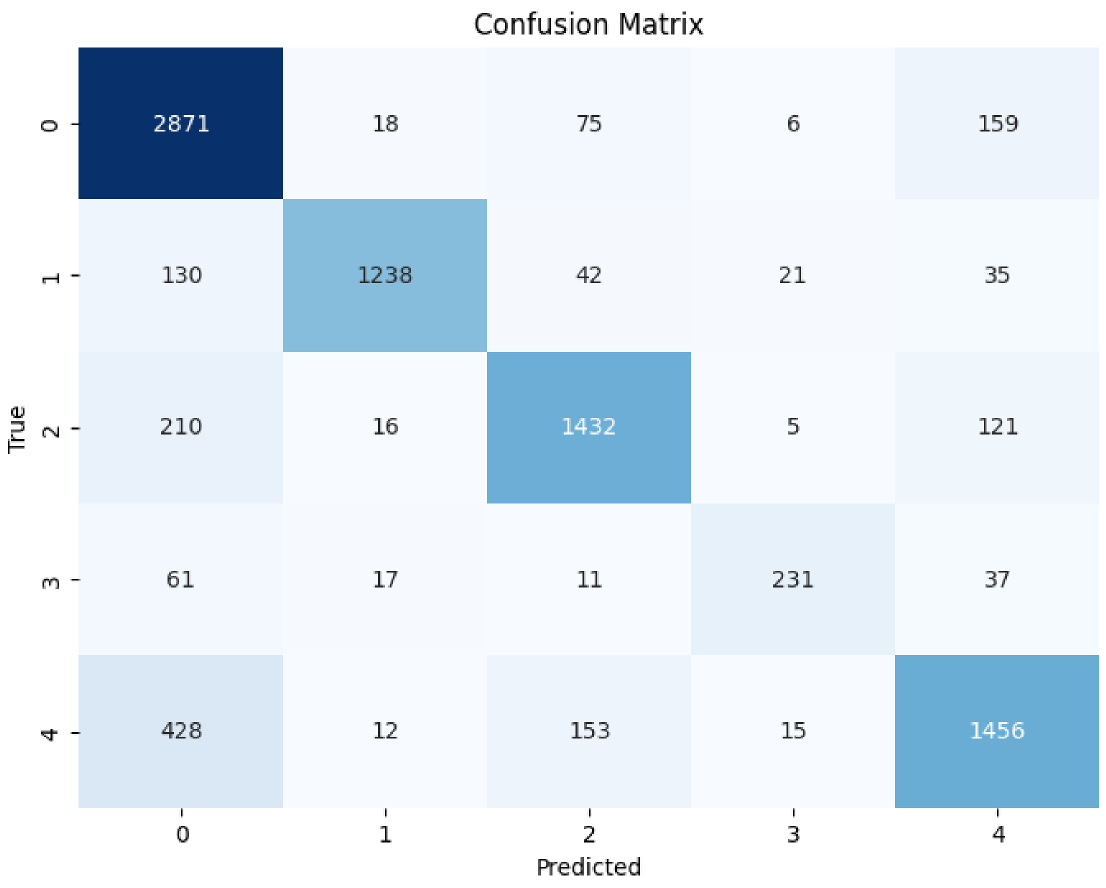


```
[ ]: from sklearn.metrics import classification_report, confusion_matrix,␣
      ↪precision_recall_fscore_support
```

```
[ ]: # Classification Report
     class_report = classification_report(true_classes, predicted_classes)
     print("Classification Report:\n", class_report)
     # Confusion Matrix
     conf_matrix = confusion_matrix(true_classes, predicted_classes)
     print("Confusion Matrix:\n", conf_matrix)
     # Plot Confusion Matrix
     plt.figure(figsize=(8, 6))
     sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False)
     plt.title('Confusion Matrix')
     plt.xlabel('Predicted')
     plt.ylabel('True')
     plt.show()
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.78      0.92      0.84      3129
           1       0.95      0.84      0.89      1466
           2       0.84      0.80      0.82      1784
           3       0.83      0.65      0.73       357
           4       0.81      0.71      0.75      2064

    accuracy                           0.82      8800
   macro avg       0.84      0.78      0.81      8800
weighted avg       0.83      0.82      0.82      8800
```

```
Confusion Matrix:
 [[2871   18   75    6  159]
 [ 130 1238   42   21   35]
 [ 210   16 1432    5  121]
 [  61   17   11  231   37]
 [ 428   12  153   15 1456]]
```



```
[ ]: # Extract TP, TN, FP, FN from confusion matrix
     TP = conf_matrix[1, 1]  # True Positives
     TN = conf_matrix[0, 0]  # True Negatives
     FP = conf_matrix[0, 1]  # False Positives
     FN = conf_matrix[1, 0]  # False Negatives

     print("True Positives:", TP)
     print("True Negatives:", TN)
     print("False Positives:", FP)
     print("False Negatives:", FN)
```

```
True Positives: 1238
True Negatives: 2871
False Positives: 18
False Negatives: 130
```

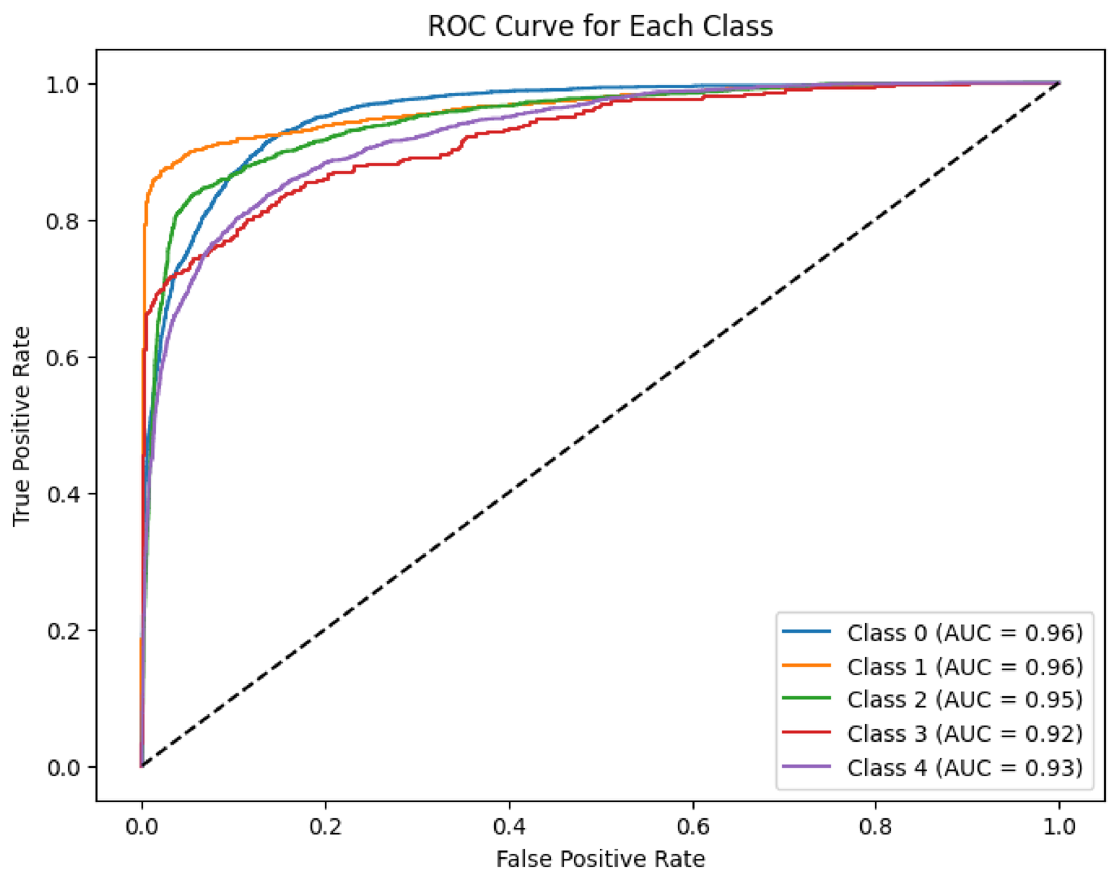
[ ]:
```
# Precision, Recall, F1-score
precision, recall, f1_score, _ = precision_recall_fscore_support(true_classes,␣
↪predicted_classes, average='weighted')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1_score:.4f}')
```

```
Precision: 0.8265
Recall: 0.8214
F1-score: 0.8200
```

[ ]:
```
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve, roc_auc_score
```

[ ]:
```
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve, roc_auc_score

# Get predictions for test data
predictions = model.predict(test_dataset.batch(32))
predicted_classes = predictions.argmax(axis=1)
true_classes = test_labels

# Transform true labels to binary format
label_binarizer = LabelBinarizer()
true_labels_bin = label_binarizer.fit_transform(true_classes)

# Calculate ROC curve and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
num_labels = len(label_binarizer.classes_)

for i in range(num_labels):
    fpr[i], tpr[i], _ = roc_curve(true_labels_bin[:, i], predictions[:, i])
    roc_auc[i] = roc_auc_score(true_labels_bin[:, i], predictions[:, i])

# Plot ROC curve for each class
plt.figure(figsize=(8, 6))

for i in range(num_labels):
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal reference line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Each Class')
plt.legend(loc="lower right")
plt.show()
```

```
275/275 [==============================] - 5s 18ms/step
```



LSTM Model

```
[ ]: from tensorflow.keras.layers import Input, Embedding, LSTM, Dropout, Dense,␣
      ↪Bidirectional
     from tensorflow.keras.models import Model
```

```
[ ]: def LSTMmodel(vocab_size, embedding_dim=128, sequence_length=128):
         # Define input layer
         sequences = Input(shape=(sequence_length,), dtype=tf.int32,␣
      ↪name="sequences")

         embedded_sequences = Embedding(vocab_size, embedding_dim)(sequences)
```

```
    # LSTM layers
    lstm_out = Bidirectional(LSTM(128,␣
  ↪return_sequences=True))(embedded_sequences)
    lstm_out = Dropout(0.5)(lstm_out)
    lstm_out = LSTM(64, return_sequences=True)(lstm_out)
    lstm_out = Dropout(0.5)(lstm_out)

    # Attention layer
    attention = Attention()([lstm_out, lstm_out])
    x = Flatten()(attention)

    # Dense layers
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.5)(x)
    num_classes = len(set(train_labels))
    x = Dense(num_classes, activation='softmax')(x)

    return Model(inputs=sequences, outputs=x)
```

[ ]:
```
# Create the model
vocab_size = tokenizer.vocab_size
model = LSTMmodel(vocab_size)
print(model.summary())
```

```
Model: "model_1"
________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #   Connected to
================================================================================
==================
 sequences (InputLayer)      [(None, 128)]                0         []

 embedding_1 (Embedding)     (None, 128, 128)             1305280
['sequences[0][0]']
                                                          0

 bidirectional_4 (Bidirecti  (None, 128, 256)             263168
['embedding_1[0][0]']
 onal)

 dropout_2 (Dropout)         (None, 128, 256)             0
['bidirectional_4[0][0]']

 lstm_3 (LSTM)               (None, 128, 64)              82176
['dropout_2[0][0]']
```

```
 dropout_3 (Dropout)         (None, 128, 64)              0
['lstm_3[0][0]']

 attention_1 (Attention)     (None, 128, 64)              0
['dropout_3[0][0]',
'dropout_3[0][0]']

 flatten_1 (Flatten)         (None, 8192)                 0
['attention_1[0][0]']

 dense_4 (Dense)             (None, 64)                   524352
['flatten_1[0][0]']

 dropout_4 (Dropout)         (None, 64)                   0
['dense_4[0][0]']

 dense_5 (Dense)             (None, 5)                    325
['dropout_4[0][0]']

================================================================================
==================
Total params: 13922821 (53.11 MB)
Trainable params: 13922821 (53.11 MB)
Non-trainable params: 0 (0.00 Byte)
--------------------------------------------------------------------------------
------------------
None
```

[ ]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
  ↪metrics=['accuracy'])
```

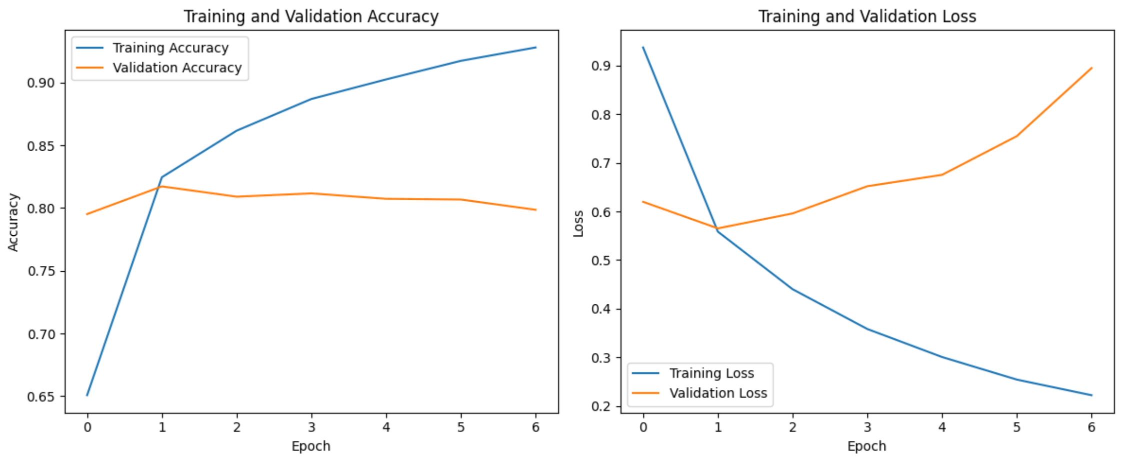
[ ]:
```
# Train the model
history_1 = model.fit(
    train_encodings['input_ids'],
    train_labels_onehot,
    validation_data=(test_encodings['input_ids'],
                     test_labels_onehot),
    epochs=10,
    batch_size=32,
    callbacks=[early_stopping]
)
```

```
Epoch 1/10
1100/1100 [==============================] - 64s 51ms/step - loss: 0.9375 -
accuracy: 0.6508 - val_loss: 0.6198 - val_accuracy: 0.7951
Epoch 2/10
1100/1100 [==============================] - 29s 26ms/step - loss: 0.5585 -
accuracy: 0.8245 - val_loss: 0.5652 - val_accuracy: 0.8172
```

```
Epoch 3/10
1100/1100 [==============================] - 26s 24ms/step - loss: 0.4400 -
accuracy: 0.8616 - val_loss: 0.5959 - val_accuracy: 0.8090
Epoch 4/10
1100/1100 [==============================] - 25s 23ms/step - loss: 0.3581 -
accuracy: 0.8868 - val_loss: 0.6519 - val_accuracy: 0.8116
Epoch 5/10
1100/1100 [==============================] - 25s 23ms/step - loss: 0.3004 -
accuracy: 0.9024 - val_loss: 0.6754 - val_accuracy: 0.8073
Epoch 6/10
1100/1100 [==============================] - 25s 23ms/step - loss: 0.2540 -
accuracy: 0.9172 - val_loss: 0.7550 - val_accuracy: 0.8067
Epoch 7/10
1100/1100 [==============================] - 25s 23ms/step - loss: 0.2218 -
accuracy: 0.9279 - val_loss: 0.8951 - val_accuracy: 0.7985
```

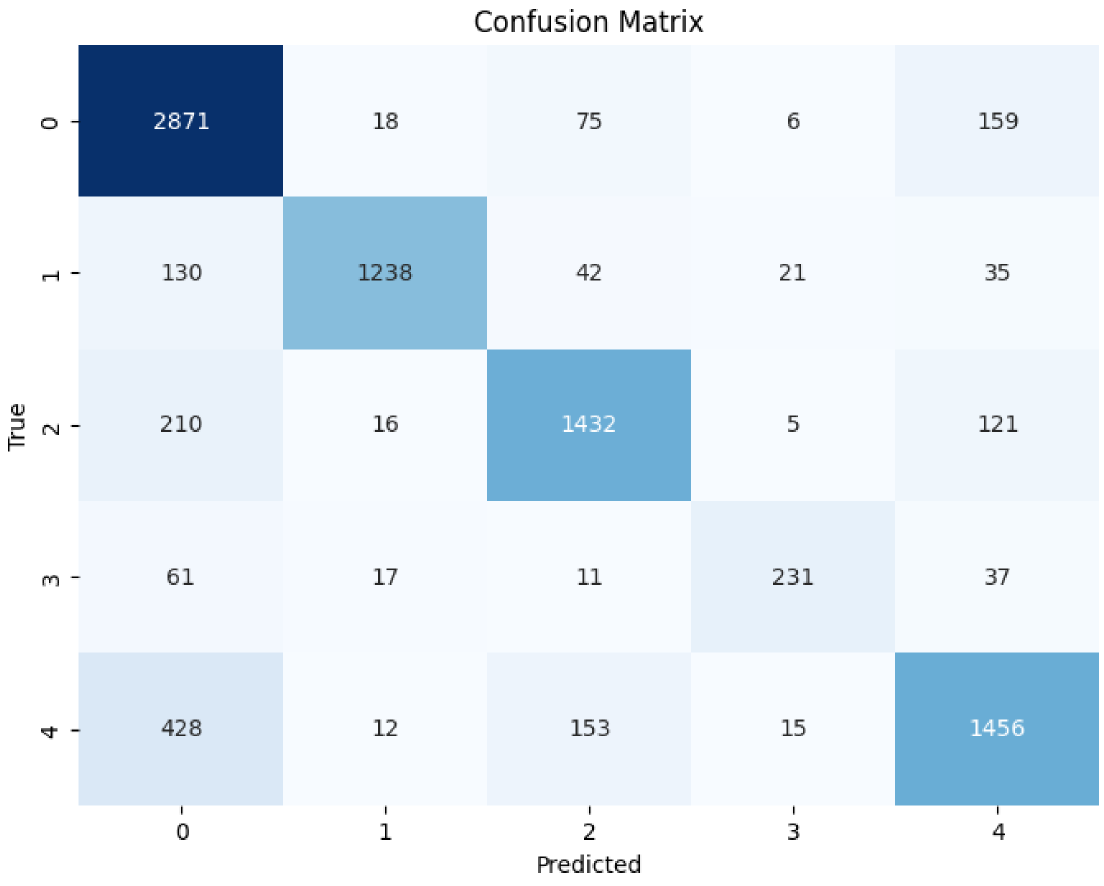
[ ]:

```python
# Plotting training & validation accuracy
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history_1.history['accuracy'], label='Training Accuracy')
plt.plot(history_1.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

# Plotting training & validation loss
plt.subplot(1, 2, 2)
plt.plot(history_1.history['loss'], label='Training Loss')
plt.plot(history_1.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')

plt.tight_layout()
plt.show()
```



```
[ ]: # Classification Report
     class_report = classification_report(true_classes, predicted_classes)
     print("Classification Report:\n", class_report)
     # Confusion Matrix
     conf_matrix = confusion_matrix(true_classes, predicted_classes)
     print("Confusion Matrix:\n", conf_matrix)
     # Plot Confusion Matrix
     plt.figure(figsize=(8, 6))
     sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False)
     plt.title('Confusion Matrix')
     plt.xlabel('Predicted')
     plt.ylabel('True')
     plt.show()
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.78      0.92      0.84      3129
           1       0.95      0.84      0.89      1466
           2       0.84      0.80      0.82      1784
           3       0.83      0.65      0.73       357
           4       0.81      0.71      0.75      2064

    accuracy                           0.82      8800
   macro avg       0.84      0.78      0.81      8800
weighted avg       0.83      0.82      0.82      8800

Confusion Matrix:
 [[2871   18   75    6  159]
 [ 130 1238   42   21   35]
 [ 210   16 1432    5  121]
 [  61   17   11  231   37]
 [ 428   12  153   15 1456]]
```



[ ]:
```python
# Extract TP, TN, FP, FN from confusion matrix
TP = conf_matrix[1, 1]  # True Positives
TN = conf_matrix[0, 0]  # True Negatives
FP = conf_matrix[0, 1]  # False Positives
FN = conf_matrix[1, 0]  # False Negatives

print("True Positives:", TP)
print("True Negatives:", TN)
print("False Positives:", FP)
print("False Negatives:", FN)
```

```
True Positives: 1238
True Negatives: 2871
False Positives: 18
False Negatives: 130
```

[ ]:
```
# Evaluate the model on test data
loss, accuracy = model.evaluate(test_dataset.batch(32))
print(f'Test Accuracy: {accuracy * 100:.2f}%')
print(f'Test Loss: {loss:.4f}')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/functional.py:642:
UserWarning: Input dict contained keys ['attention_mask'] which did not match
any model input. They will be ignored by the model.
  inputs = self._flatten_to_reference_inputs(inputs)
```

```
275/275 [==============================] - 3s 8ms/step - loss: 0.5652 -
accuracy: 0.8172
Test Accuracy: 81.72%
Test Loss: 0.5652
```

[ ]:
```
# Precision, Recall, F1-score
precision, recall, f1_score, _ = precision_recall_fscore_support(true_classes,␣
 ↪predicted_classes, average='weighted')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1_score:.4f}')
```

```
Precision: 0.8265
Recall: 0.8214
F1-score: 0.8200
```

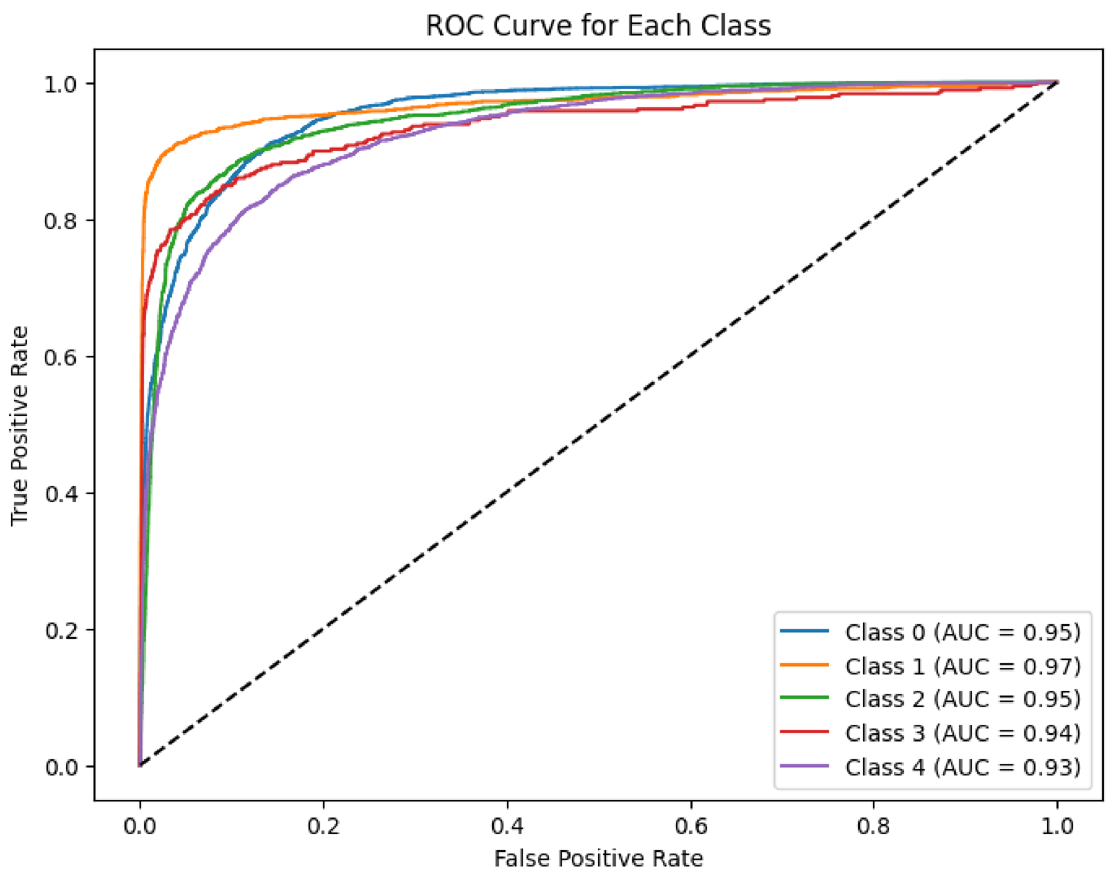
[ ]:
```
# Save the entire model to a HDF5 file
model.save('/content/drive/MyDrive/Bully/LSTM_Model')
```

[ ]:
```
# Get predictions for test data
predictions = model.predict(test_dataset.batch(32))
predicted_classes = predictions.argmax(axis=1)
true_classes = test_labels

# Transform true labels to binary format
label_binarizer = LabelBinarizer()
true_labels_bin = label_binarizer.fit_transform(true_classes)

# Calculate ROC curve and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
num_labels = len(label_binarizer.classes_)

for i in range(num_labels):
    fpr[i], tpr[i], _ = roc_curve(true_labels_bin[:, i], predictions[:, i])
    roc_auc[i] = roc_auc_score(true_labels_bin[:, i], predictions[:, i])
```

```
# Plot ROC curve for each class
plt.figure(figsize=(8, 6))

for i in range(num_labels):
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal reference line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Each Class')
plt.legend(loc="lower right")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/functional.py:642:
UserWarning: Input dict contained keys ['attention_mask'] which did not match
any model input. They will be ignored by the model.
  inputs = self._flatten_to_reference_inputs(inputs)
```

```
275/275 [==============================] - 4s 9ms/step
```



CNN Model

[ ]:
```
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten
```

[ ]:
```
def CNNmodel(vocab_size, embedding_dim=128, sequence_length=128):
    # Define input layer
    sequences = Input(shape=(sequence_length,), dtype=tf.int32,␣
 ↪name="sequences")

    embedded_sequences = Embedding(vocab_size, embedding_dim)(sequences)

    # 1D Convolution layers
    x = Conv1D(128, 5, activation='relu')(embedded_sequences)
    x = MaxPooling1D(5)(x)
    x = Conv1D(128, 5, activation='relu')(x)
    x = MaxPooling1D(5)(x)

    # Attention layer
    attention = Attention()([x, x])

    # GlobalMaxPooling1D
    x = Flatten()(attention)
    x = Dense(128, activation='relu')(x)
    x = Dense(64, activation='relu')(x)
    x = Dense(32, activation='relu')(x)

    # Output layer
    num_classes = len(set(train_labels))
    x = Dense(num_classes, activation='softmax')(x)

    return Model(inputs=sequences, outputs=x)
```

[ ]:
```
# Create the model
vocab_size = tokenizer.vocab_size
model = CNNmodel(vocab_size)
print(model.summary())
```

```
Model: "model_2"
________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #   Connected to
================================================================================
==================
 sequences (InputLayer)      [(None, 128)]                0         []

 embedding_2 (Embedding)     (None, 128, 128)             1305280
['sequences[0][0]']

                                                          0
 conv1d (Conv1D)             (None, 124, 128)             82048
['embedding_2[0][0]']

 max_pooling1d (MaxPooling1  (None, 24, 128)              0
['conv1d[0][0]']
 D)

 conv1d_1 (Conv1D)           (None, 20, 128)              82048
['max_pooling1d[0][0]']

 max_pooling1d_1 (MaxPoolin  (None, 4, 128)               0
['conv1d_1[0][0]']
 g1D)

 attention_2 (Attention)     (None, 4, 128)               0
['max_pooling1d_1[0][0]',
'max_pooling1d_1[0][0]']

 flatten_2 (Flatten)         (None, 512)                  0
['attention_2[0][0]']

 dense_6 (Dense)             (None, 128)                  65664
['flatten_2[0][0]']

 dense_7 (Dense)             (None, 64)                   8256
['dense_6[0][0]']

 dense_8 (Dense)             (None, 32)                   2080
['dense_7[0][0]']

 dense_9 (Dense)             (None, 5)                    165
['dense_8[0][0]']

================================================================================
==================
Total params: 13293061 (50.71 MB)
Trainable params: 13293061 (50.71 MB)
Non-trainable params: 0 (0.00 Byte)
--------------------------------------------------------------------------------
------------------
None
```

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

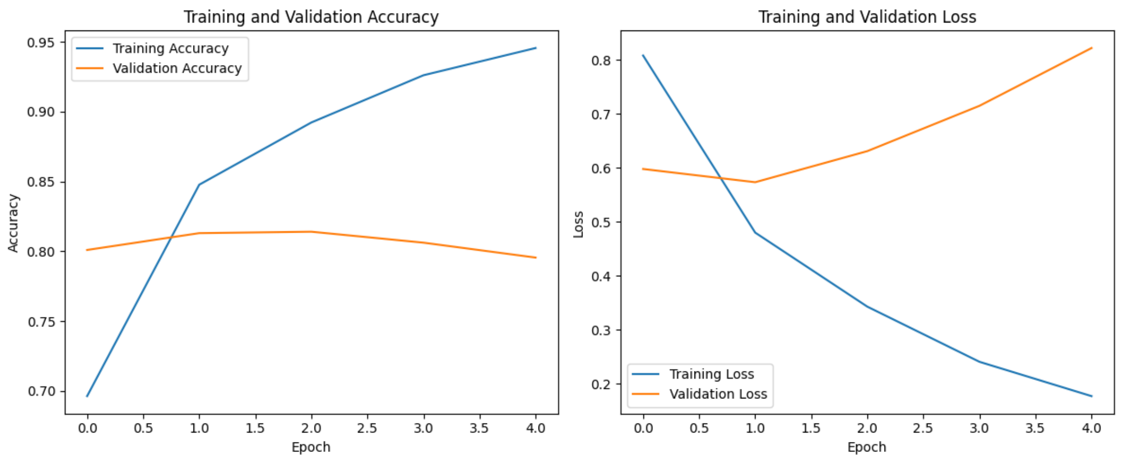
```
[ ]: from tensorflow.keras.callbacks import EarlyStopping
```

```
[ ]: early_stopping = EarlyStopping(monitor='val_loss', patience=3,␣
      ↪restore_best_weights=True, verbose=1)
```

```
[ ]: history_2 = model.fit(train_encodings['input_ids'], train_labels_onehot,
                         validation_data=(test_encodings['input_ids'],␣
      ↪test_labels_onehot),
                         epochs=10, batch_size=32, callbacks=[early_stopping])
```

```
Epoch 1/10
1100/1100 [==============================] - 63s 50ms/step - loss: 0.8079 -
accuracy: 0.6963 - val_loss: 0.5978 - val_accuracy: 0.8009
Epoch 2/10
1100/1100 [==============================] - 14s 13ms/step - loss: 0.4800 -
accuracy: 0.8476 - val_loss: 0.5734 - val_accuracy: 0.8130
Epoch 3/10
1100/1100 [==============================] - 13s 12ms/step - loss: 0.3430 -
accuracy: 0.8922 - val_loss: 0.6309 - val_accuracy: 0.8140
Epoch 4/10
1100/1100 [==============================] - 12s 11ms/step - loss: 0.2409 -
accuracy: 0.9260 - val_loss: 0.7148 - val_accuracy: 0.8061
Epoch 5/10
1099/1100 [============================>.] - ETA: 0s - loss: 0.1774 - accuracy:
0.9455Restoring model weights from the end of the best epoch: 2.
1100/1100 [==============================] - 11s 10ms/step - loss: 0.1773 -
accuracy: 0.9455 - val_loss: 0.8218 - val_accuracy: 0.7955
Epoch 5: early stopping
```

```
[ ]: plt.figure(figsize=(12, 5))

     # Accuracy plot
     plt.subplot(1, 2, 1)
     plt.plot(history_2.history['accuracy'], label='Training Accuracy')
     plt.plot(history_2.history['val_accuracy'], label='Validation Accuracy')
     plt.xlabel('Epoch')
     plt.ylabel('Accuracy')
     plt.legend()
     plt.title('Training and Validation Accuracy')

     # Loss plot
     plt.subplot(1, 2, 2)
     plt.plot(history_2.history['loss'], label='Training Loss')
     plt.plot(history_2.history['val_loss'], label='Validation Loss')
     plt.xlabel('Epoch')
     plt.ylabel('Loss')
     plt.legend()
     plt.title('Training and Validation Loss')

     plt.tight_layout()
     plt.show()
```

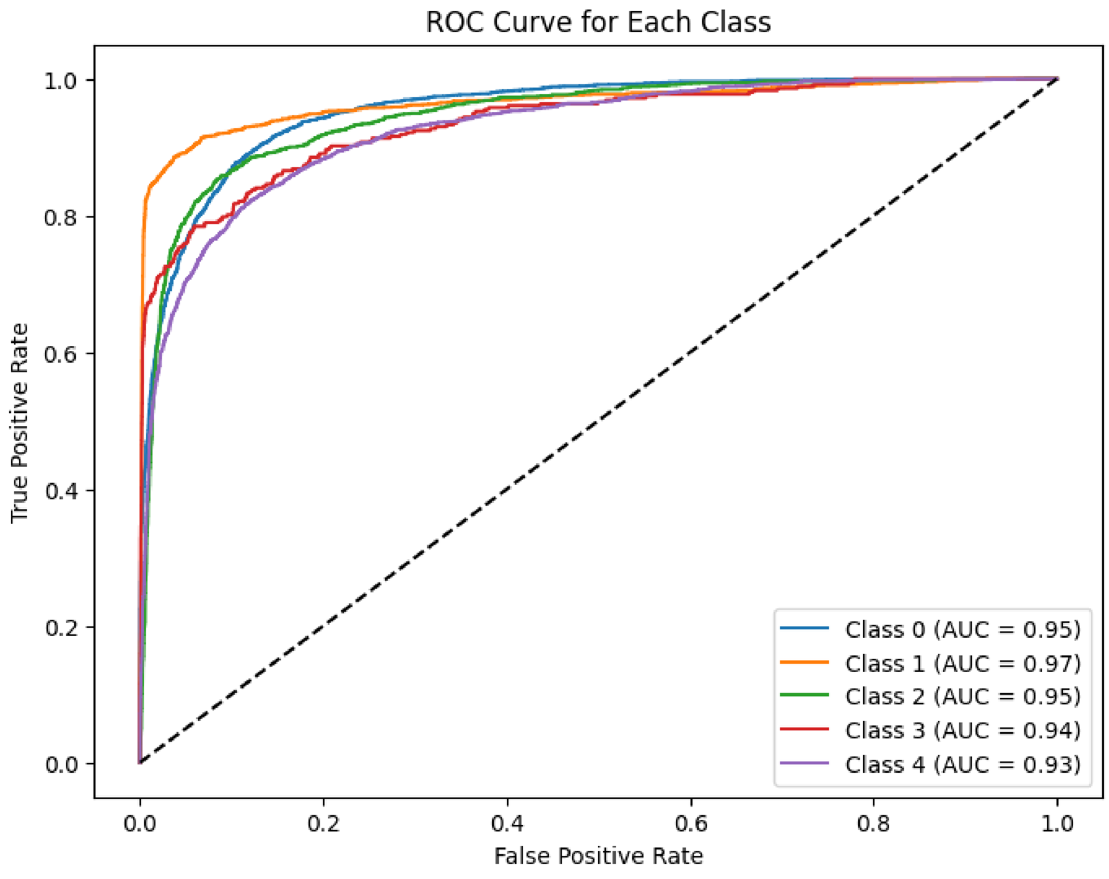


[ ]:
```
# Get predictions for test data
predictions = model.predict(test_dataset.batch(32))
predicted_classes = predictions.argmax(axis=1)
true_classes = test_labels

# Transform true labels to binary format
label_binarizer = LabelBinarizer()
true_labels_bin = label_binarizer.fit_transform(true_classes)

# Calculate ROC curve and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
num_labels = len(label_binarizer.classes_)

for i in range(num_labels):
    fpr[i], tpr[i], _ = roc_curve(true_labels_bin[:, i], predictions[:, i])
    roc_auc[i] = roc_auc_score(true_labels_bin[:, i], predictions[:, i])

# Plot ROC curve for each class
plt.figure(figsize=(8, 6))

for i in range(num_labels):
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal reference line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Each Class')
plt.legend(loc="lower right")
plt.show()
```

```
 53/275 [====>…] - ETA: 0s
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/functional.py:642:
UserWarning: Input dict contained keys ['attention_mask'] which did not match
any model input. They will be ignored by the model.
  inputs = self._flatten_to_reference_inputs(inputs)
```

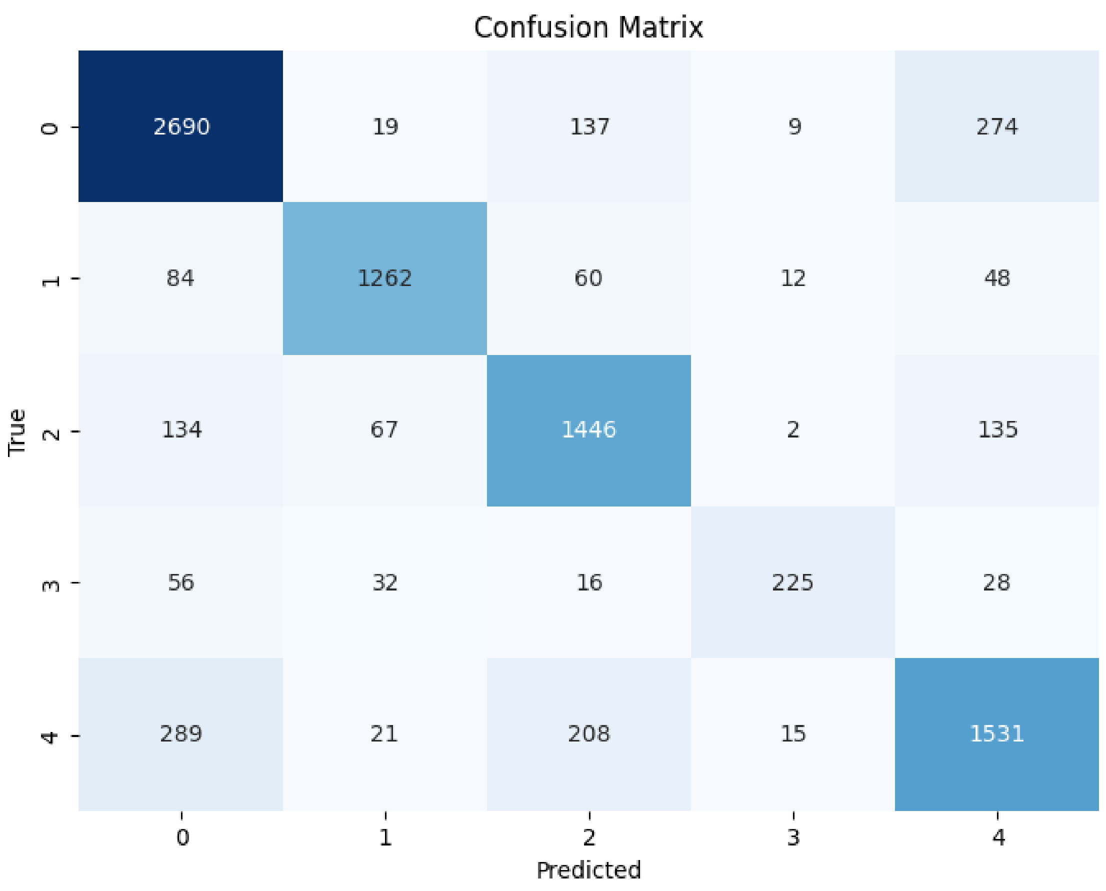
```
275/275 [==============================] - 1s 2ms/step
```



[ ]:
```
# Evaluate the model on test data
loss, accuracy = model.evaluate(test_dataset.batch(32))
print(f'Test Accuracy: {accuracy * 100:.2f}%')
print(f'Test Loss: {loss:.4f}')
```

```
# Classification Report
class_report = classification_report(true_classes, predicted_classes)
print("Classification Report:\n", class_report)
# Confusion Matrix
conf_matrix = confusion_matrix(true_classes, predicted_classes)
print("Confusion Matrix:\n", conf_matrix)
# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Precision, Recall, F1-score
precision, recall, f1_score, _ = precision_recall_fscore_support(true_classes,␣
 ↪predicted_classes, average='weighted')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1_score:.4f}')
```

```
275/275 [==============================] - 1s 3ms/step - loss: 0.5734 -
accuracy: 0.8130
Test Accuracy: 81.30%
Test Loss: 0.5734
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.86      0.84      3129
           1       0.90      0.86      0.88      1466
           2       0.77      0.81      0.79      1784
           3       0.86      0.63      0.73       357
           4       0.76      0.74      0.75      2064

    accuracy                           0.81      8800
   macro avg       0.82      0.78      0.80      8800
weighted avg       0.81      0.81      0.81      8800

Confusion Matrix:
 [[2690   19  137    9  274]
 [  84 1262   60   12   48]
 [ 134   67 1446    2  135]
 [  56   32   16  225   28]
 [ 289   21  208   15 1531]]
```



```
Precision: 0.8139
Recall: 0.8130
F1-score: 0.8125
```

[ ]:
```
# Extract TP, TN, FP, FN from confusion matrix
TP = conf_matrix[1, 1]  # True Positives
TN = conf_matrix[0, 0]  # True Negatives
FP = conf_matrix[0, 1]  # False Positives
FN = conf_matrix[1, 0]  # False Negatives

print("True Positives:", TP)
print("True Negatives:", TN)
print("False Positives:", FP)
print("False Negatives:", FN)
```

```
True Positives: 1262
True Negatives: 2690
False Positives: 19
False Negatives: 84
```

```
[ ]: # Save the entire model to a HDF5 file
     model.save('/content/drive/MyDrive/Bully/CNN_Model')
```

Ensemble Model

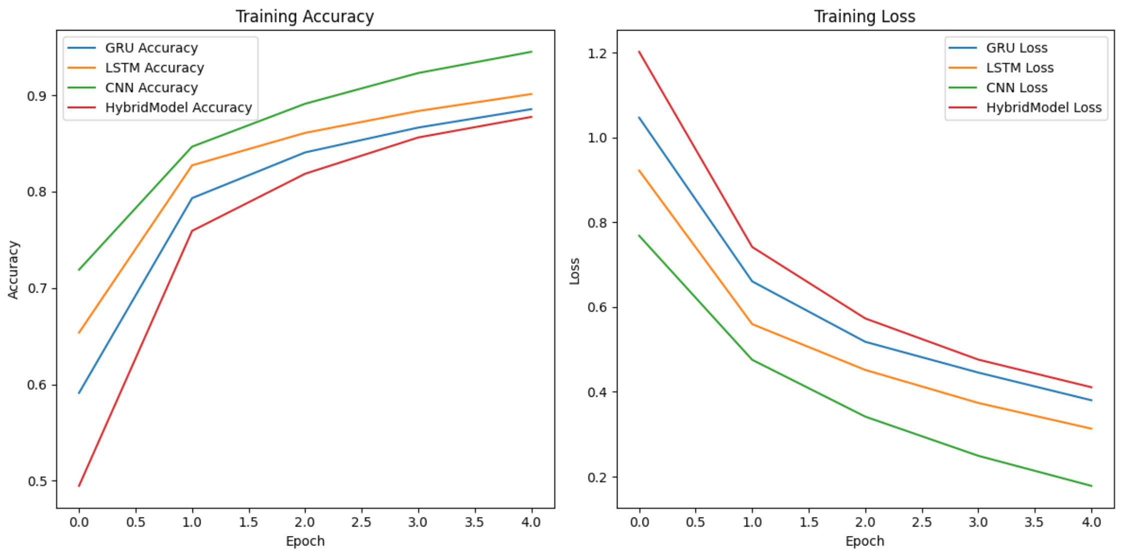
```
[ ]: models = {'GRU': GRUmodel(tokenizer.vocab_size),
               'LSTM': LSTMmodel(tokenizer.vocab_size),
               'CNN': CNNmodel(tokenizer.vocab_size),
               'HybridModel' : HybridModel(tokenizer.vocab_size)}
```

```
[ ]: histories = {}  # To store training histories of each model

     for name, model in models.items():
         print(f"Training {name} model...")
         model.compile(optimizer='adam', loss='categorical_crossentropy',␣
      ↪metrics=['accuracy'])
         history = model.fit(train_encodings['input_ids'], train_labels_onehot,␣
      ↪epochs=5, batch_size=32)
         histories[name] = history
```

```
Training GRU model…
Epoch 1/5
1100/1100 [==============================] - 92s 75ms/step - loss: 1.0465 -
accuracy: 0.5912
Epoch 2/5
1100/1100 [==============================] - 36s 33ms/step - loss: 0.6605 -
accuracy: 0.7933
Epoch 3/5
1100/1100 [==============================] - 37s 34ms/step - loss: 0.5180 -
accuracy: 0.8407
Epoch 4/5
1100/1100 [==============================] - 34s 31ms/step - loss: 0.4455 -
accuracy: 0.8665
Epoch 5/5
1100/1100 [==============================] - 45s 41ms/step - loss: 0.3802 -
accuracy: 0.8856
Training LSTM model…
Epoch 1/5
1100/1100 [==============================] - 87s 70ms/step - loss: 0.9218 -
accuracy: 0.6537
Epoch 2/5
1100/1100 [==============================] - 33s 30ms/step - loss: 0.5596 -
accuracy: 0.8273
Epoch 3/5
1100/1100 [==============================] - 27s 24ms/step - loss: 0.4517 -
accuracy: 0.8610
Epoch 4/5
1100/1100 [==============================] - 24s 22ms/step - loss: 0.3739 -
```

```
accuracy: 0.8837
Epoch 5/5
1100/1100 [==============================] - 23s 21ms/step - loss: 0.3132 -
accuracy: 0.9013
Training CNN model…
Epoch 1/5
1100/1100 [==============================] - 40s 35ms/step - loss: 0.7682 -
accuracy: 0.7190
Epoch 2/5
1100/1100 [==============================] - 13s 12ms/step - loss: 0.4756 -
accuracy: 0.8467
Epoch 3/5
1100/1100 [==============================] - 12s 11ms/step - loss: 0.3416 -
accuracy: 0.8912
Epoch 4/5
1100/1100 [==============================] - 9s 9ms/step - loss: 0.2493 -
accuracy: 0.9231
Epoch 5/5
1100/1100 [==============================] - 11s 10ms/step - loss: 0.1784 -
accuracy: 0.9451
Training HybridModel model…
Epoch 1/5
1100/1100 [==============================] - 88s 68ms/step - loss: 1.2017 -
accuracy: 0.4947
Epoch 2/5
1100/1100 [==============================] - 51s 47ms/step - loss: 0.7415 -
accuracy: 0.7593
Epoch 3/5
1100/1100 [==============================] - 48s 43ms/step - loss: 0.5731 -
accuracy: 0.8186
Epoch 4/5
1100/1100 [==============================] - 48s 43ms/step - loss: 0.4760 -
accuracy: 0.8562
Epoch 5/5
1100/1100 [==============================] - 47s 42ms/step - loss: 0.4109 -
accuracy: 0.8775
```

[ ]:
```python
# Plotting training accuracy and loss for each model
plt.figure(figsize=(12, 6))

for name, history in histories.items():
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label=f'{name} Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Training Accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label=f'{name} Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training Loss')
    plt.legend()

plt.tight_layout()
plt.show()
```



[ ]:
```
train_predictions = {}
test_predictions = {}

for name, model in models.items():
    train_predictions[name] = model.predict(train_encodings['input_ids'])
    test_predictions[name] = model.predict(test_encodings['input_ids'])
```

```
1100/1100 [==============================] - 11s 10ms/step
275/275 [==============================] - 3s 10ms/step
1100/1100 [==============================] - 9s 8ms/step
275/275 [==============================] - 2s 7ms/step
1100/1100 [==============================] - 2s 2ms/step
275/275 [==============================] - 1s 2ms/step
1100/1100 [==============================] - 18s 16ms/step
275/275 [==============================] - 4s 15ms/step
```

[ ]:
```
stacked_train_predictions = np.column_stack([train_predictions[name] for name␣
 ↪in models])
stacked_test_predictions = np.column_stack([test_predictions[name] for name in␣
 ↪models])
```

[ ]:
```
stacked_train, stacked_val, train_labels_train, train_labels_val =␣
 ↪train_test_split(
    stacked_train_predictions,
    train_labels_onehot,
    test_size=0.2,
    random_state=42
)
```

[ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

[ ]:
```
# Initialize the Random Forest model
rf_meta_learner = RandomForestClassifier(n_estimators=100, random_state=42)
rf_meta_learner.fit(stacked_train, train_labels_train)
rf_meta_predictions = rf_meta_learner.predict(stacked_val)
```

[ ]:
```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(train_labels_val, rf_meta_predictions)
print(f"Accuracy of Random Forest meta-learner: {accuracy * 100:.2f}%")
```

```
Accuracy of Random Forest meta-learner: 96.38%
```

[ ]:
```
from sklearn.metrics import precision_recall_fscore_support

precision, recall, f1_score, _ =␣
 ↪precision_recall_fscore_support(train_labels_val, rf_meta_predictions,␣
 ↪average='weighted')

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1_score:.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      2483
           1       0.97      0.99      0.98      1209
           2       0.97      0.98      0.97      1452
           3       0.92      0.91      0.91       269
           4       0.96      0.94      0.95      1627

    accuracy                           0.97      7040
```

```
   macro avg       0.96      0.96      0.96      7040
weighted avg       0.97      0.97      0.97      7040
```

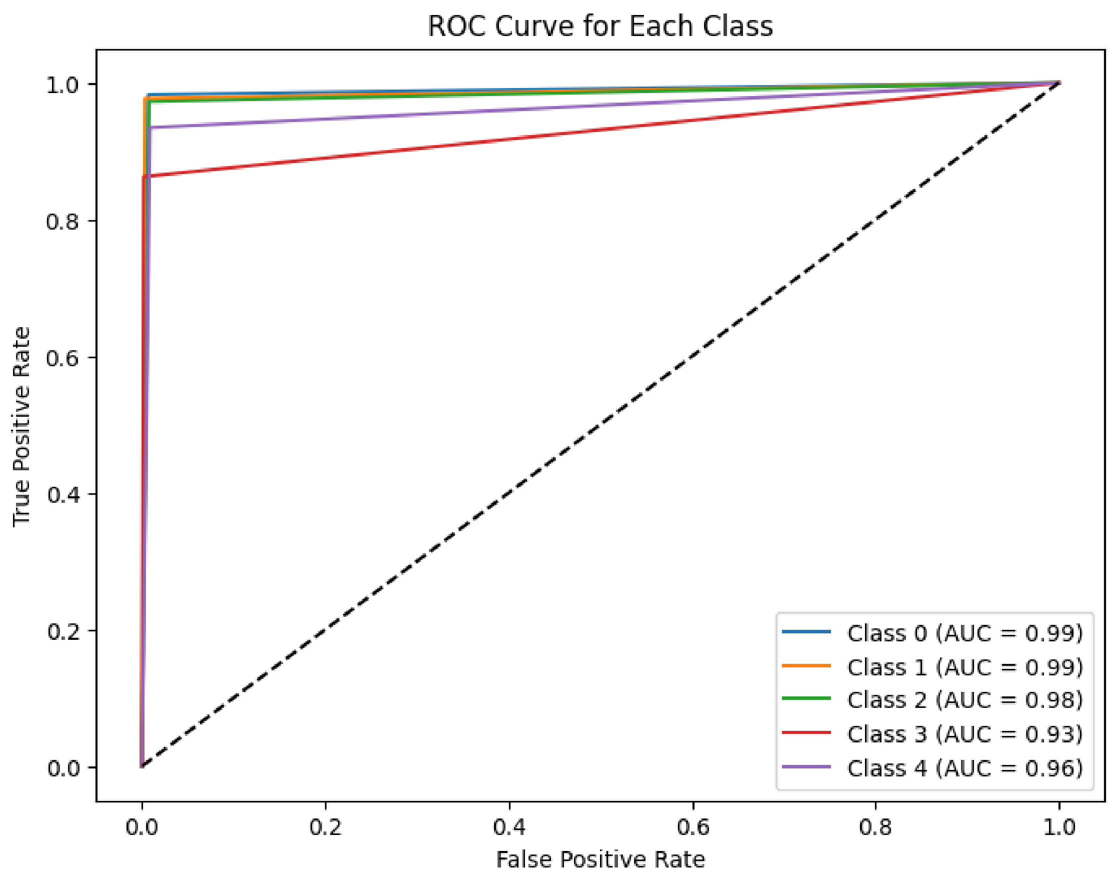
```
[ ]: from sklearn.metrics import precision_recall_fscore_support
```

```
[ ]: # Extract precision, recall, and f1-score
     precision, recall, f1_score, _ = precision_recall_fscore_support(true_labels,␣
      ↪log_reg_predictions, average='weighted')

     # Print the results
     print(f"Precision: {precision:.4f}")
     print(f"Recall: {recall:.4f}")
     print(f"F1 Score: {f1_score:.4f}")
```

```
Precision: 0.9704
Recall: 0.9705
F1 Score: 0.9704
```

```
[ ]: from sklearn.preprocessing import label_binarize
     from sklearn.metrics import roc_curve, auc
```

```
[ ]: from sklearn.preprocessing import label_binarize

     if len(train_labels_val.shape) > 1 and train_labels_val.shape[1] > 1:
         train_labels_val = np.argmax(train_labels_val, axis=1)

     # Binarize the labels for multiclass ROC calculation
     label_binarizer = label_binarize(train_labels_val, classes=np.
      ↪unique(train_labels_val))

     # Calculate ROC curve for the Random Forest meta-learner
     fpr = dict()
     tpr = dict()
     roc_auc = dict()

     for i in range(label_binarizer.shape[1]):
         fpr[i], tpr[i], _ = roc_curve(label_binarizer[:, i], rf_meta_predictions[:,␣
      ↪i])
         roc_auc[i] = auc(fpr[i], tpr[i])

     # Plot ROC curve for each class
     plt.figure(figsize=(8, 6))

     for i in range(label_binarizer.shape[1]):
         plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')
     plt.plot([0, 1], [0, 1], 'k--')  # Diagonal reference line
     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.title('ROC Curve for Each Class')
     plt.legend(loc="lower right")
     plt.show()
```



[ ]:
```
# Binarize the labels for multiclass ROC calculation
label_binarizer = label_binarize(train_labels_val, classes=np.
 ↪unique(train_labels_val))

# Ensure the number of samples match
if label_binarizer.shape[0] != log_reg_predictions.shape[0]:
    raise ValueError("Inconsistent number of samples between label_binarizer␣
 ↪and log_reg_predictions.")

# Calculate ROC curve for the Logistic Regression meta-learner
fpr, tpr, _ = roc_curve(label_binarizer.ravel(), log_reg_predictions.ravel())
roc_auc = auc(fpr, tpr)
```

```python
# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal reference line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression')
plt.legend(loc="lower right")
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-111-61b1966bb40c> in <cell line: 9>()
      7
      8 # Calculate ROC curve for the Logistic Regression meta-learner
----> 9 fpr, tpr, _ = roc_curve(label_binarizer.ravel(), log_reg_predictions.
 ↪ravel())
     10 roc_auc = auc(fpr, tpr)
     11

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_ranking.py in␣
 ↪roc_curve(y_true, y_score, pos_label, sample_weight, drop_intermediate)
    990     array([1.8 , 0.8 , 0.4 , 0.35, 0.1 ])
    991     """
--> 992     fps, tps, thresholds = _binary_clf_curve(

    993         y_true, y_score, pos_label=pos_label, sample_weight=sample_weig t
    994     )

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_ranking.py in␣
 ↪_binary_clf_curve(y_true, y_score, pos_label, sample_weight)
    749         raise ValueError("{0} format is not supported".format(y_type))
    750
--> 751     check_consistent_length(y_true, y_score, sample_weight)
    752     y_true = column_or_1d(y_true)
    753     y_score = column_or_1d(y_score)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in␣
 ↪check_consistent_length(*arrays)
    395     uniques = np.unique(lengths)
    396     if len(uniques) > 1:
--> 397         raise ValueError(

    398             "Found input variables with inconsistent numbers of samples ␣
 ↪%r"
    399             % [int(l) for l in lengths]
```

```
ValueError: Found input variables with inconsistent numbers of samples: [35200,
↪7040]
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:
```
!pip install joblib
```

```
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(1.3.2)
```

[ ]:
```
from joblib import dump

# Assuming rf_meta_learner is your trained Random Forest meta-learner
model_filename = '/content/drive/MyDrive/Bully/rf_meta_learner.joblib'

# Save the trained model to a file
dump(rf_meta_learner, model_filename)
```

[ ]:
```
#from sklearn.preprocessing import label_binarize
#
#if len(train_labels_val.shape) > 1 and train_labels_val.shape[1] > 1:
#    train_labels_val = np.argmax(train_labels_val, axis=1)
#
## Binarize the labels for multiclass ROC calculation
#label_binarizer = label_binarize(train_labels_val, classes=np.
 ↪unique(train_labels_val))
#
## Calculate ROC curve for the Random Forest meta-learner
#fpr = dict()
#tpr = dict()
#roc_auc = dict()
#
#for i in range(label_binarizer.shape[1]):
#    fpr[i], tpr[i], _ = roc_curve(label_binarizer[:, i], rf_meta_predictions[:
 ↪, i])
#    roc_auc[i] = auc(fpr[i], tpr[i])
#
```

```
## Plot ROC curve for each class
#plt.figure(figsize=(8, 6))
#
#for i in range(label_binarizer.shape[1]):
#    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')
#
#plt.plot([0, 1], [0, 1], 'k--')  # Diagonal reference line
#plt.xlabel('False Positive Rate')
#plt.ylabel('True Positive Rate')
#plt.title('ROC Curve for Each Class')
#plt.legend(loc="lower right")
#plt.show()
```

```
[ ]: from sklearn.preprocessing import LabelEncoder

     if len(train_labels_val.shape) > 1 and train_labels_val.shape[1] > 1:
         train_labels_val = np.argmax(train_labels_val, axis=1)

     # Use LabelEncoder to ensure train_labels_val is represented as integers
     label_encoder = LabelEncoder()
     train_labels_val_encoded = label_encoder.fit_transform(train_labels_val)

     # Convert rf_meta_predictions to integer classes
     rf_meta_predictions_int = np.argmax(rf_meta_predictions, axis=1)

     # Ensure shapes align
     print("Shapes - True Labels:", train_labels_val_encoded.shape, "Predictions:",␣
      ↪rf_meta_predictions_int.shape)

     # Then try calculating confusion matrix and classification report
     conf_matrix = confusion_matrix(train_labels_val_encoded,␣
      ↪rf_meta_predictions_int)
     class_report = classification_report(train_labels_val_encoded,␣
      ↪rf_meta_predictions_int)
     print("Confusion Matrix:\n", conf_matrix)
     print("Classification Report:\n", class_report)
```

```
[ ]: # Plotting the confusion matrix
     plt.figure(figsize=(8, 6))
     sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d', cbar=False)
     plt.xlabel('Predicted Labels')
     plt.ylabel('True Labels')
     plt.title('Confusion Matrix')
     plt.show()
```

Lime

```
[ ]: !pip install shap lime
```

```
Collecting shap
  Downloading shap-0.44.0-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64
.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (533 kB)
                           533.5/533.5

kB 8.3 MB/s eta 0:00:00
Collecting lime
  Downloading lime-0.2.0.1.tar.gz (275 kB)
                           275.7/275.7

kB 35.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from shap) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from shap) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
packages (from shap) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from shap) (1.5.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-
packages (from shap) (4.66.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-
packages (from shap) (23.2)
Collecting slicer==0.0.7 (from shap)
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages
(from shap) (0.58.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-
packages (from shap) (2.2.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from lime) (3.7.1)
Requirement already satisfied: scikit-image>=0.12 in
/usr/local/lib/python3.10/dist-packages (from lime) (0.19.3)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-
packages (from scikit-image>=0.12->lime) (3.2.1)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (9.4.0)
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-image>=0.12->lime) (2.31.6)
Requirement already satisfied: tifffile>=2019.7.26 in
/usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime)
(2023.12.9)
Requirement already satisfied: PyWavelets>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (1.5.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
```

```
packages (from scikit-learn->shap) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (3.2.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->lime) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.4.5)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (2.8.2)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba->shap) (0.41.1)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->shap) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib->lime) (1.16.0)
Building wheels for collected packages: lime
  Building wheel for lime (setup.py) … done
  Created wheel for lime: filename=lime-0.2.0.1-py3-none-any.whl size=283835
sha256=57526409ff2e0b8024478bfe93d6d1aa7a8ad908d7ded4b4cbe985f1f18ac94f
  Stored in directory: /root/.cache/pip/wheels/fd/a2/af/9ac0a1a85a27f314a06b39e1
f492bee1547d52549a4606ed89
Successfully built lime
Installing collected packages: slicer, shap, lime
Successfully installed lime-0.2.0.1 shap-0.44.0 slicer-0.0.7
```

[ ]:
```python
import shap
from lime import lime_tabular
```

[ ]:
```
!free -h
```

```
               total        used        free      shared  buff/cache   available
Mem:            12Gi       3.9Gi       2.9Gi        21Mi       5.9Gi       8.5Gi
Swap:             0B          0B          0B
```

[ ]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer
```

[ ]:
```python
X = df['comment']
y = df['label']

# Split the data into training and validation sets
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_val_vec = vectorizer.transform(X_val)
```

[ ]:
```
# Get the unique topic names from the 'label' column
topic_names = df['label'].unique()
print("Unique topic names:", topic_names)
```

```
Unique topic names: [2 0 4 1 3]
```

[ ]:
```
!pip install textblob
```

```
Requirement already satisfied: textblob in /usr/local/lib/python3.10/dist-
packages (0.17.1)
Requirement already satisfied: nltk>=3.1 in /usr/local/lib/python3.10/dist-
packages (from textblob) (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages
(from nltk>=3.1->textblob) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from nltk>=3.1->textblob) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from nltk>=3.1->textblob) (4.66.1)
```

[ ]:
```
from textblob import TextBlob

def analyze_sentiment(text):
    # Create a TextBlob object
    blob = TextBlob(text)

    # Perform sentiment analysis
    sentiment_score = blob.sentiment.polarity

    return sentiment_score
```

[ ]:
```
!pip install gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.10/dist-packages
(4.3.2)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dist-
packages (from gensim) (1.23.5)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.10/dist-
packages (from gensim) (1.11.4)
```

```
Requirement already satisfied: smart-open>=1.8.1 in
/usr/local/lib/python3.10/dist-packages (from gensim) (6.4.0)
```

```
[ ]: from gensim import corpora, models
     import matplotlib.pyplot as plt

     # Assuming 'corpus' is a preprocessed list of text documents

     # Create a dictionary representation of the documents
     dictionary = corpora.Dictionary(corpus)

     # Convert the corpus to a document-term matrix
     doc_term_matrix = [dictionary.doc2bow(doc) for doc in corpus]

     # Train the LDA model
     lda_model = models.LdaModel(doc_term_matrix, num_topics=5, id2word=dictionary,␣
      ↪passes=15)

     # Assign labels/categories based on the identified topics
     topics = lda_model.show_topics(formatted=False)
     topic_labels = {0: 'not bully', 1: 'troll', 2: 'sexual', 3: 'religious', 4:␣
      ↪'threat'}  # Assign labels manually

     # Plotting the probabilities of categories
     for topic_id, topic in topics:
         topic_probabilities = [prob for _, prob in topic]
         plt.plot(topic_probabilities, label=topic_labels[topic_id])

     plt.xlabel('Word ID')
     plt.ylabel('Probability')
     plt.legend()
     plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-146-6f4b67661efe> in <cell line: 7>()
      5
      6 # Create a dictionary representation of the documents
----> 7 dictionary = corpora.Dictionary(corpus)
      8
      9 # Convert the corpus to a document-term matrix

/usr/local/lib/python3.10/dist-packages/gensim/corpora/dictionary.py in␣
 ↪__init__(self, documents, prune_at)
     76
     77         if documents is not None:
---> 78             self.add_documents(documents, prune_at=prune_at)
```

```
     79                 self.add_lifecycle_event(
     80                     "created",

/usr/local/lib/python3.10/dist-packages/gensim/corpora/dictionary.py in␣
 ↪add_documents(self, documents, prune_at)
    202
    203                 # update Dictionary with the document
--> 204                 self.doc2bow(document, allow_update=True)  # ignore the␣
 ↪result, here we only care about updating token ids
    205
    206             logger.info("built %s from %i documents (total %i corpus␣
 ↪positions)", self, self.num_docs, self.num_pos)

/usr/local/lib/python3.10/dist-packages/gensim/corpora/dictionary.py in␣
 ↪doc2bow(self, document, allow_update, return_missing)
    239             """
    240             if isinstance(document, str):
--> 241                 raise TypeError("doc2bow expects an array of unicode tokens␣
 ↪on input, not a single string")
    242
    243             # Construct (word, frequency) mapping.

TypeError: doc2bow expects an array of unicode tokens on input, not a single␣
 ↪string
```

[ ]:
```python
# Sample input text
input_text = "                                                                    ␣
 ↪                   "

# Tokenize and preprocess the input text
processed_input = dictionary.doc2bow(simple_preprocess(input_text))

# Get the topic distribution for the input text
topic_distribution = lda_model[processed_input]

# Display the extracted topics and their probabilities for the input text
print(topic_distribution)
```

```
[(0, 0.018206181), (1, 0.018206127), (2, 0.018206157), (3, 0.01820615), (4,
0.9271754)]
```

[ ]:
```python
```

[ ]:
```python
from transformers import BertTokenizer

# Load the Bangla BERT tokenizer
tokenizer = BertTokenizer.from_pretrained("sagorsarker/bangla-bert-base")
```

```
# Input text
input_text = "                                                   ,     ␣
  ↪                   "

# Tokenize the input text using the BERT tokenizer
tokenized_input = tokenizer.encode_plus(
    input_text,
    max_length=100,
    truncation=True,
    padding="max_length",
    return_tensors="np"
)

# Extract features
word_count = len(tokenized_input["input_ids"][0])
sentence_length = len(input_text.split('.'))
average_word_length = sum(len(word) for word in input_text.split()) / word_count

# Sentiment analysis (this requires a trained sentiment analysis model)
sentiment_score = analyze_sentiment(input_text)  # Placeholder function

# Assuming you need BERT token indices as features
bert_token_indices = tokenized_input["input_ids"][0]

# Display extracted features
print(f"Word count: {word_count}")
print(f"Sentence length: {sentence_length}")
print(f"Average word length: {average_word_length}")
print(f"Sentiment score: {sentiment_score}")
print(f"BERT token indices: {bert_token_indices}")
```

```
Word count: 100
Sentence length: 1
Average word length: 1.28
Sentiment score: 0.0
BERT token indices: [  101 10706  7448  4931 85209  9294  2844 25151 35705  5931
5740  9294
  9294 51264  6741 72612  5843  7932  2093   100  2285 45931  2097  1014
 16137 21327  2040  6741 33983 59464  8833  2237  7373  1011 16137 21327
  2040  4374  2076  5500  6399  3364  5740  9294  9294  2069  8705  9294
  2446 21327  5740  9294  9294  2058  2069  3447  5843  1014   102     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0]
```

```
[ ]: encoded_text = tokenizer.encode(input_text, max_length=100, truncation=True,␣
      ↪padding='max_length', return_tensors='pt')
     bert_tokens = encoded_text[0].tolist()  # Converting tensor to list
     print("BERT token indices:", bert_tokens)
```

```
BERT token indices: [101, 10706, 7448, 4931, 85209, 9294, 2844, 25151, 35705,
5931, 5740, 9294, 9294, 51264, 6741, 72612, 5843, 7932, 2093, 100, 2285, 45931,
2097, 1014, 16137, 21327, 2040, 6741, 33983, 59464, 8833, 2237, 7373, 1011,
16137, 21327, 2040, 4374, 2076, 5500, 6399, 3364, 5740, 9294, 9294, 2069, 8705,
9294, 2446, 21327, 5740, 9294, 9294, 2058, 2069, 3447, 5843, 1014, 102, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

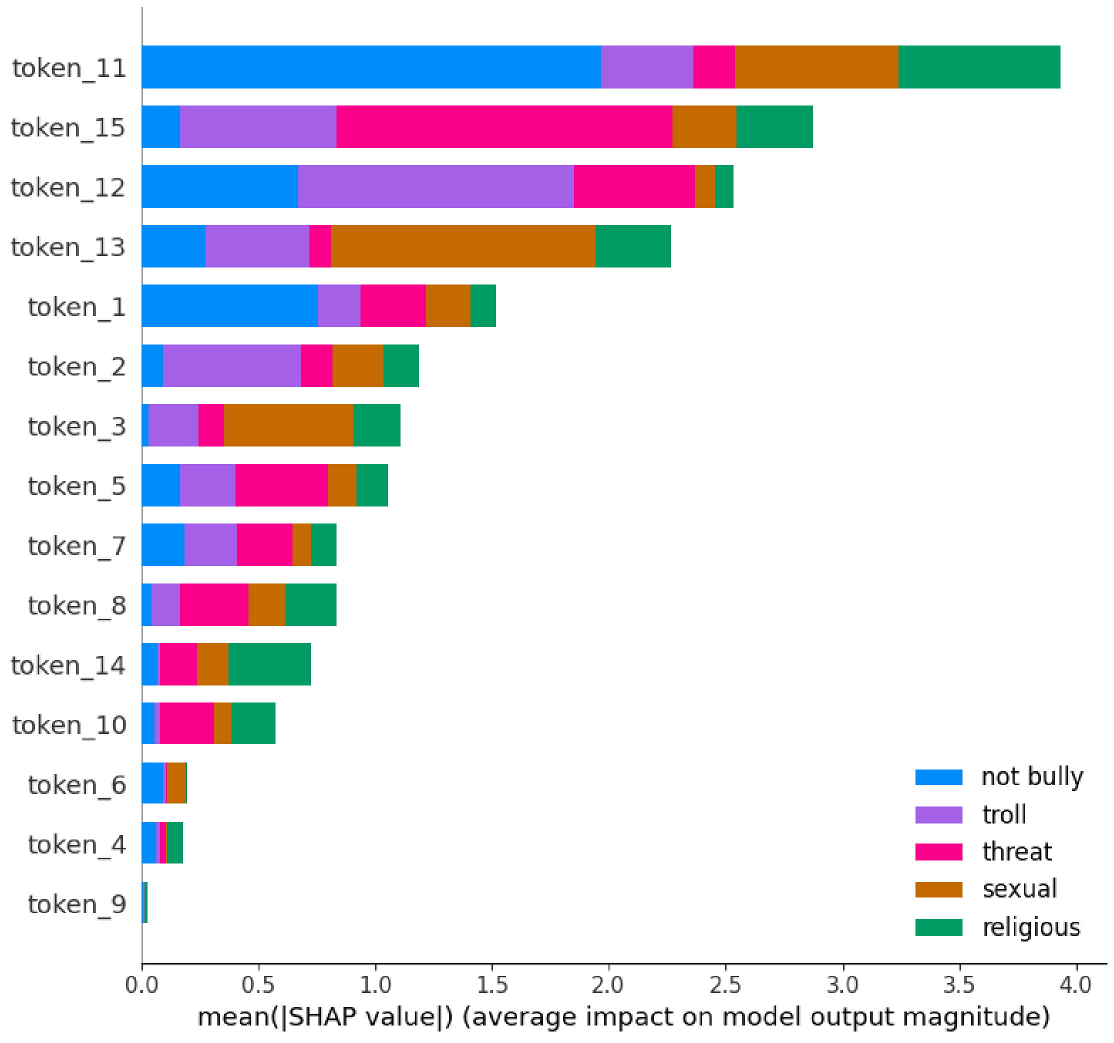
```
[ ]: # Define the feature names
     feature_names = []

     # BERT token indices
     for i in range(100):  # Assuming max_length=100 for BERT tokenizer
         feature_names.append(f"token_{i+1}")

     # Text statistics
     feature_names.extend([
         'word_count',
         'sentence_length',
         'average_word_length',
     ])
```

```
[ ]: # Sentiment analysis score
     feature_names.append('sentiment_score')

     # Model-specific features
     for model_name in ['BiGRU', 'BiLSTM', 'CNN']:
         for i in range(50):  # Assuming 50 intermediate outputs for each model
             feature_names.append(f"{model_name}_output_{i+1}")
```

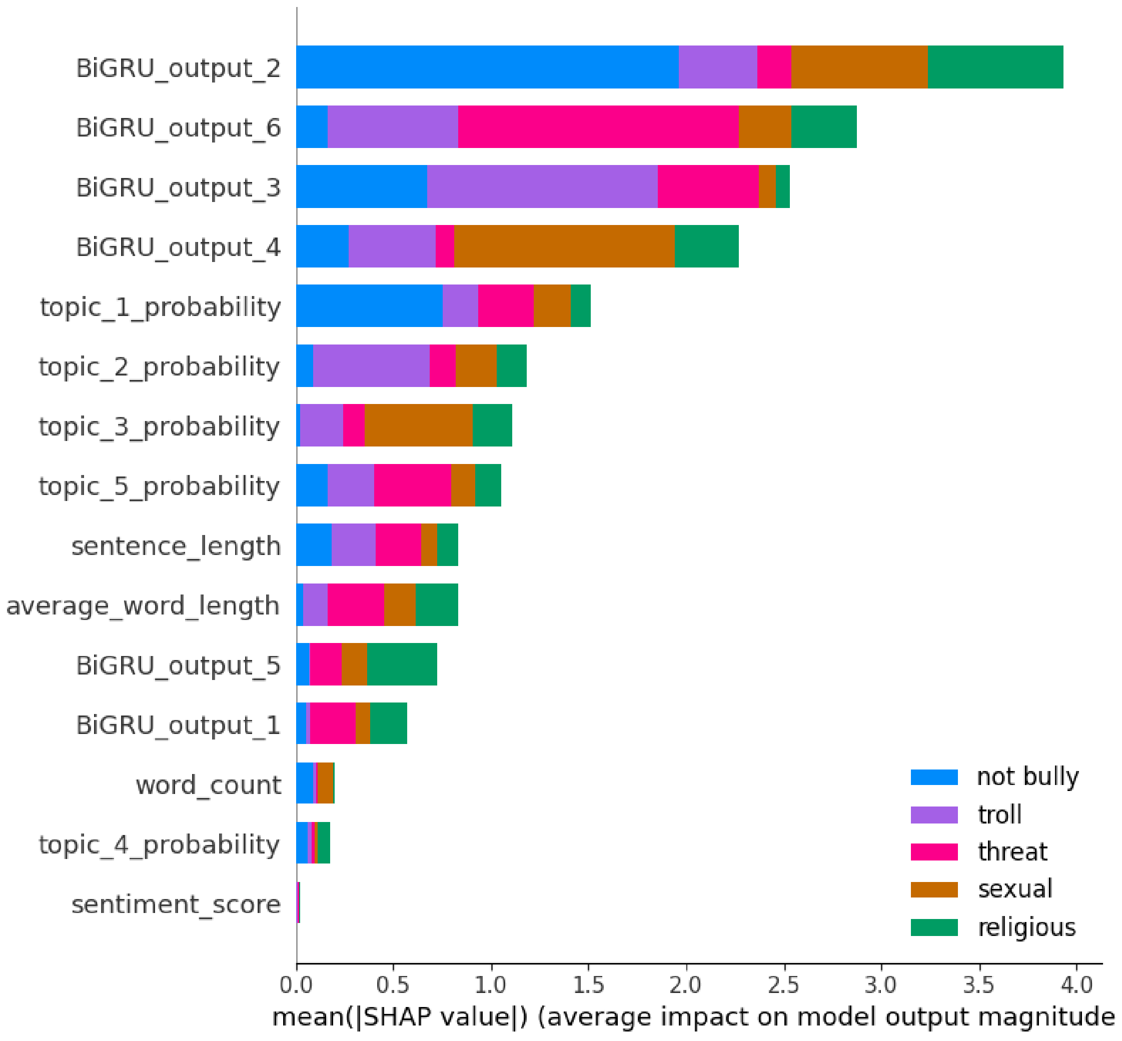
```
[ ]: # Meta-model features (logistic regression)
     for i in range(10):  # Assuming 10 features in the logistic regression␣
      ↪meta-model
         feature_names.append(f"log_reg_feature_{i+1}")

     # Generate random topic probabilities for demonstration
     num_samples = len(stacked_val)
     num_topics = 5
     topic_probabilities = np.random.rand(num_samples, num_topics)

     # Example feature names for topic probabilities
     for i in range(num_topics):
```

```
    feature_names.append(f"topic_{i+1}_probability")
```

```
[ ]: # Use SHAP for explanation
explainer = shap.Explainer(log_reg, stacked_train)
shap_values = explainer.shap_values(stacked_val)

# Assuming 'label_names' is a list of class names for classification
label_names = [
    'not bully',
    'troll',
    'sexual',
    'religious',
    'threat'
]

# Visualize SHAP summary plot
shap.summary_plot(shap_values, stacked_val, feature_names=feature_names,␣
 ↪class_names=label_names)
```



```
[ ]: # Using SHAP for explanation
     explainer = shap.Explainer(log_reg, stacked_train)  # log_reg is your logistic␣
      ↪regression model
     shap_values = explainer.shap_values(stacked_val)

     # Visualize SHAP summary plot
     shap.summary_plot(shap_values, stacked_val, feature_names=feature_names,␣
      ↪class_names=label_names)
```



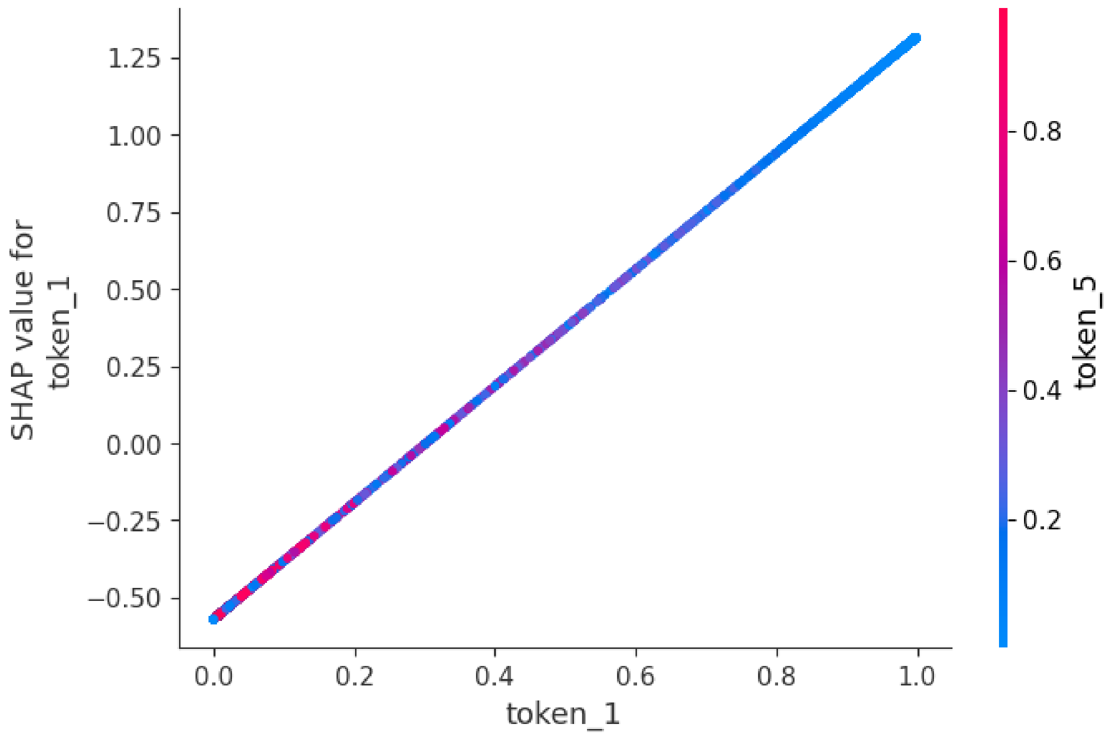
```
[ ]: print(stacked_val.shape)
```

```
(7040, 15)
```

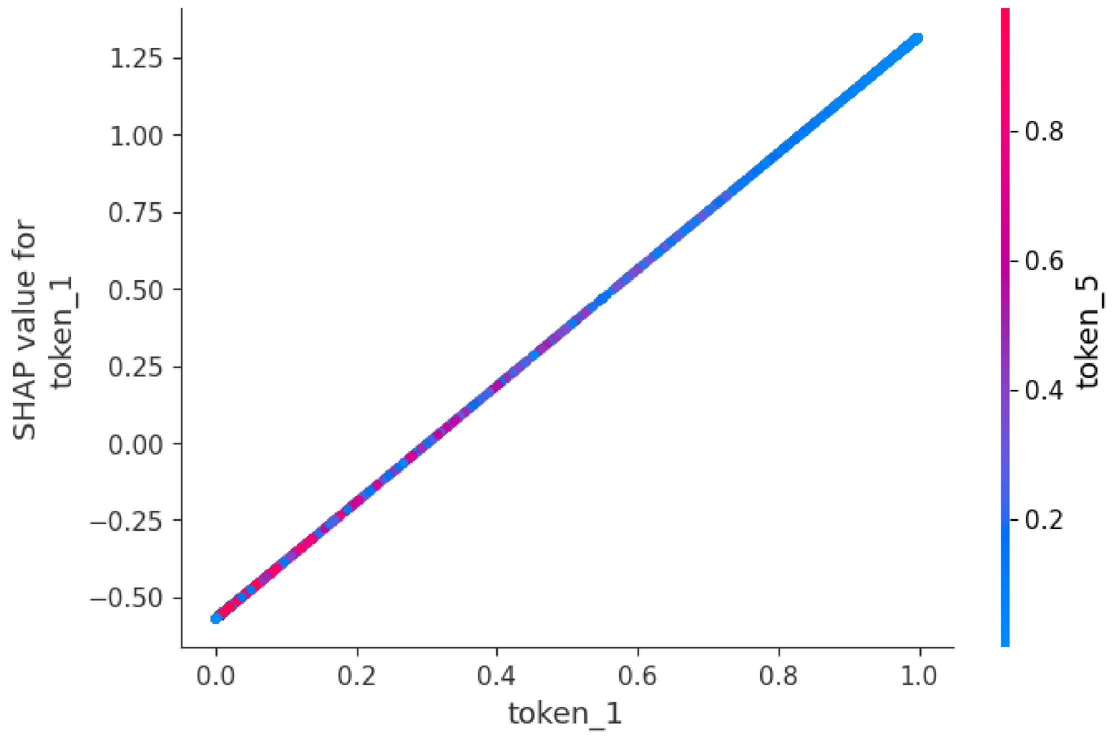
```
[ ]: print(len(feature_names))
```

```
269
```

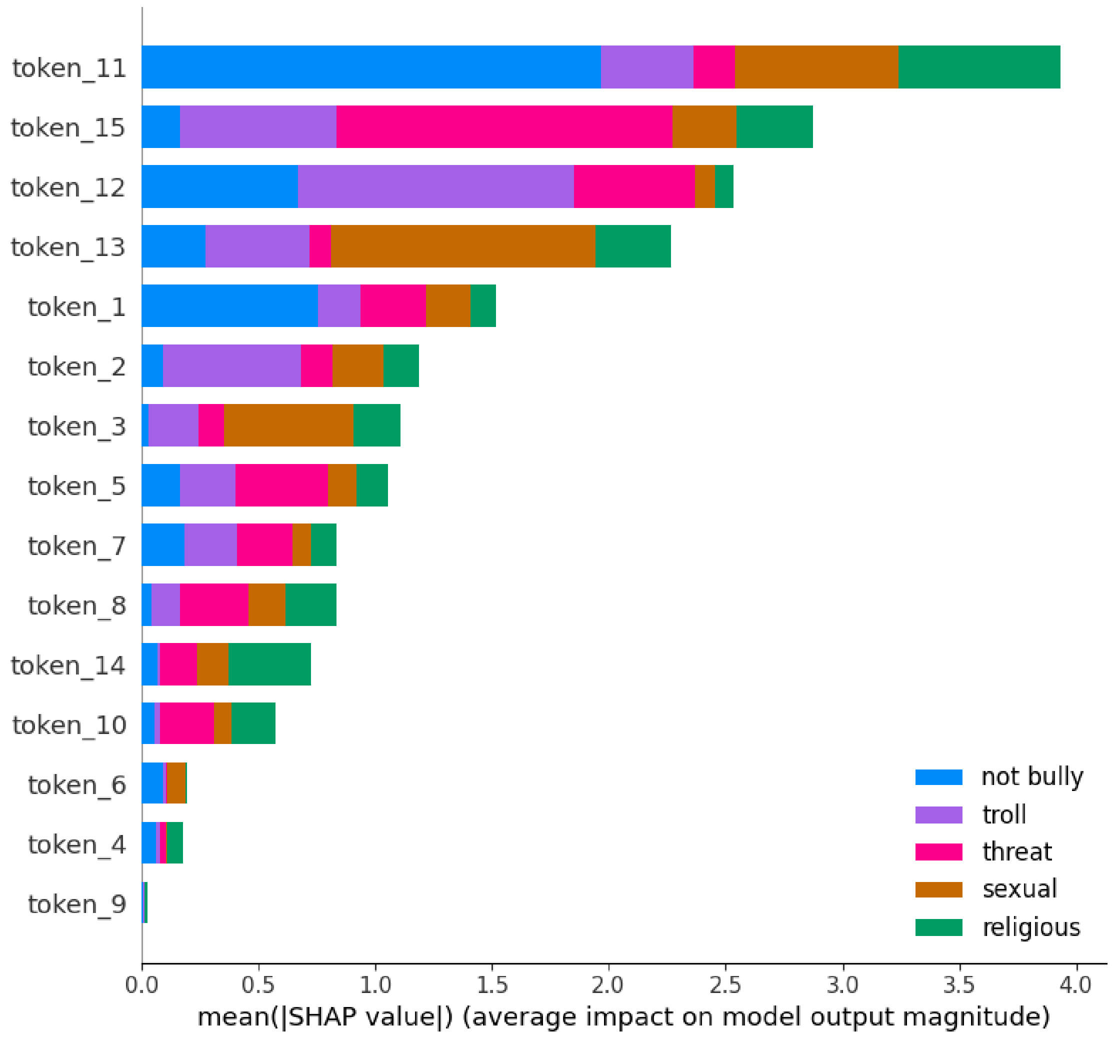
```
[ ]: # Replace '0' with the desired feature index
     shap.dependence_plot(0, shap_values[0], stacked_val,␣
      ↪feature_names=feature_names)
```



```
[ ]: # Replace '0' with the desired feature index
     shap.dependence_plot(0, shap_values[0], stacked_val,␣
      ↪feature_names=feature_names, interaction_index='auto')
```



```
[ ]: # Replace 'label_names' with your actual label names
     label_names = [
         'not bully',
         'troll',
         'sexual',
         'religious',
         'threat'
     ]

     # For summary plot
     shap.summary_plot(shap_values, stacked_val, feature_names=feature_names,␣
      ↪class_names=label_names)
```



```
[ ]: print("Size of SHAP values:", len(shap_values))
     print("Size of feature names:", len(feature_names))
```

```
Size of SHAP values: 5
Size of feature names: 269
```

```
[ ]: class_index = 0  # Replace this with the desired class index (0 to 14)
     shap.force_plot(explainer.expected_value[class_index],␣
      ↪shap_values[class_index][0], stacked_val[0], feature_names=feature_names)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-171-100139b251ae> in <cell line: 2>()
      1 class_index = 0  # Replace this with the desired class index (0 to 14)
```

```
----> 2 shap.force_plot(explainer.expected_value[class_index],␣
 ↪shap_values[class_index][0], stacked_val[0], feature_names=feature_names)
      3

/usr/local/lib/python3.10/dist-packages/shap/plots/_force.py in␣
 ↪force(base_value, shap_values, features, feature_names, out_names, link,␣
 ↪plot_cmap, matplotlib, show, figsize, ordering_keys,␣
 ↪ordering_keys_time_format, text_rotation, contribution_threshold)
    194         )
    195
--> 196         return visualize(e,

    197                          plot_cmap,
    198                          matplotlib,

/usr/local/lib/python3.10/dist-packages/shap/plots/_force.py in visualize(e,␣
 ↪plot_cmap, matplotlib, figsize, show, ordering_keys,␣
 ↪ordering_keys_time_format, text_rotation, min_perc)
    412             )
    413         else:
--> 414             return AdditiveForceVisualizer(e, plot_cmap=plot_cmap)
    415     elif isinstance(e, Explanation):
    416         if matplotlib:

/usr/local/lib/python3.10/dist-packages/shap/plots/_force.py in __init__(self,␣
 ↪e, plot_cmap)
    490         # build the json data
    491         features = {}
--> 492         for i in filter(lambda j: e.effects[j] != 0, range(len(e.data.
 ↪group_names))):
    493             features[i] = {
    494                 "effect": ensure_not_numpy(e.effects[i]),

/usr/local/lib/python3.10/dist-packages/shap/plots/_force.py in <lambda>(j)
    490         # build the json data
    491         features = {}
--> 492         for i in filter(lambda j: e.effects[j] != 0, range(len(e.data.
 ↪group_names))):
    493             features[i] = {
    494                 "effect": ensure_not_numpy(e.effects[i]),

IndexError: index 15 is out of bounds for axis 0 with size 15
```

[ ]:

```
```

[ ]:

```
num_samples = 1000
num_topics = 5
```

```
topic_probabilities = np.random.rand(num_samples, num_topics)

# Example feature names for topic probabilities
feature_names = []
for i in range(num_topics):
    feature_names.append(f"topic_{i+1}_probability")
```

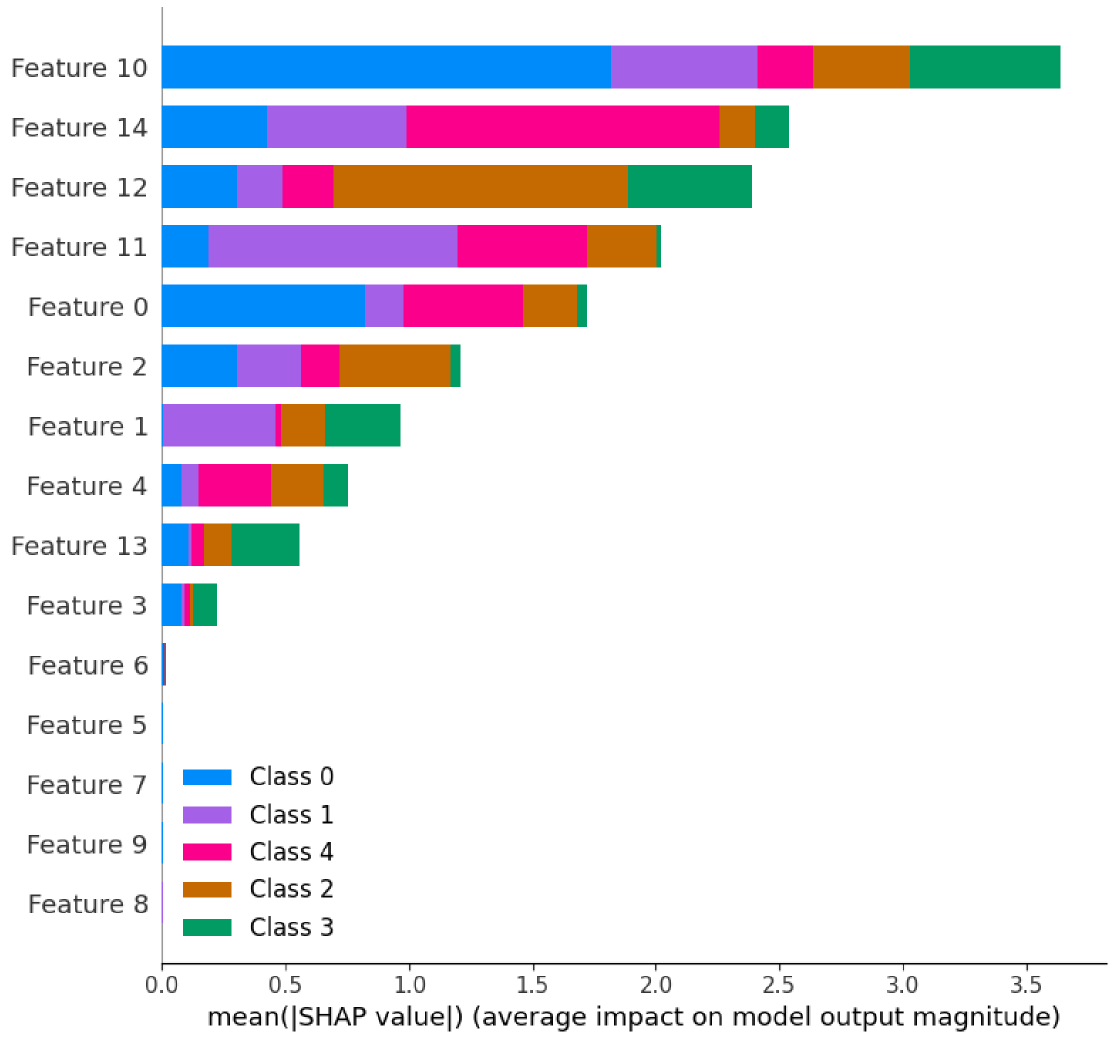
[ ]:
```
# Create a SHAP explainer object for the logistic regression model
explainer = shap.Explainer(log_reg, stacked_train)

# Calculate SHAP values for the validation data
shap_values = explainer.shap_values(stacked_val)

# Visualize SHAP summary plot
shap.summary_plot(shap_values, stacked_val, feature_names=feature_names)
```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```
# Create a SHAP explainer object for the logistic regression model
explainer = shap.Explainer(log_reg, stacked_train)

# Calculate SHAP values for the validation data
shap_values = explainer.shap_values(stacked_val)

# Visualize SHAP summary plot
shap.summary_plot(shap_values, stacked_val)
```



[ ]:
```
num_features = stacked_val.shape[1]
print("Number of features in stacked_val:", num_features)
print("Number of elements in label_names:", len(label_names))
```

```
Number of features in stacked_val: 15
Number of elements in label_names: 5
```

[ ]:
```
def predict_log_reg_proba(texts):
    # Assuming you have a tokenizer for text data
    encodings = tokenizer(texts, truncation=True, padding='max_length',␣
↪max_length=100, return_tensors='tf')['input_ids']
    inputs = {'sequences': encodings}

    # Use your 'log_reg' Logistic Regression model to predict probabilities
    predictions = log_reg.predict_proba(inputs)  # Replace 'inputs' with the␣
↪appropriately preprocessed data

    return predictions
```

```
[ ]: feature_names = [
         'not bully',
         'troll',
         'sexual',
         'religious',
         'threat'
     ]
```

```
[ ]: from lime.lime_tabular import LimeTabularExplainer

     # Assuming stacked_val represents your validation data for the Logistic␣
     ↪Regression model
     # Replace 'feature_names' with the actual names of the features used in the␣
     ↪Logistic Regression model
     # 'train_labels_val' should be the labels corresponding to 'stacked_val'
     explainer = LimeTabularExplainer(np.array(stacked_val),
                                      mode='classification',
                                      training_labels=train_labels_val,
                                      feature_names=feature_names,  # Replace with␣
     ↪actual feature names
                                      discretize_continuous=True)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-123-9477b49d77f7> in <cell line: 6>()
      4 # Replace 'feature_names' with the actual names of the features used in␣
↪the Logistic Regression model
      5 # 'train_labels_val' should be the labels corresponding to 'stacked_val
----> 6 explainer = LimeTabularExplainer(np.array(stacked_val),

      7                                  mode='classification',
      8                                  training_labels=train_labels_val,

/usr/local/lib/python3.10/dist-packages/lime/lime_tabular.py in __init__(self,␣
↪training_data, mode, training_labels, feature_names, categorical_features,␣
↪categorical_names, kernel_width, kernel, verbose, class_names,␣
↪feature_selection, discretize_continuous, discretizer, sample_around_instance ␣
↪random_state, training_data_stats)
    213
    214             if discretizer == 'quartile':
--> 215                 self.discretizer = QuartileDiscretizer(

    216                         training_data, self.categorical_features,
    217                         self.feature_names, labels=training_labels,
/usr/local/lib/python3.10/dist-packages/lime/discretize.py in __init__(self,␣
↪data, categorical_features, feature_names, labels, random_state)
    176     def __init__(self, data, categorical_features, feature_names,␣
↪labels=None, random_state=None):
    177
--> 178         BaseDiscretizer.__init__(self, data, categorical_features,

    179                                  feature_names, labels=labels,
    180                                  random_state=random_state)

/usr/local/lib/python3.10/dist-packages/lime/discretize.py in __init__(self,␣
↪data, categorical_features, feature_names, labels, random_state, data_stats)
     62             n_bins = qts.shape[0]  # Actually number of borders (=␣
↪#bins-1)
     63             boundaries = np.min(data[:, feature]), np.max(data[:,␣
↪feature])
---> 64             name = feature_names[feature]
     65
     66             self.names[feature] = ['%s <= %.2f' % (name, qts[0])]

IndexError: list index out of range
```

[ ]:

```

```

[ ]:

```

```

[ ]:

```
# Initialize KernelExplainer
kernel_explainer = shap.KernelExplainer(rf_meta_learner.predict, stacked_train)
```

```
WARNING:shap:Using 28158 background data samples could cause slower run times.
Consider using shap.sample(data, K) or shap.kmeans(data, K) to summarize the
background as K samples.
```

[ ]:

```
import shap

# Initialize KernelExplainer
kernel_explainer = shap.KernelExplainer(rf_meta_learner.predict, stacked_train)

# Sample a subset for SHAP visualization (adjust the subset size based on your␣
↪resources)
subset_size_shap = 20
subset_indices_shap = np.random.choice(len(test_texts), size=subset_size_shap,␣
↪replace=False)
subset_test_texts = [test_texts[i] for i in subset_indices_shap]
subset_test_input_ids = tokenizer(subset_test_texts, truncation=True,␣
↪padding=True, max_length=max_length, return_tensors="tf")
```

```
subset_stacked_val_embeddings = models['GRU'].
↪predict(subset_test_input_ids['input_ids'].numpy())
```

```
WARNING:shap:Using 28158 background data samples could cause slower run times.
Consider using shap.sample(data, K) or shap.kmeans(data, K) to summarize the
background as K samples.

1/1 [==============================] - 2s 2s/step
```

[ ]:
```
import shap
import numpy as np

# Sample a subset for SHAP visualization (adjust the subset size based on your␣
↪resources)
subset_size_shap = 20
subset_indices_shap = np.random.choice(len(test_texts), size=subset_size_shap,␣
↪replace=False)
subset_test_texts = [test_texts[i] for i in subset_indices_shap]
subset_test_input_ids = tokenizer(subset_test_texts, truncation=True,␣
↪padding=True, max_length=max_length, return_tensors="tf")
subset_stacked_val_embeddings = models['GRU'].
↪predict(subset_test_input_ids['input_ids'].numpy())

# Initialize KernelExplainer with the RandomForest model
kernel_explainer = shap.KernelExplainer(rf_meta_learner.predict, shap.
↪sample(stacked_train, 100))

# Calculate SHAP values using KernelExplainer
shap_values_kernel = kernel_explainer.shap_values(subset_stacked_val_embeddings)

# Calculate SHAP values using KernelExplainer
shap_values_kernel = kernel_explainer.shap_values(subset_stacked_val_embeddings)

# Summary plot for the first instance
shap.summary_plot(shap_values_kernel, subset_stacked_val_embeddings,␣
↪feature_names=models.keys())
```

```
1/1 [==============================] - 0s 37ms/step

  0%|          | 0/20 [00:00<?, ?it/s]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-103-98bbc0632a55> in <cell line: 15>()
     13
     14 # Calculate SHAP values using KernelExplainer
---> 15 shap_values_kernel = kernel_explainer.
 ↪shap_values(subset_stacked_val_embeddings)
```

```
     16
     17 # Calculate SHAP values using KernelExplainer

/usr/local/lib/python3.10/dist-packages/shap/explainers/_kernel.py in␣
 ↪shap_values(self, X, **kwargs)
    242                     if self.keep_index:
    243                         data = convert_to_instance_with_index(data,␣
 ↪column_name, index_value[i:i + 1], index_name)
--> 244                     explanations.append(self.explain(data, **kwargs))
    245                     if kwargs.get("gc_collect", False):
    246                         gc.collect()

/usr/local/lib/python3.10/dist-packages/shap/explainers/_kernel.py in␣
 ↪explain(self, incoming_instance, **kwargs)
    269         # convert incoming input to a standardized iml object
    270         instance = convert_to_instance(incoming_instance)
--> 271         match_instance_to_data(instance, self.data)
    272
    273         # find the feature groups we will test. If a feature does not␣
 ↪change from its

/usr/local/lib/python3.10/dist-packages/shap/utils/_legacy.py in␣
 ↪match_instance_to_data(instance, data)
     88     if isinstance(data, DenseData):
     89         if instance.group_display_values is None:
---> 90             instance.group_display_values = [instance.x[0, group[0]] if␣
 ↪len(group) == 1 else "" for group in data.groups]
     91         assert len(instance.group_display_values) == len(data.groups)
     92         instance.groups = data.groups

/usr/local/lib/python3.10/dist-packages/shap/utils/_legacy.py in <listcomp>(.0)
     88     if isinstance(data, DenseData):
     89         if instance.group_display_values is None:
---> 90             instance.group_display_values = [instance.x[0, group[0]] if␣
 ↪len(group) == 1 else "" for group in data.groups]
     91         assert len(instance.group_display_values) == len(data.groups)
     92         instance.groups = data.groups

IndexError: index 5 is out of bounds for axis 1 with size 5
```

```
[ ]: print("Shape of subset_stacked_val_embeddings:", subset_stacked_val_embeddings.
      ↪shape)
     print("Shape of background dataset (stacked_train):", stacked_train.shape)
     shap_values_kernel = kernel_explainer.shap_values(subset_stacked_val_embeddings)
```

```
Shape of subset_stacked_val_embeddings: (20, 5)
Shape of background dataset (stacked_train): (28158, 15)
```

```
0%|          | 0/20 [00:00<?, ?it/s]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-91-71190363c975> in <cell line: 3>()
      1 print("Shape of subset_stacked_val_embeddings:",␣
 ↪subset_stacked_val_embeddings.shape)
      2 print("Shape of background dataset (stacked_train):", stacked_train.
 ↪shape)
----> 3 shap_values_kernel = kernel_explainer.
 ↪shap_values(subset_stacked_val_embeddings)

/usr/local/lib/python3.10/dist-packages/shap/explainers/_kernel.py in␣
 ↪shap_values(self, X, **kwargs)
    242                 if self.keep_index:
    243                     data = convert_to_instance_with_index(data,␣
 ↪column_name, index_value[i:i + 1], index_name)
--> 244                 explanations.append(self.explain(data, **kwargs))
    245                 if kwargs.get("gc_collect", False):
    246                     gc.collect()

/usr/local/lib/python3.10/dist-packages/shap/explainers/_kernel.py in␣
 ↪explain(self, incoming_instance, **kwargs)
    269         # convert incoming input to a standardized iml object
    270         instance = convert_to_instance(incoming_instance)
--> 271         match_instance_to_data(instance, self.data)
    272 
    273         # find the feature groups we will test. If a feature does not␣
 ↪change from its

/usr/local/lib/python3.10/dist-packages/shap/utils/_legacy.py in␣
 ↪match_instance_to_data(instance, data)
     88     if isinstance(data, DenseData):
     89         if instance.group_display_values is None:
---> 90             instance.group_display_values = [instance.x[0, group[0]] if␣
 ↪len(group) == 1 else "" for group in data.groups]
     91         assert len(instance.group_display_values) == len(data.groups)
     92         instance.groups = data.groups

/usr/local/lib/python3.10/dist-packages/shap/utils/_legacy.py in <listcomp>(.0)
     88     if isinstance(data, DenseData):
     89         if instance.group_display_values is None:
---> 90             instance.group_display_values = [instance.x[0, group[0]] if␣
 ↪len(group) == 1 else "" for group in data.groups]
     91         assert len(instance.group_display_values) == len(data.groups)
     92         instance.groups = data.groups
```

```
IndexError: index 5 is out of bounds for axis 1 with size 5
```

[ ]:
```
# Print intermediate shapes and values for debugging
print("Shape of BERT embeddings (before):", subset_stacked_val_embeddings.shape)

# Check the structure of the obtained embeddings
print("First instance of BERT embeddings:", subset_stacked_val_embeddings[0])
```

```
Shape of BERT embeddings (before): (10, 5)
First instance of BERT embeddings: [0.00691598 0.01117767 0.88472986 0.0036564
0.09352009]
```

[ ]:
```
# Check the shape of the model training data
print("Training Data Shape:", stacked_train.shape)

# Check the shape of the input data for the explainer
print("Explainer Input Shape:", subset_stacked_val_embeddings.shape)
```

```
Training Data Shape: (28158, 15)
Explainer Input Shape: (100, 5)
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:
```
gru_model = tf.keras.models.load_model("/content/drive/MyDrive/Cyberbullying␣
  ↪Detection/GRU_Model")
```

[ ]:
```
cnn_model = tf.keras.models.load_model("/content/drive/MyDrive/Cyberbullying␣
  ↪Detection/CNN_Model")
lstm_model = tf.keras.models.load_model("/content/drive/MyDrive/Cyberbullying␣
  ↪Detection/LSTM_Model")
ensemble_model = tf.keras.models.load_model("/content/drive/MyDrive/
  ↪Cyberbullying Detection/Stacking_Model.h5")
```

[ ]:
```
from lime import lime_text
from lime.lime_text import LimeTextExplainer
from keras.preprocessing.sequence import pad_sequences
```

[ ]:
```
def predict_proba(texts):
    encodings = tokenizer(texts, truncation=True, padding='max_length',␣
  ↪max_length=100, return_tensors='tf')['input_ids']
    inputs = {'sequences': encodings}
    predictions = gru_model.predict(inputs)
    return predictions
```

```
[ ]: label_names = [
         'not bully',
         'troll',
         'sexual',
         'religious',
         'threat'
     ]
```

```
[ ]: explainer = LimeTextExplainer(class_names = label_names)
```

```
[ ]: for i, sample in enumerate(test_dataset.take(20)):
         input_data, sample_text = sample
         sample_text = sample_text.numpy().decode('utf-8')
         explanation = explainer.explain_instance(sample_text, predict_proba,␣
      ↪num_features=10)
         # If you want to see the explanation for each instance, you can visualize␣
      ↪it here.
         print(f"Explanation for sample {i+1}:")
         explanation.show_in_notebook()
```

```
157/157 [==============================] - 3s 17ms/step
Explanation for sample 1:

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 14ms/step
Explanation for sample 2:

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 15ms/step
Explanation for sample 3:

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 15ms/step
Explanation for sample 4:

<IPython.core.display.HTML object>

157/157 [==============================] - 3s 19ms/step
Explanation for sample 5:

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 15ms/step
Explanation for sample 6:

<IPython.core.display.HTML object>
```

```
157/157 [==============================] - 2s 15ms/step
Explanation for sample 7:

<IPython.core.display.HTML object>

157/157 [==============================] - 4s 23ms/step
Explanation for sample 8:

<IPython.core.display.HTML object>

157/157 [==============================] - 3s 18ms/step
Explanation for sample 9:

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 15ms/step
Explanation for sample 10:

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 15ms/step
Explanation for sample 11:

<IPython.core.display.HTML object>

157/157 [==============================] - 3s 17ms/step
Explanation for sample 12:

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 15ms/step
Explanation for sample 13:

<IPython.core.display.HTML object>

157/157 [==============================] - 3s 19ms/step
Explanation for sample 14:

<IPython.core.display.HTML object>

157/157 [==============================] - 4s 24ms/step
Explanation for sample 15:

<IPython.core.display.HTML object>

157/157 [==============================] - 3s 20ms/step
Explanation for sample 16:

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 15ms/step
Explanation for sample 17:

<IPython.core.display.HTML object>

157/157 [==============================] - 3s 19ms/step
Explanation for sample 18:

<IPython.core.display.HTML object>
```

```
157/157 [==============================] - 2s 15ms/step
Explanation for sample 19:

<IPython.core.display.HTML object>

157/157 [==============================] - 3s 16ms/step
Explanation for sample 20:

<IPython.core.display.HTML object>
```

```
[ ]: def predict_proba(texts):
         encodings = tokenizer(texts, truncation=True, padding='max_length',␣
      ↪max_length=100, return_tensors='tf')['input_ids']
         inputs = {'sequences': encodings}
         predictions = cnn_model.predict(inputs)
         return predictions
```

```
[ ]: for i, sample in enumerate(test_dataset.take(5)):
         input_data, sample_text = sample
         sample_text = sample_text.numpy().decode('utf-8')
         explanation = explainer.explain_instance(sample_text, predict_proba,␣
      ↪num_features=10)
         # If you want to see the explanation for each instance, you can visualize␣
      ↪it here.
         print(f"Explanation for sample {i+1}:")
         explanation.show_in_notebook()
```

```
157/157 [==============================] - 4s 3ms/step
Explanation for sample 1:

<IPython.core.display.HTML object>

157/157 [==============================] - 1s 3ms/step
Explanation for sample 2:

<IPython.core.display.HTML object>

157/157 [==============================] - 1s 3ms/step
Explanation for sample 3:

<IPython.core.display.HTML object>

157/157 [==============================] - 1s 4ms/step
Explanation for sample 4:

<IPython.core.display.HTML object>

157/157 [==============================] - 0s 3ms/step
Explanation for sample 5:

<IPython.core.display.HTML object>
```

```
[ ]: def predict_proba(texts):
         encodings = tokenizer(texts, truncation=True, padding='max_length',␣
      ↪max_length=100, return_tensors='tf')['input_ids']
         inputs = {'sequences': encodings}
         predictions = lstm_model.predict(inputs)
         return predictions
```

```
[ ]: for i, sample in enumerate(test_dataset.take(5)):
         input_data, sample_text = sample
         sample_text = sample_text.numpy().decode('utf-8')
         explanation = explainer.explain_instance(sample_text, predict_proba,␣
      ↪num_features=10)
         # If you want to see the explanation for each instance, you can visualize␣
      ↪it here.
         print(f"Explanation for sample {i+1}:")
         explanation.show_in_notebook()
```

```
[ ]: def predict_proba(texts):
         encodings = tokenizer(texts, truncation=True, padding='max_length',␣
      ↪max_length=100, return_tensors='tf')['input_ids']
         inputs = {'sequences': encodings}
         predictions = ensemble_model.predict(inputs)
         return predictions
```

```
[ ]: for i, sample in enumerate(test_dataset.take(5)):
         input_data, sample_text = sample
         sample_text = sample_text.numpy().decode('utf-8')
         explanation = explainer.explain_instance(sample_text, predict_proba,␣
      ↪num_features=10)
         # If you want to see the explanation for each instance, you can visualize␣
      ↪it here.
         print(f"Explanation for sample {i+1}:")
         explanation.show_in_notebook()
```

```
157/157 [==============================] - 1s 3ms/step
Explanation for sample 1:

<IPython.core.display.HTML object>

157/157 [==============================] - 1s 5ms/step
Explanation for sample 2:

<IPython.core.display.HTML object>

157/157 [==============================] - 1s 3ms/step
Explanation for sample 3:

<IPython.core.display.HTML object>
```

```
157/157 [==============================] - 1s 3ms/step
Explanation for sample 4:

<IPython.core.display.HTML object>

157/157 [==============================] - 1s 3ms/step
Explanation for sample 5:

<IPython.core.display.HTML object>
```

Integrated Gradients

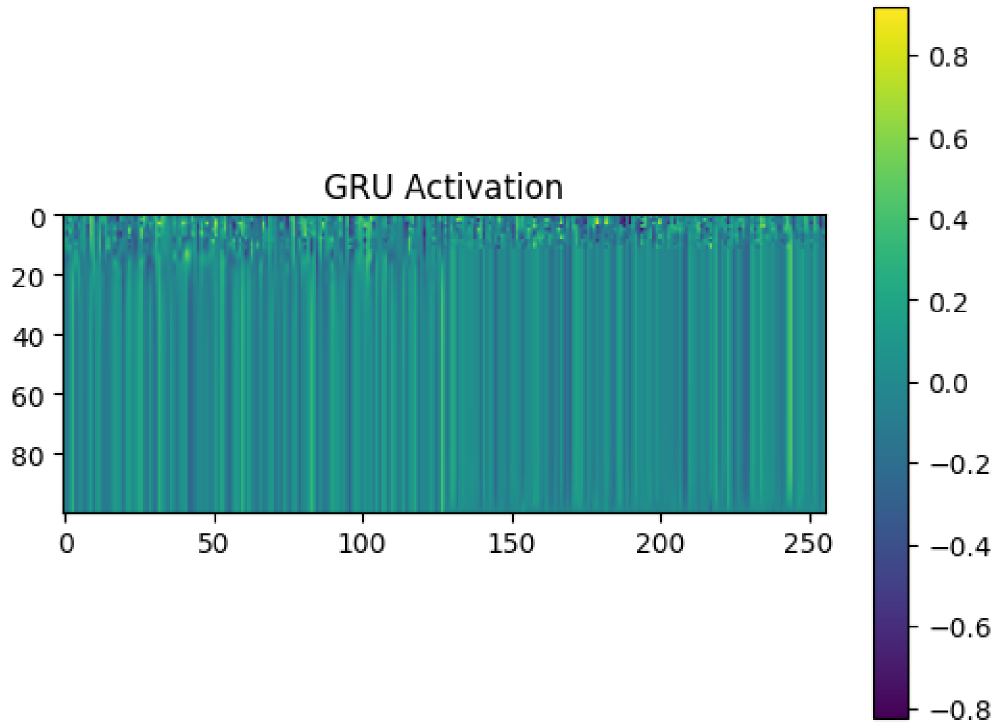
```
[ ]: !pip install shap
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.10/dist-packages
(0.44.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from shap) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from shap) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
packages (from shap) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from shap) (1.5.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-
packages (from shap) (4.66.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-
packages (from shap) (23.2)
Requirement already satisfied: slicer==0.0.7 in /usr/local/lib/python3.10/dist-
packages (from shap) (0.0.7)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages
(from shap) (0.58.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-
packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba->shap) (0.41.1)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->shap) (2023.3.post1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn->shap) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (3.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.1->pandas->shap) (1.16.0)
```

```
[ ]: # Assuming you have a specific index for the sequence you want to visualize
     sequence_index = 3  # Change this to the index you want to visualize
```

```
# Extract the input data from the test dataset
for i, sample in enumerate(test_dataset):
    if i == sequence_index:
        input_data, _ = sample
        break

# Ensure the input data is in the correct shape
if len(input_data['input_ids'].shape) == 1:
    sequences_input = np.expand_dims(input_data['input_ids'], axis=0)
else:
    sequences_input = input_data['input_ids']

# Predict with the intermediate model
gru_output = intermediate_layer_model.predict({'sequences': sequences_input})

# Visualize the GRU activation
plt.imshow(gru_output[0])
plt.colorbar()
plt.title('GRU Activation')
plt.show()
```

```
1/1 [==============================] - 8s 8s/step
```



```
[ ]: cnn_output = intermediate_layer_model.predict({'sequences': sequences_input})
lstm_output = intermediate_layer_model.predict({'sequences': sequences_input})
ensemble_output = intermediate_layer_model.predict({'sequences':␣
↪sequences_input})
```

```
[ ]: # Print layers to identify the desired layer's index
for i, layer in enumerate(gru_model.layers):
    print(i, layer.name, layer.__class__.__name__)

# Create the intermediate model (adjust the index accordingly)
desired_layer_index = 3
intermediate_layer_model = Model(inputs=gru_model.input, outputs=gru_model.
↪layers[desired_layer_index].output)

input_data = {'input_ids': ...}
if len(input_data['input_ids'].shape) == 1:
    sequences_input = np.expand_dims(input_data['input_ids'], axis=0)
else:
    sequences_input = input_data['input_ids']

# Predict with the intermediate model
gru_output = intermediate_layer_model.predict({'sequences': sequences_input})

# Visualize
plt.imshow(gru_output[0])
plt.colorbar()
plt.title('GRU Activation')
plt.show()
```

```
0 sequences InputLayer
1 embedding_1 Embedding
2 bidirectional_4 Bidirectional
3 dropout_43 Dropout
4 bidirectional_5 Bidirectional
5 dropout_44 Dropout
6 bidirectional_6 Bidirectional
7 dropout_45 Dropout
8 gru_8 GRU
9 dropout_46 Dropout
10 dense_2 Dense
11 dropout_47 Dropout
12 dense_3 Dense
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-192-9abc00062de2> in <cell line: 10>()
      8
      9 input_data = {'input_ids': …}
---> 10 if len(input_data['input_ids'].shape) == 1:
     11     sequences_input = np.expand_dims(input_data['input_ids'], axis=0)
     12 else:

AttributeError: 'ellipsis' object has no attribute 'shape'
```

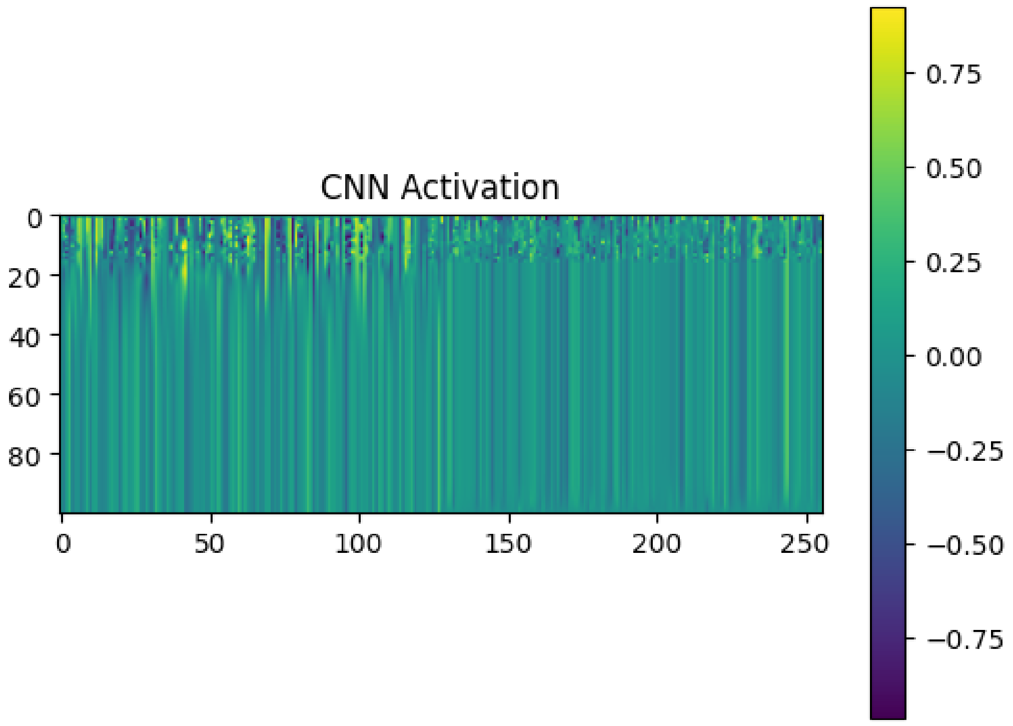
[ ]:
```
# Print layers to identify the desired layer's index
for i, layer in enumerate(cnn_model.layers):
    print(i, layer.name, layer.__class__.__name__)

# Create the intermediate model (adjust the index accordingly)
desired_layer_index = 3
intermediate_layer_model = Model(inputs=cnn_model.input, outputs=cnn_model.
 ↪layers[desired_layer_index].output)

input_data = {'input_ids': ...}
if len(input_data['input_ids'].shape) == 1:
    sequences_input = np.expand_dims(input_data['input_ids'], axis=0)
else:
    sequences_input = input_data['input_ids']

# Predict with the intermediate model
cnn_output = intermediate_layer_model.predict({'sequences': sequences_input})

# Visualize
plt.imshow(gru_output[0])
plt.colorbar()
plt.title('CNN Activation')
plt.show()
```

[ ]:
```
# Print layers to identify the desired layer's index
for i, layer in enumerate(cnn_model.layers):
    print(i, layer.name, layer.__class__.__name__)

# Create the intermediate model (adjust the index accordingly)
desired_layer_index = 3
intermediate_layer_model = Model(inputs=cnn_model.input, outputs=cnn_model.
 ↪layers[desired_layer_index].output)

input_data = {'input_ids': ...}
if len(input_data['input_ids'].shape) == 1:
    sequences_input = np.expand_dims(input_data['input_ids'], axis=0)
else:
    sequences_input = input_data['input_ids']
```

```python
# Predict with the intermediate model
cnn_output = intermediate_layer_model.predict({'sequences': sequences_input})

# Visualize
plt.imshow(gru_output[0])
plt.colorbar()
plt.title('CNN Activation')
plt.show()
```

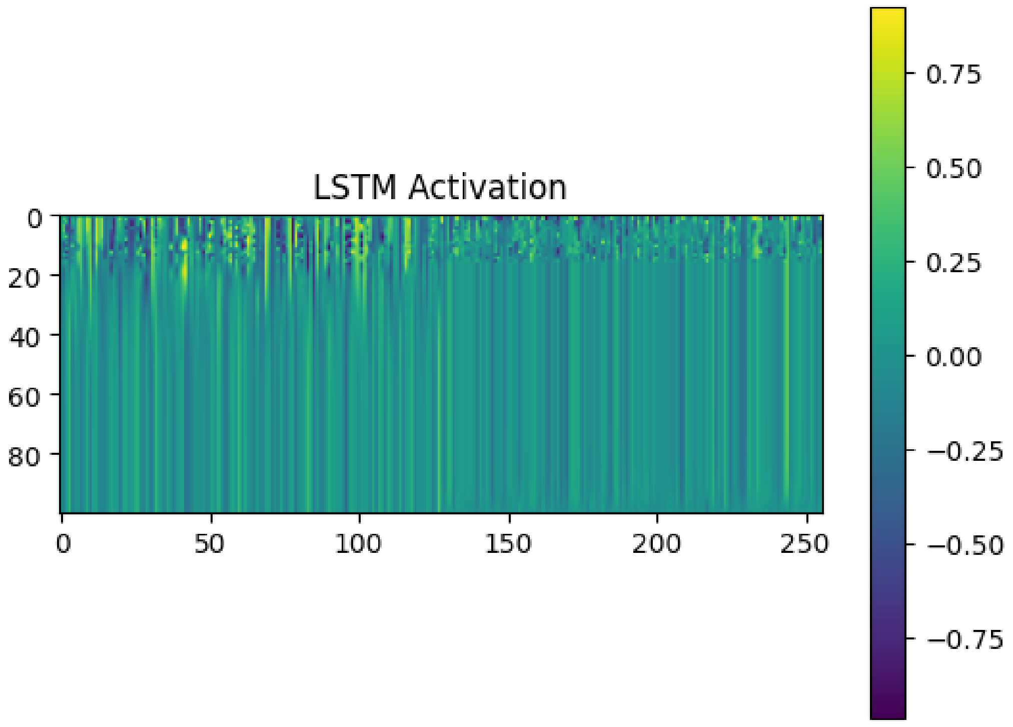
```
WARNING:tensorflow:6 out of the last 790 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x79d7b510aa70>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
0 sequences InputLayer
1 embedding_4 Embedding
2 conv1d_2 Conv1D
3 max_pooling1d_2 MaxPooling1D
4 conv1d_3 Conv1D
5 max_pooling1d_3 MaxPooling1D
6 flatten_1 Flatten
7 dense_8 Dense
8 dense_9 Dense
9 dense_10 Dense
10 dense_11 Dense
1/1 [==============================] - 0s 78ms/step
```



```
[ ]: # Print layers to identify the desired layer's index
     for i, layer in enumerate(lstm_model.layers):
         print(i, layer.name, layer.__class__.__name__)

     # Create the intermediate model (adjust the index accordingly)
     desired_layer_index = 3
     intermediate_layer_model = Model(inputs=lstm_model.input, outputs=lstm_model.
      ↪layers[desired_layer_index].output)


     if len(input_data['input_ids'].shape) == 1:
         sequences_input = np.expand_dims(input_data['input_ids'], axis=0)
     else:
         sequences_input = input_data['input_ids']

     # Predict with the intermediate model
     cnn_output = intermediate_layer_model.predict({'sequences': sequences_input})

     # Visualize
     plt.imshow(gru_output[0])
     plt.colorbar()
     plt.title('LSTM Activation')
     plt.show()
```

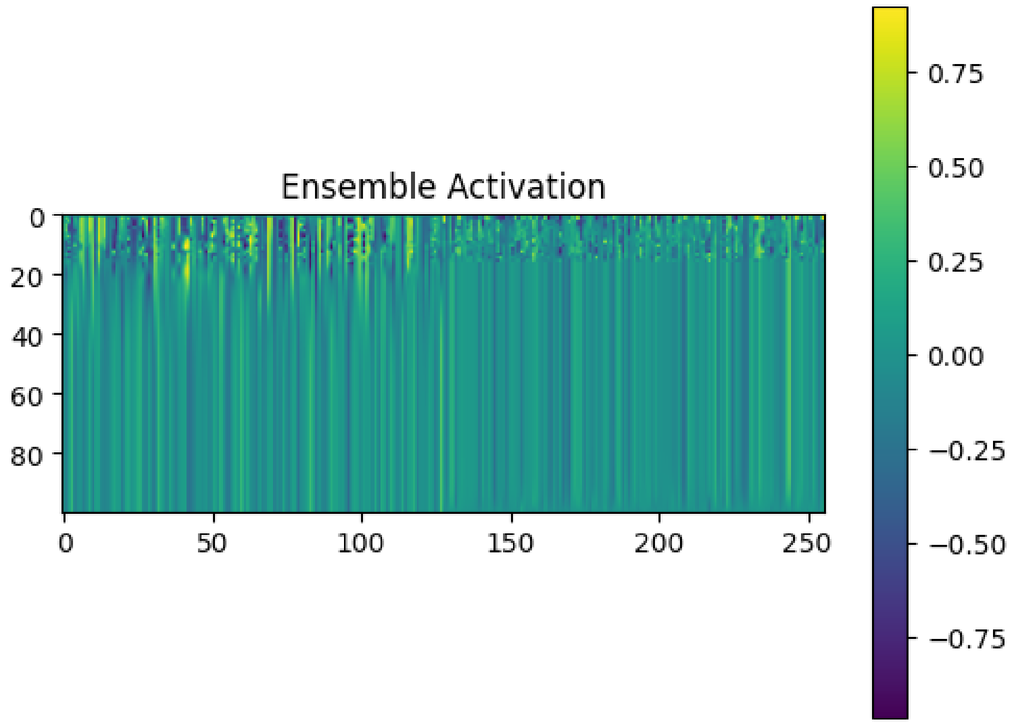
```
0 sequences InputLayer
1 embedding_2 Embedding
2 bidirectional_7 Bidirectional
3 dropout_48 Dropout
4 lstm_1 LSTM
5 dropout_49 Dropout
6 dense_4 Dense
7 dropout_50 Dropout
8 dense_5 Dense
1/1 [==============================] - 2s 2s/step
```



```
[ ]: # Print layers to identify the desired layer's index
for i, layer in enumerate(ensemble_model.layers):
    print(i, layer.name, layer.__class__.__name__)


desired_layer_index = 3
intermediate_layer_model = Model(inputs=ensemble_model.input,␣
 ↪outputs=ensemble_model.layers[desired_layer_index].output)


if len(input_data['input_ids'].shape) == 1:
    sequences_input = np.expand_dims(input_data['input_ids'], axis=0)
```

```
else:
    sequences_input = input_data['input_ids']

# Predict with the intermediate model
cnn_output = intermediate_layer_model.predict({'sequences': sequences_input})

# Visualize
plt.imshow(gru_output[0])
plt.colorbar()
plt.title('Ensemble Activation')
plt.show()
```

```
0 sequences InputLayer
1 embedding_5 Embedding
2 conv1d_2 Conv1D
3 max_pooling1d_2 MaxPooling1D
4 conv1d_3 Conv1D
5 max_pooling1d_3 MaxPooling1D
6 flatten_1 Flatten
7 dense_12 Dense
8 dense_13 Dense
9 dense_14 Dense
10 dense_15 Dense
1/1 [==============================] - 0s 48ms/step
```



```
[ ]:
```