nlp-hybrid-ensemble

May 15, 2024

[ ]:
```
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()
```

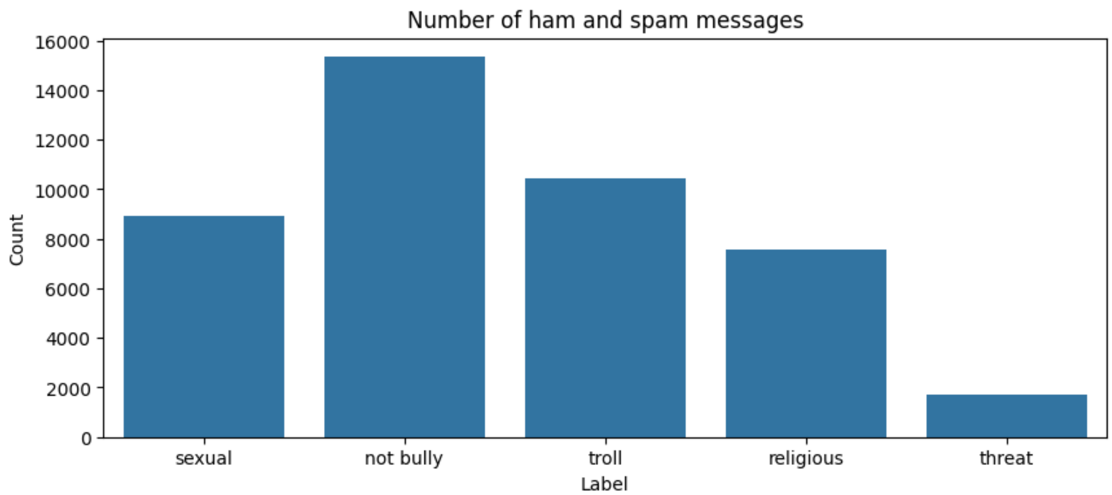
[ ]:
```
comment                0
Category               0
Gender                 0
comment react number   3
label                  0
dtype: int64
```

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

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

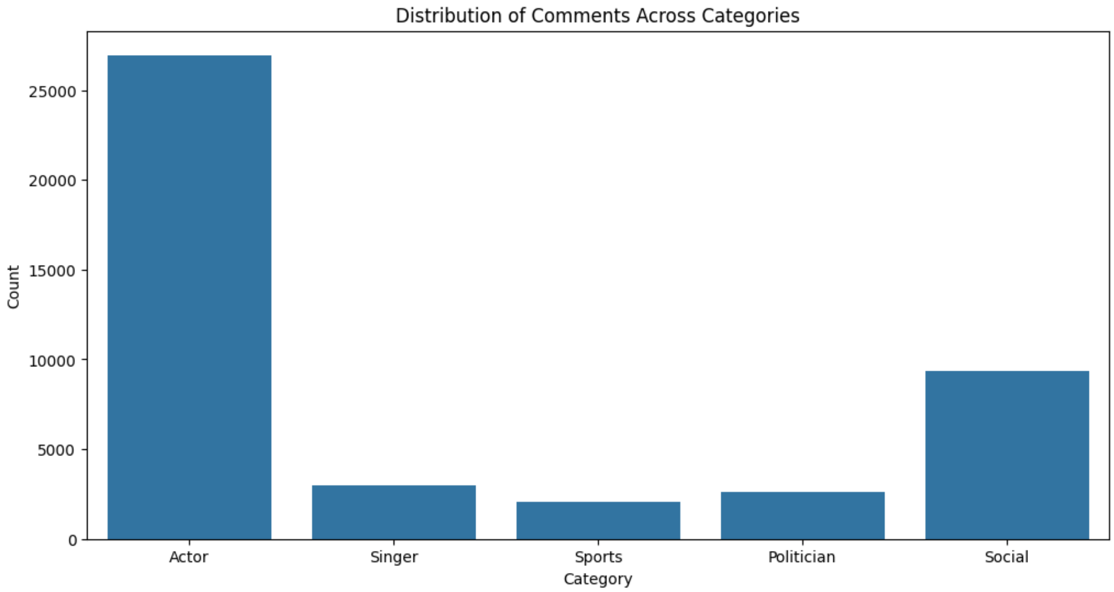
[ ]:
```
not bully    15339
troll        10462
sexual        8928
religious     7575
threat        1694
Name: label, dtype: int64
```

[ ]:
```
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()
```

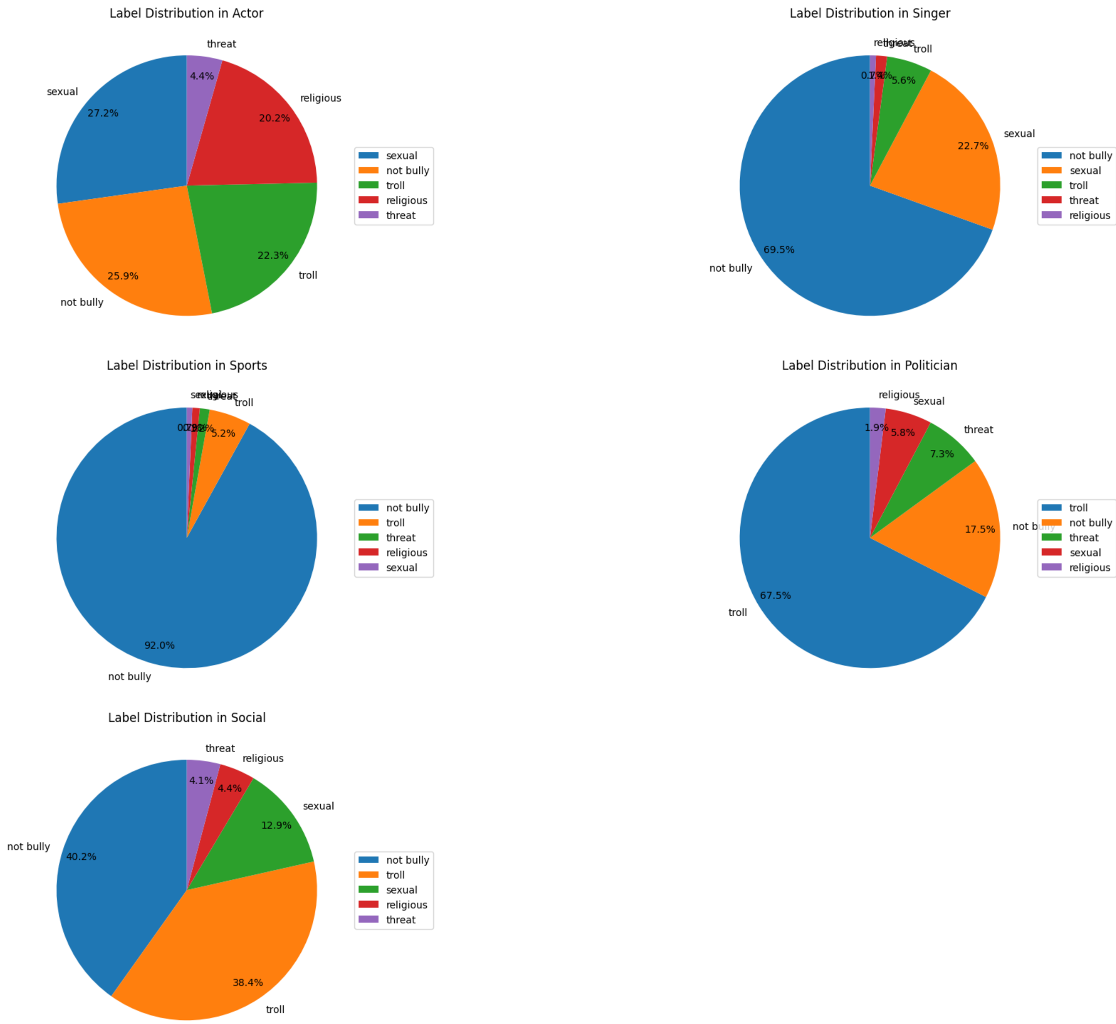


[ ]:
```
plt.figure(figsize=(12, 6))
sns.countplot(x='Category', data=df)
plt.title('Distribution of Comments Across Categories')
```

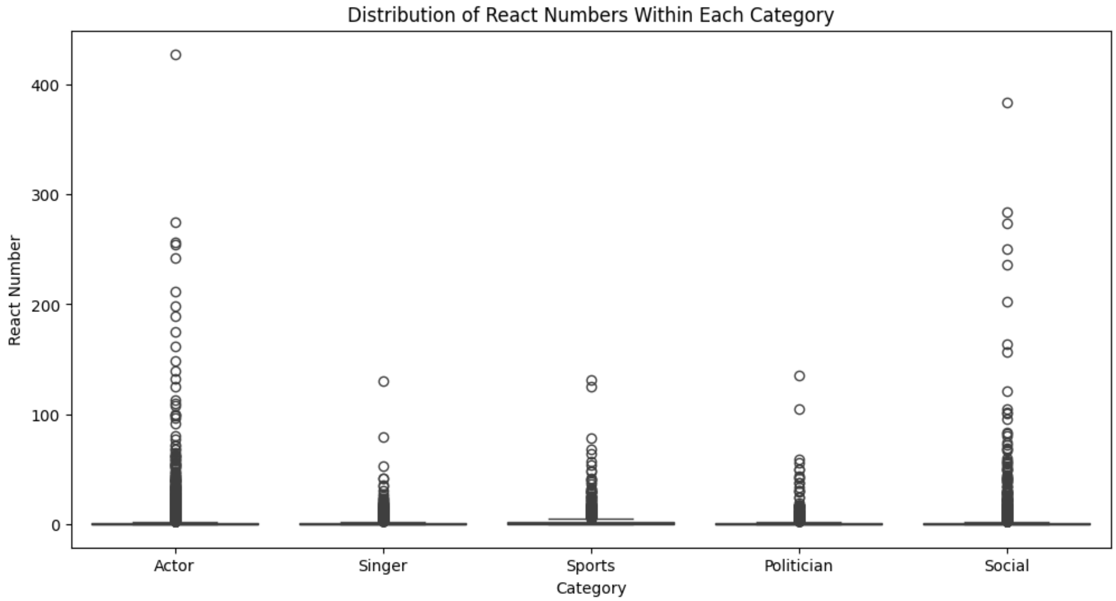
```
plt.xlabel('Category')
plt.ylabel('Count')
plt.show()
```



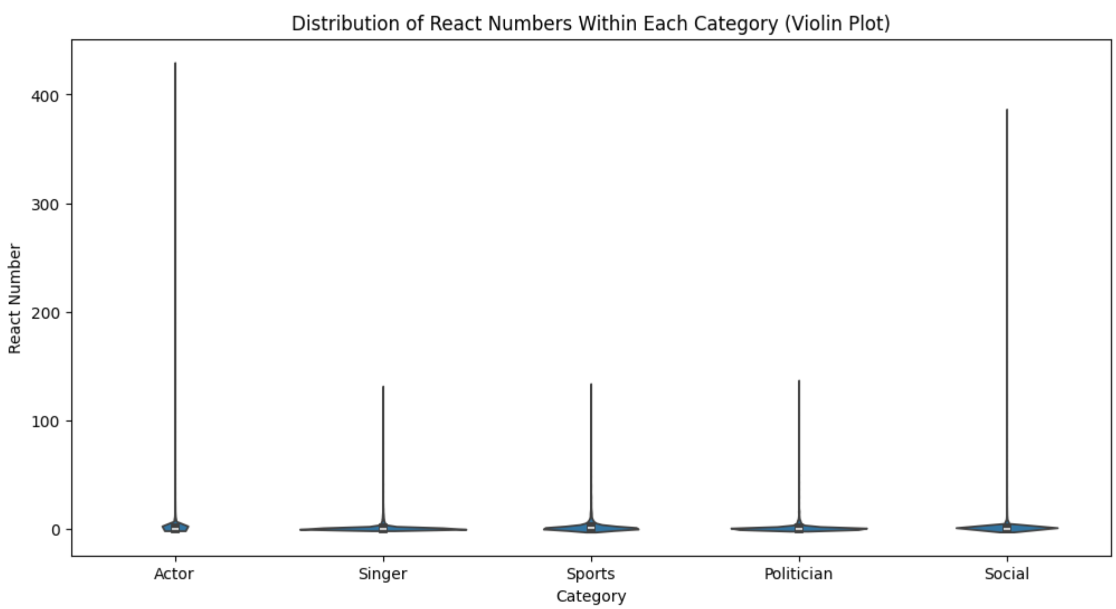
[ ]:
```
# Step 2: Pie Charts for percentage distribution of labels within each category
plt.figure(figsize=(20, 15))  # Increase the figure size
categories = df['Category'].unique()
num_categories = len(categories)

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()
```



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

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

[ ]: True

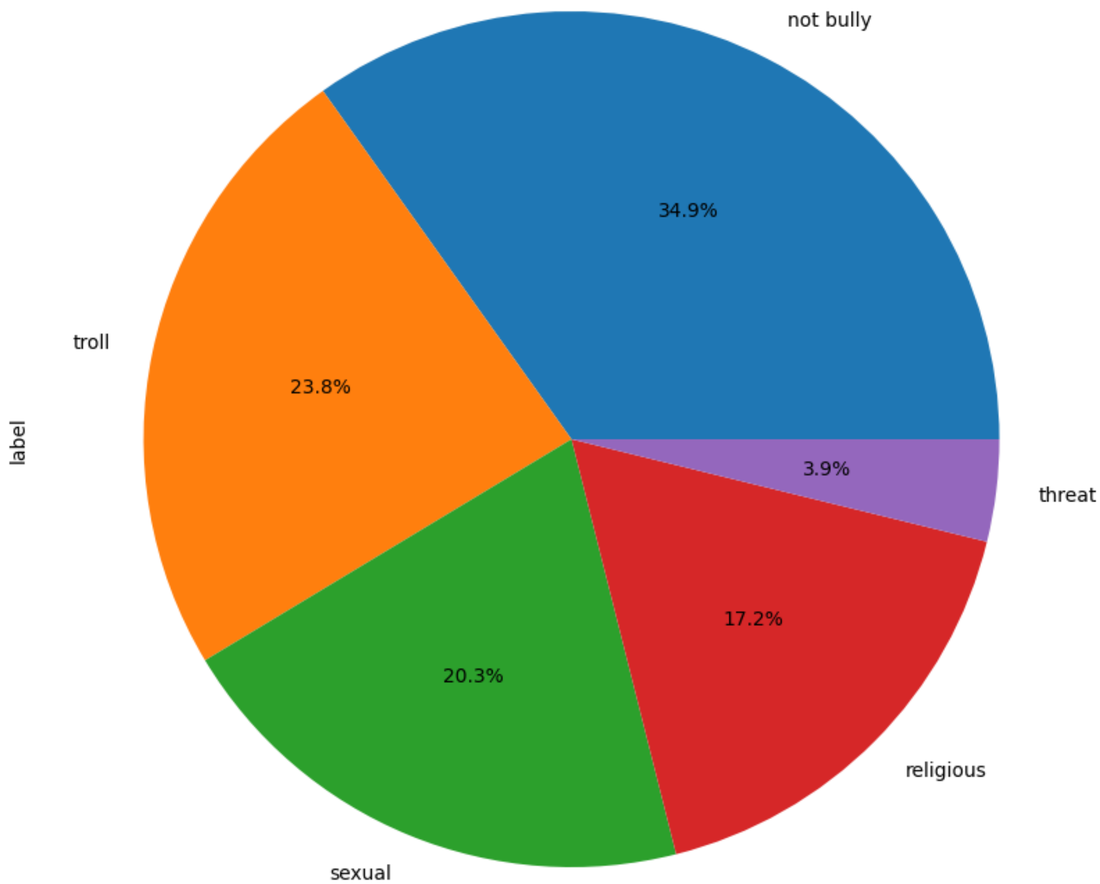
[ ]:
```python
import nltk
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')

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.
```

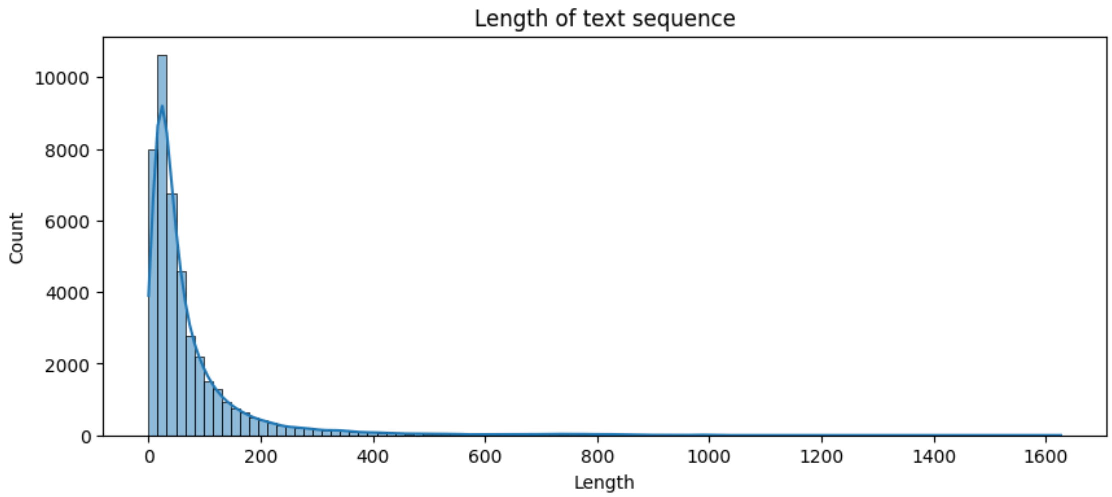
```
{'  ', '  ', '  ', '   ', '  ', ' ', '   ', ' ', '  ',
'   ', '   ', ' ', '   ', '  ', '  ', '   ', ' ', '   ',
'  ', '  ', '  ', ' ', '  ', '  ', '   ', ' ', '  ', '   ',
'  ', ' ', '  ', '  ', '  ', '  ', ' ', '  ', '  ', '  ',
'   ', '   ', '  ', ' ', '   ', '  ', '  ', '  ', ' ', '  ',
'  ',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

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

[ ]:
```
# 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","\u
  ␣
↪"\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)
```

```
[ ]: 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
```

```
[ ]: # 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)
```

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

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

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

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

```
[ ]: 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 = [
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
```

```
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ', ' ',
#     ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',␣
  ↪' ',
```

```
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
#   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
```

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

[ ]:
```
## 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
------------------------------------------------------------
```

[ ]:
```
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
```

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

[ ]:
```
0    15339
4    10462
2     8928
1     7575
3     1694
Name: label, dtype: int64
```

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

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

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

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88:
```

```
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']

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 tensorflow.keras.layers import Input, Bidirectional, GRU, Dropout, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Embedding
from tensorflow.keras import regularizers
```

```
[ ]: vocab_size = tokenizer.vocab_size + 1
embedding_dim = 300
```

```
[ ]: import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, GRU, Dense, Embedding,␣
 ↪Bidirectional, Concatenate, GlobalMaxPooling1D
```

```
[ ]: 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)

    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)
print(model.summary())
```

```
Model: "model"

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

 embedding (Embedding)       (None, 128, 128)          13052800

 bidirectional (Bidirection  (None, 128, 512)          592896
 al)

 bidirectional_1 (Bidirecti  (None, 256)               493056
 onal)

 dense (Dense)               (None, 64)                16448

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 128)               8320

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 64)                8256

 dropout_2 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 5)                 165

=================================================================
Total params: 14174021 (54.07 MB)
Trainable params: 14174021 (54.07 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

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

[ ]:
```python
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)))
```

[ ]:
```python
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 [==============================] - 77s 58ms/step - loss: 0.7938 -
accuracy: 0.7197 - val_loss: 0.5848 - val_accuracy: 0.8100
Epoch 2/10
1100/1100 [==============================] - 40s 37ms/step - loss: 0.5272 -
accuracy: 0.8372 - val_loss: 0.5304 - val_accuracy: 0.8234
Epoch 3/10
1100/1100 [==============================] - 39s 35ms/step - loss: 0.4165 -
accuracy: 0.8700 - val_loss: 0.5577 - val_accuracy: 0.8240
Epoch 4/10
1100/1100 [==============================] - 40s 37ms/step - loss: 0.3457 -
accuracy: 0.8922 - val_loss: 0.5523 - val_accuracy: 0.8293
Epoch 5/10
1100/1100 [==============================] - 40s 36ms/step - loss: 0.2870 -
accuracy: 0.9090 - val_loss: 0.5983 - val_accuracy: 0.8234
```

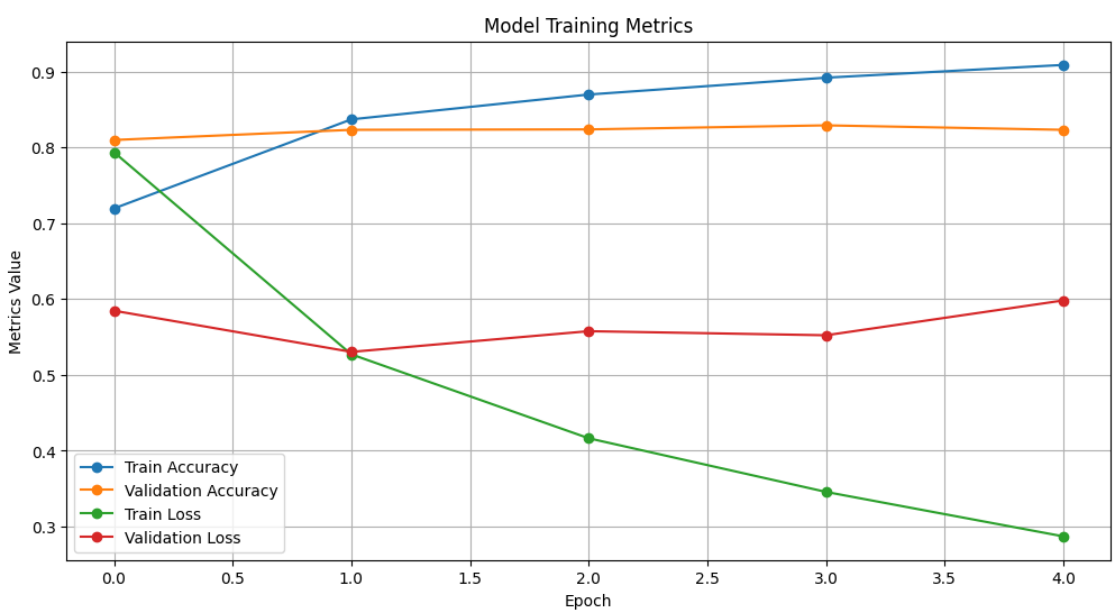
[ ]:
```
# Save the entire model to a HDF5 file
model.save('/content/drive/MyDrive/Bully/GRU_Model')
```

[ ]:
```
# 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 [==============================] - 4s 12ms/step - loss: 0.5304 -
accuracy: 0.8234
Test Accuracy: 82.34%
Test Loss: 0.5304
275/275 [==============================] - 5s 11ms/step
```

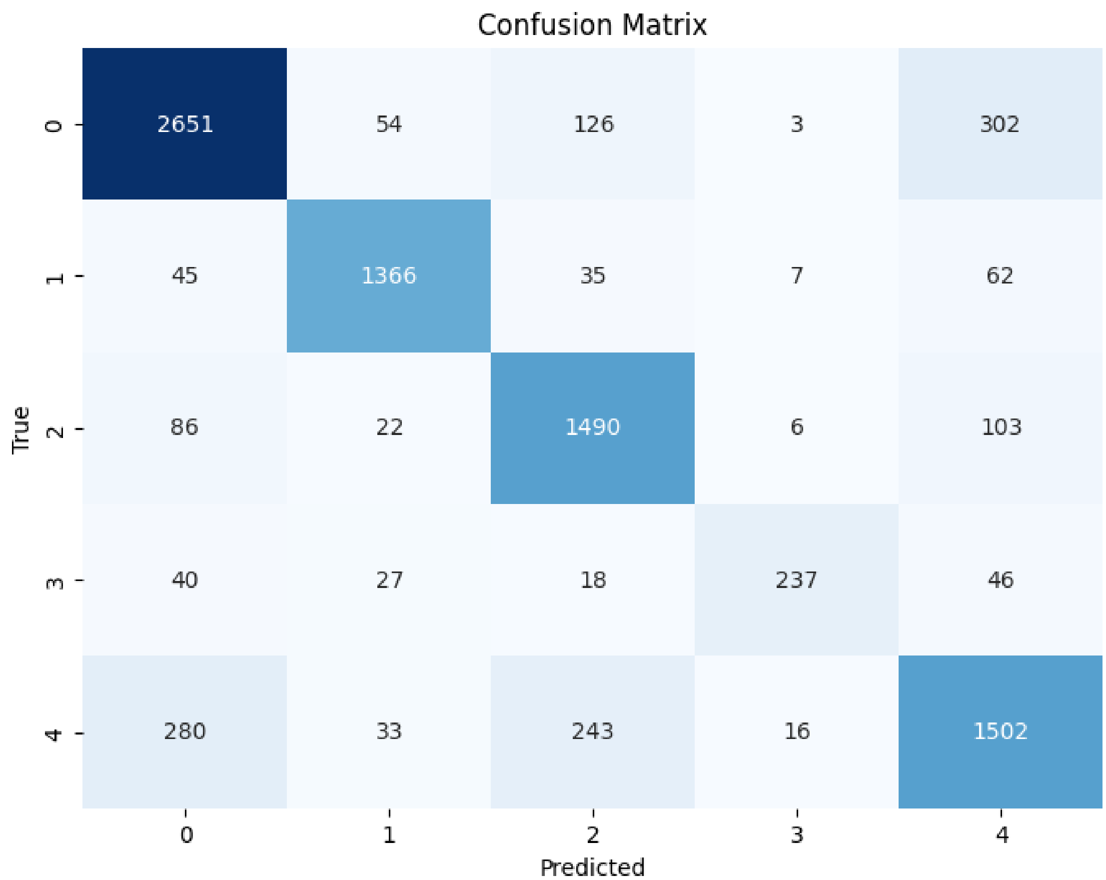
[ ]:
```
# Plot training & validation accuracy and loss values in a single plot
plt.figure(figsize=(12, 6))

# Plot accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy', marker='o')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy',␣
↪marker='o')

# Plot loss
plt.plot(history.history['loss'], label='Train Loss', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')

plt.title('Model Training Metrics')
plt.xlabel('Epoch')
plt.ylabel('Metrics Value')
plt.legend()
plt.grid(True)
plt.show()
```



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

[ ]:
```python
# 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       0.85      0.85      0.85      3136
           1       0.91      0.90      0.91      1515
           2       0.78      0.87      0.82      1707
           3       0.88      0.64      0.74       368
           4       0.75      0.72      0.73      2074

    accuracy                           0.82      8800
   macro avg       0.83      0.80      0.81      8800
weighted avg       0.82      0.82      0.82      8800

Confusion Matrix:
 [[2651   54  126    3  302]
 [  45 1366   35    7   62]
 [  86   22 1490    6  103]
 [  40   27   18  237   46]
 [ 280   33  243   16 1502]]
```



[ ]:
```
from sklearn.metrics import confusion_matrix, classification_report,␣
↪roc_auc_score, roc_curve, auc
```

[ ]:
```
from sklearn.metrics import confusion_matrix, classification_report,␣
↪roc_auc_score, roc_curve, auc
import numpy as np

# Convert predicted classes to class names
class_names = ["Not Bully", "Troll", "Sexual", "Religious", "Threat"]
predicted_class_names = [class_names[i] for i in predicted_classes]

# Convert true classes to class names
true_class_names = [class_names[i] for i in true_classes]

# Confusion Matrix
cm = confusion_matrix(true_class_names, predicted_class_names)
print("Confusion Matrix:")
print(cm)
```

```
# Classification Report
report = classification_report(true_class_names, predicted_class_names)
print("Classification Report:")
print(report)
```

```
Confusion Matrix:
[[2651    3  126  302   54]
 [  40  237   18   46   27]
 [  86    6 1490  103   22]
 [ 280   16  243 1502   33]
 [  45    7   35   62 1366]]
Classification Report:
              precision    recall  f1-score   support

   Not Bully       0.85      0.85      0.85      3136
   Religious       0.88      0.64      0.74       368
      Sexual       0.78      0.87      0.82      1707
      Threat       0.75      0.72      0.73      2074
       Troll       0.91      0.90      0.91      1515

    accuracy                           0.82      8800
   macro avg       0.83      0.80      0.81      8800
weighted avg       0.82      0.82      0.82      8800
```

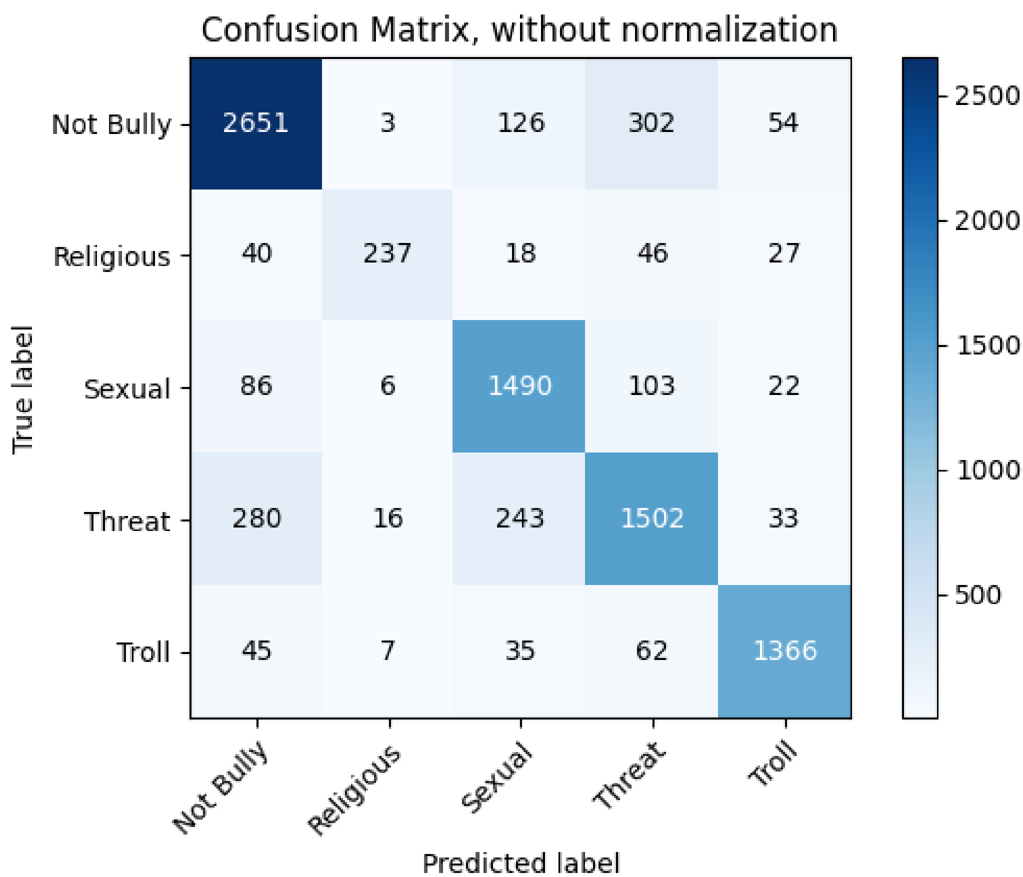
[ ]:
```
from sklearn.utils.multiclass import unique_labels
```

[ ]:
```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.utils.multiclass import unique_labels

# Convert predicted classes to class names
class_names = ["Not Bully", "Troll", "Sexual", "Religious", "Threat"]
predicted_class_names = [class_names[i] for i in predicted_classes]

# Convert true classes to class names
true_class_names = [class_names[i] for i in true_classes]

# Confusion Matrix
cm = confusion_matrix(true_class_names, predicted_class_names)

# Plot Confusion Matrix
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
```

```
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized Confusion Matrix'
        else:
            title = 'Confusion Matrix'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = unique_labels(y_true, y_pred)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(len(classes)),
           yticks=np.arange(len(classes)),
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(len(classes)):
        for j in range(len(classes)):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax
```

```python
# Plot non-normalized confusion matrix
plot_confusion_matrix(true_class_names, predicted_class_names, classes=np.
 ↪array(class_names),
                      title='Confusion Matrix, without normalization')

plt.show()
```

```
Confusion matrix, without normalization
```



[ ]:
```python
# 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: 0.8248
Recall: 0.8234
```

```
F1-score: 0.8228
```

[ ]:
```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: 1373
True Negatives: 2725
False Positives: 64
False Negatives: 58
```

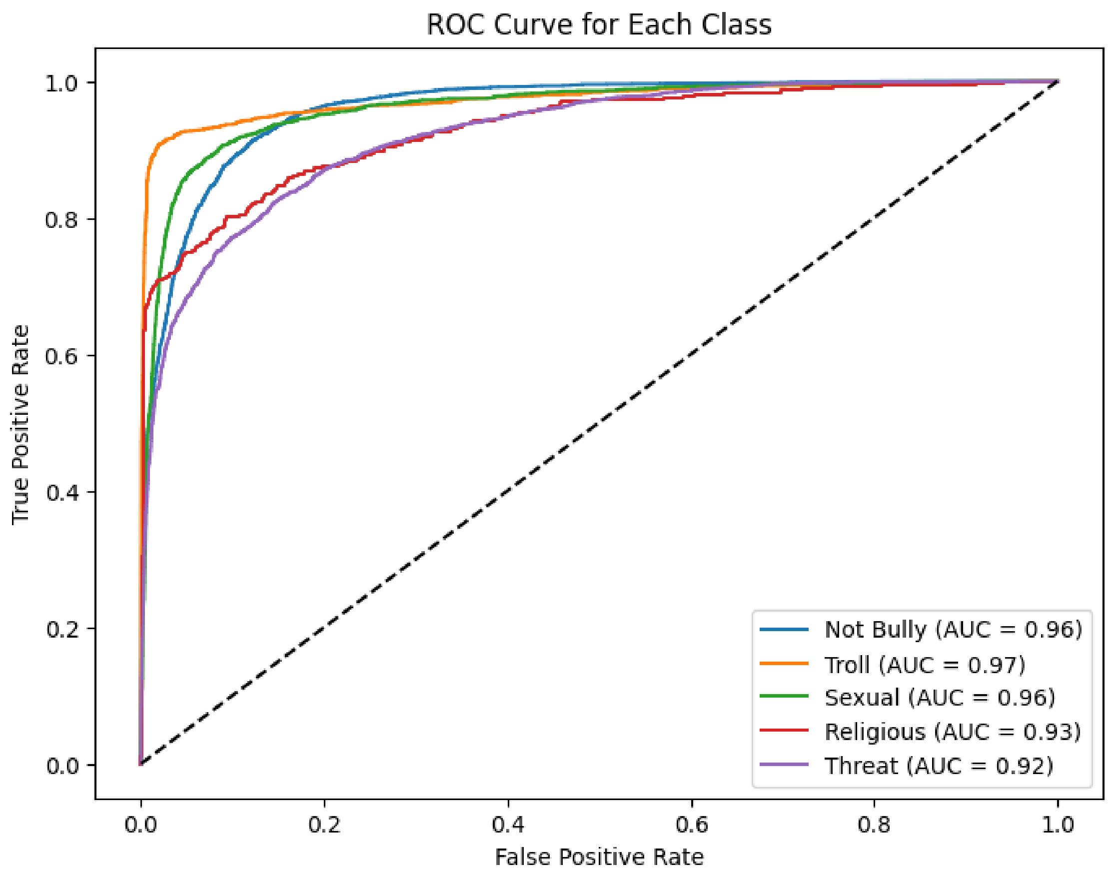
[ ]:
```python
# Calculate TP, FP, TN, FN for each class
for i, class_name in enumerate(class_names):
    tp = cm[i, i]
    fp = np.sum(cm[:, i]) - tp
    fn = np.sum(cm[i, :]) - tp
    tn = np.sum(cm) - tp - fp - fn

    print(f"\nClass: {class_name}")
    print(f"True Positives (TP): {tp}")
    print(f"False Positives (FP): {fp}")
    print(f"True Negatives (TN): {tn}")
    print(f"False Negatives (FN): {fn}")
```

```
Class: Not Bully
True Positives (TP): 2651
False Positives (FP): 451
True Negatives (TN): 5213
False Negatives (FN): 485

Class: Troll
True Positives (TP): 237
False Positives (FP): 32
True Negatives (TN): 8400
False Negatives (FN): 131

Class: Sexual
True Positives (TP): 1490
False Positives (FP): 422
True Negatives (TN): 6671
```

```
False Negatives (FN): 217

Class: Religious
True Positives (TP): 1502
False Positives (FP): 513
True Negatives (TN): 6213
False Negatives (FN): 572

Class: Threat
True Positives (TP): 1366
False Positives (FP): 136
True Negatives (TN): 7149
False Negatives (FN): 149
```

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

[ ]:
```
# Convert true classes 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_names[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()
```



LSTM Model

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

[ ]:
```
def LSTM_Model(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
    x = Bidirectional(LSTM(128, return_sequences=True))(embedded_sequences)
    x = Dropout(0.5)(x)

    # Add another LSTM layer
    x = Bidirectional(LSTM(64))(x)
    x = Dropout(0.5)(x)
```

```python
    # Dense layers
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.5)(x)

    num_classes = len(set(train_labels))  # Ensure train_labels are accessible␣
↪here
    x = Dense(num_classes, activation='softmax')(x)

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

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

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

 embedding_1 (Embedding)     (None, 128, 128)          13052800

 bidirectional_2 (Bidirecti  (None, 128, 256)          263168
 onal)

 dropout_3 (Dropout)         (None, 128, 256)          0

 bidirectional_3 (Bidirecti  (None, 128)               164352
 onal)

 dropout_4 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 64)                8256

 dropout_5 (Dropout)         (None, 64)                0

 dense_6 (Dense)             (None, 5)                 325

=================================================================
Total params: 13488901 (51.46 MB)
Trainable params: 13488901 (51.46 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

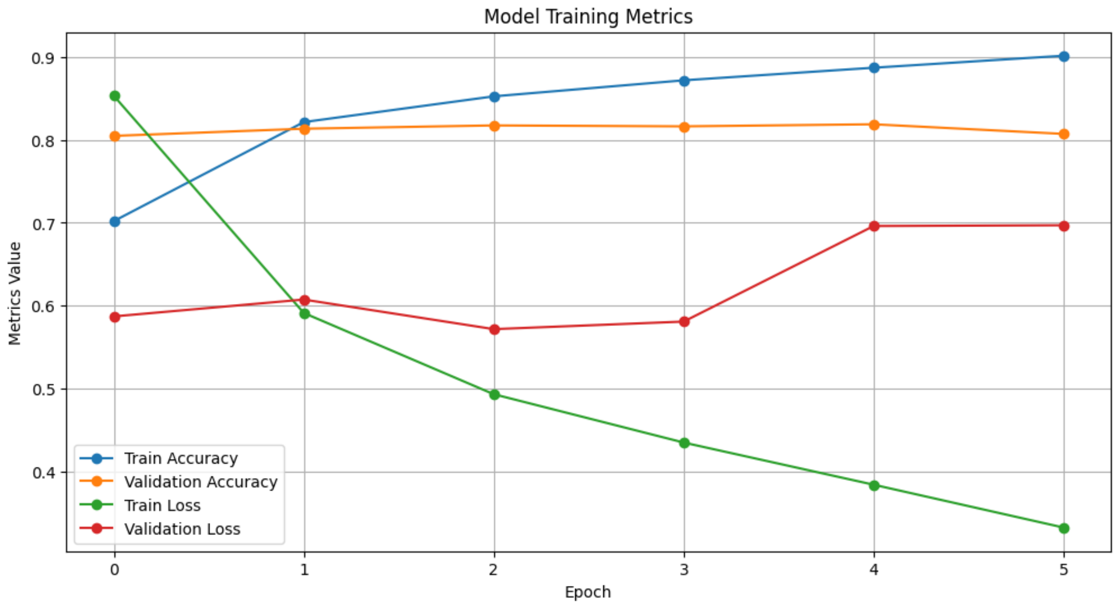
[ ]:
```python
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```python
# 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 [==============================] - 72s 58ms/step - loss: 0.8531 -
accuracy: 0.7021 - val_loss: 0.5872 - val_accuracy: 0.8048
Epoch 2/10
1100/1100 [==============================] - 33s 30ms/step - loss: 0.5914 -
accuracy: 0.8214 - val_loss: 0.6076 - val_accuracy: 0.8134
Epoch 3/10
1100/1100 [==============================] - 31s 29ms/step - loss: 0.4935 -
accuracy: 0.8524 - val_loss: 0.5719 - val_accuracy: 0.8174
Epoch 4/10
1100/1100 [==============================] - 32s 29ms/step - loss: 0.4352 -
accuracy: 0.8717 - val_loss: 0.5810 - val_accuracy: 0.8163
Epoch 5/10
1100/1100 [==============================] - 30s 27ms/step - loss: 0.3844 -
accuracy: 0.8869 - val_loss: 0.6961 - val_accuracy: 0.8188
Epoch 6/10
1100/1100 [==============================] - 30s 27ms/step - loss: 0.3328 -
accuracy: 0.9014 - val_loss: 0.6971 - val_accuracy: 0.8072
```

[ ]:
```python
# Plot training & validation accuracy and loss values in a single plot
plt.figure(figsize=(12, 6))

# Plot accuracy
plt.plot(history_1.history['accuracy'], label='Train Accuracy', marker='o')
plt.plot(history_1.history['val_accuracy'], label='Validation Accuracy',␣
 ↪marker='o')

# Plot loss
plt.plot(history_1.history['loss'], label='Train Loss', marker='o')
plt.plot(history_1.history['val_loss'], label='Validation Loss', marker='o')

plt.title('Model Training Metrics')
plt.xlabel('Epoch')
```

```
plt.ylabel('Metrics Value')
plt.legend()
plt.grid(True)
plt.show()
```



[ ]:
```
# 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
```

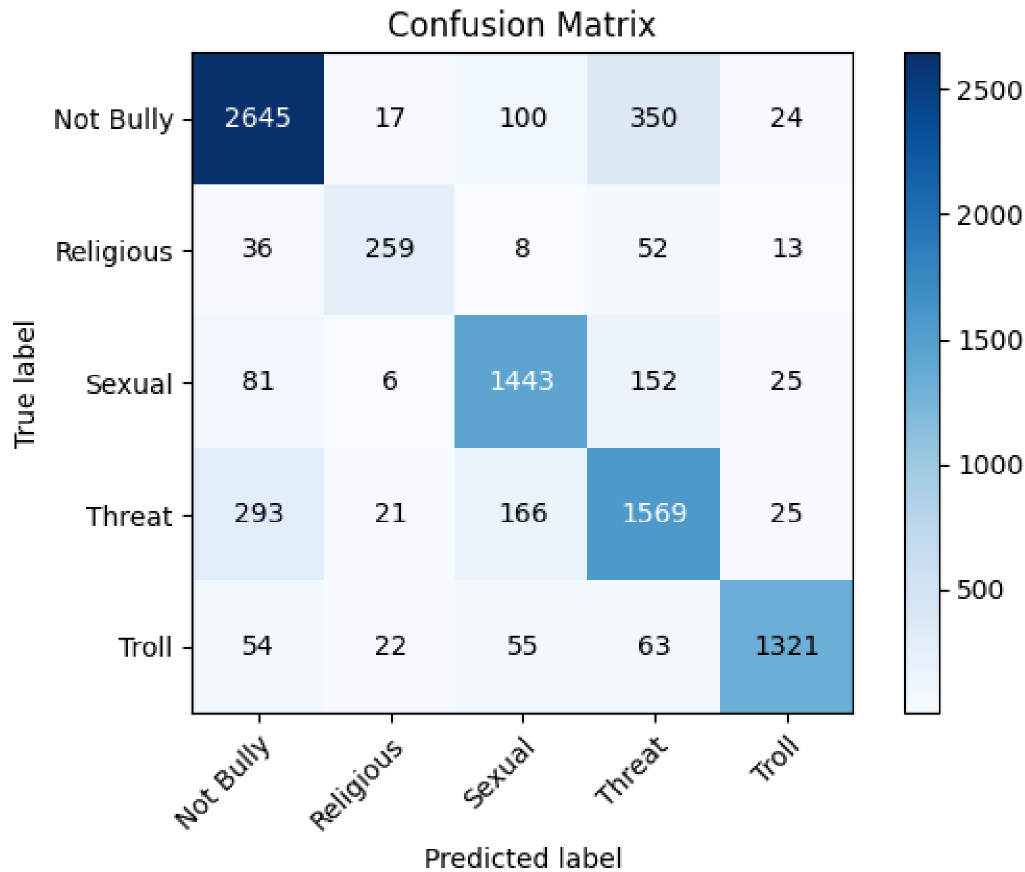
```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/functional.py:642:
UserWarning: Input dict contained keys ['attention_mask'] which did not match
any model input. They will be ignored by the model.
  inputs = self._flatten_to_reference_inputs(inputs)
```

```
275/275 [==============================] - 6s 14ms/step - loss: 0.5719 -
accuracy: 0.8174
Test Accuracy: 81.74%
Test Loss: 0.5719
275/275 [==============================] - 6s 14ms/step
```

[ ]:
```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.utils.multiclass import unique_labels

# Convert predicted classes to class names
class_names = ["Not Bully", "Troll", "Sexual", "Religious", "Threat"]
predicted_class_names = [class_names[i] for i in predicted_classes]

# Convert true classes to class names
true_class_names = [class_names[i] for i in true_classes]

# Confusion Matrix
cm = confusion_matrix(true_class_names, predicted_class_names)

# Plot Confusion Matrix
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized Confusion Matrix'
        else:
            title = 'Confusion Matrix'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = unique_labels(y_true, y_pred)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(len(classes)),
           yticks=np.arange(len(classes)),
```

```
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(len(classes)):
        for j in range(len(classes)):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


# Plot non-normalized confusion matrix
plot_confusion_matrix(true_class_names, predicted_class_names, classes=np.
 ↪array(class_names),
                      title='Confusion Matrix')

plt.show()
```

```
Confusion matrix, without normalization
```



[ ]:
```
# Calculate TP, FP, TN, FN for each class
for i, class_name in enumerate(class_names):
    tp = cm[i, i]
    fp = np.sum(cm[:, i]) - tp
    fn = np.sum(cm[i, :]) - tp
    tn = np.sum(cm) - tp - fp - fn

    print(f"\nClass: {class_name}")
    print(f"True Positives (TP): {tp}")
    print(f"False Positives (FP): {fp}")
    print(f"True Negatives (TN): {tn}")
    print(f"False Negatives (FN): {fn}")
```

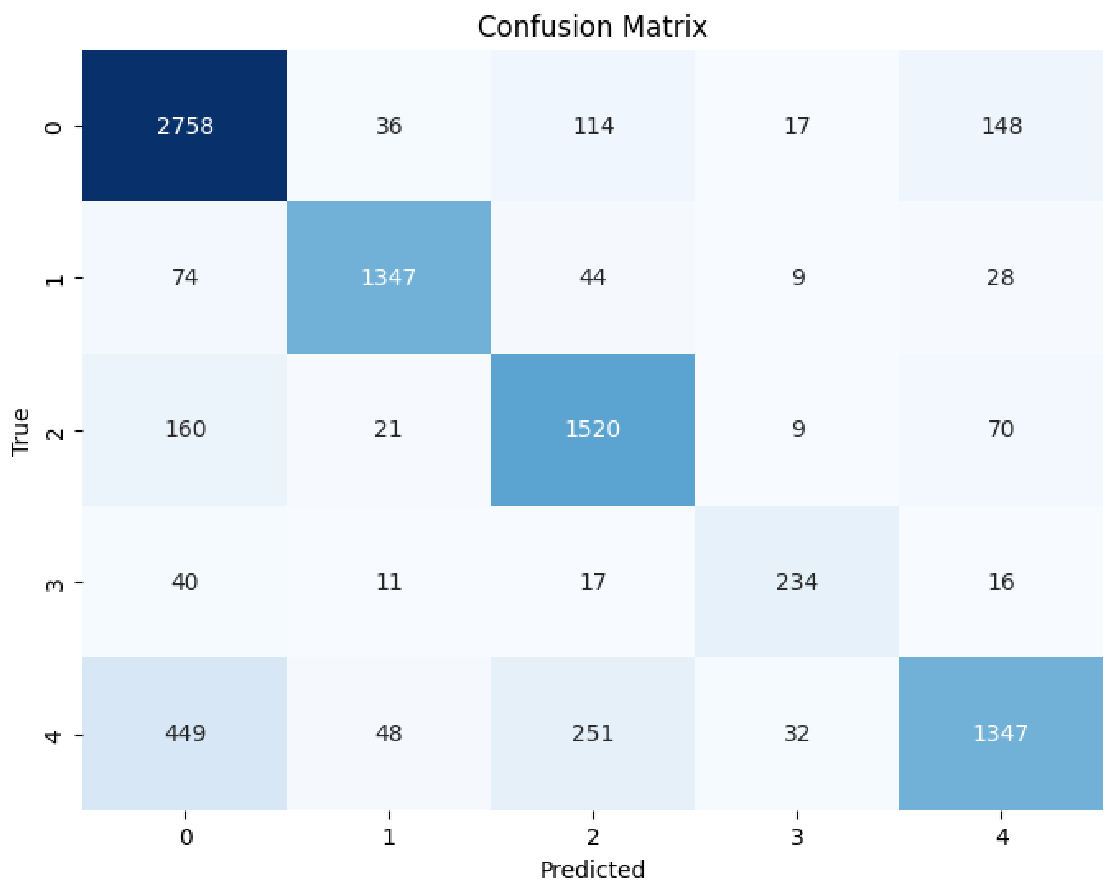
```
Class: Not Bully
True Positives (TP): 2781
False Positives (FP): 659
True Negatives (TN): 5005
False Negatives (FN): 355
```

```
Class: Troll
True Positives (TP): 275
False Positives (FP): 109
True Negatives (TN): 8323
False Negatives (FN): 93

Class: Sexual
True Positives (TP): 1451
False Positives (FP): 386
True Negatives (TN): 6707
False Negatives (FN): 256

Class: Religious
True Positives (TP): 1365
False Positives (FP): 347
True Negatives (TN): 6379
False Negatives (FN): 709

Class: Threat
True Positives (TP): 1321
False Positives (FP): 106
True Negatives (TN): 7179
False Negatives (FN): 194
```

[ ]:
```python
# 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       0.79      0.90      0.84      3073
           1       0.92      0.90      0.91      1502
           2       0.78      0.85      0.82      1780
           3       0.78      0.74      0.76       318
           4       0.84      0.63      0.72      2127
```

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

Confusion Matrix:
 [[2758   36  114   17  148]
 [  74 1347   44    9   28]
 [ 160   21 1520    9   70]
 [  40   11   17  234   16]
 [ 449   48  251   32 1347]]
```



[ ]:
```
# 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: 1347
True Negatives: 2758
False Positives: 36
False Negatives: 74
```

[ ]:
```
# 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}')
```

```
275/275 [==============================] - 2s 9ms/step - loss: 0.5578 -
accuracy: 0.8189
Test Accuracy: 81.89%
Test Loss: 0.5578
```

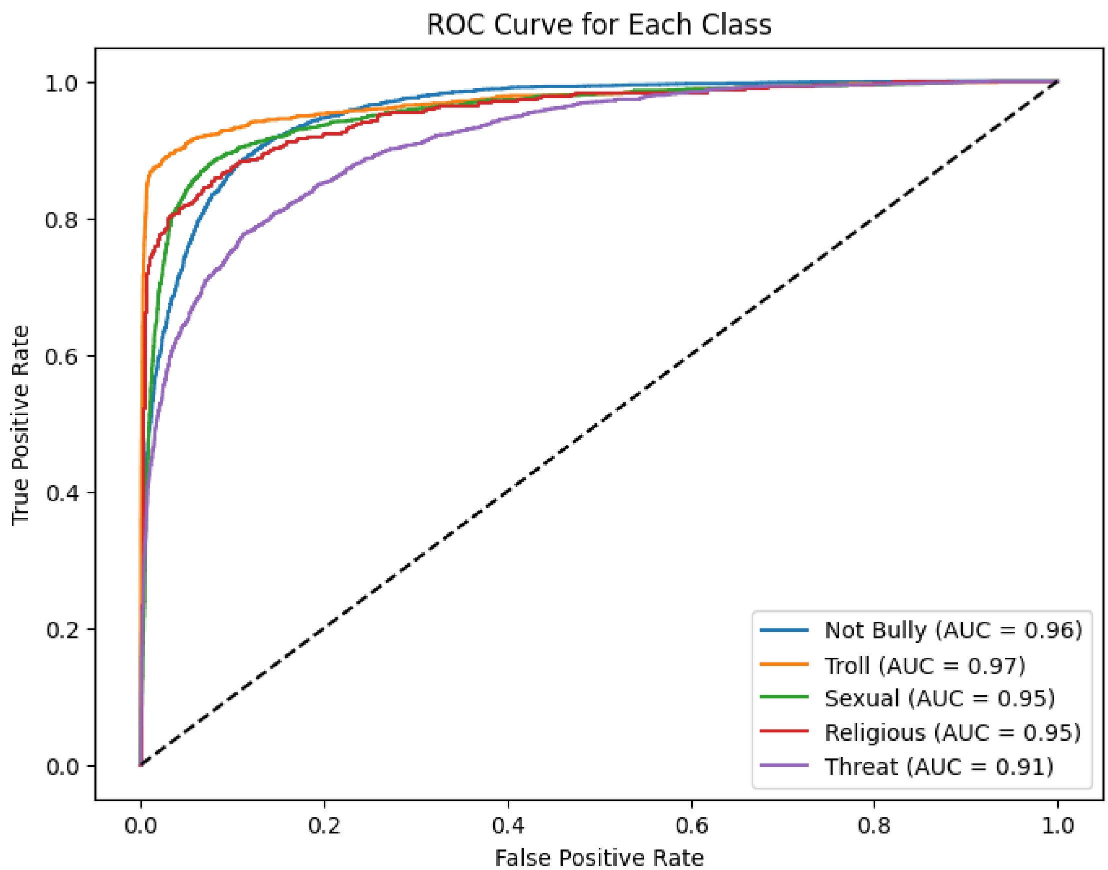
[ ]:
```
# Precision, Recall, F1-score
precision, recall, f1_score, _ = precision_recall_fscore_support(true_classes,
    predicted_classes, average='weighted')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1_score:.4f}')
```

```
Precision: 0.8223
Recall: 0.8189
F1-score: 0.8156
```

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

[ ]:
```
# Convert true classes 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_names[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()
```



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")  # Update input shape
```

```
    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)
    x = Flatten()(x)
    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 #
=================================================================
 sequences (InputLayer)      [(None, 128)]             0

 embedding_2 (Embedding)     (None, 128, 128)          13052800

 conv1d (Conv1D)             (None, 124, 128)          82048

 max_pooling1d (MaxPooling1  (None, 24, 128)           0
 D)

 conv1d_1 (Conv1D)           (None, 20, 128)           82048

 max_pooling1d_1 (MaxPoolin  (None, 4, 128)            0
 g1D)

 flatten (Flatten)           (None, 512)               0

 dense_7 (Dense)             (None, 128)               65664

 dense_8 (Dense)             (None, 64)                8256
```

```
 dense_9 (Dense)             (None, 32)               2080


 dense_10 (Dense)            (None, 5)                165

=================================================================
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
```

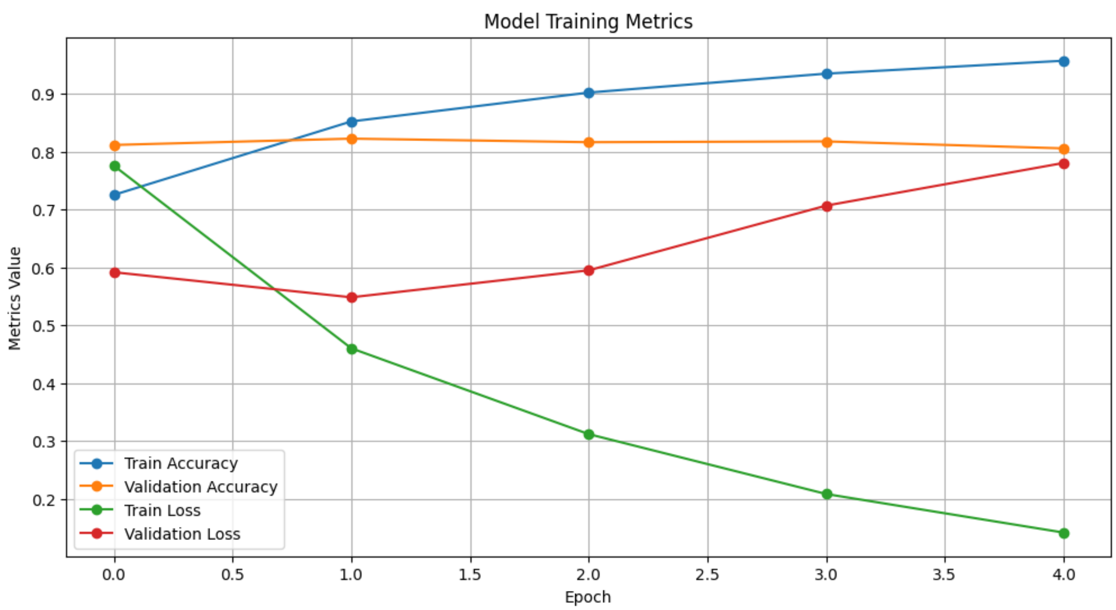
[ ]:
```
early_stopping = EarlyStopping(monitor='val_loss', patience=3,␣
 ↪restore_best_weights=True, verbose=1)
```

[ ]:
```
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 [==============================] - 41s 34ms/step - loss: 0.7757 -
accuracy: 0.7252 - val_loss: 0.5916 - val_accuracy: 0.8115
Epoch 2/10
1100/1100 [==============================] - 14s 13ms/step - loss: 0.4603 -
accuracy: 0.8521 - val_loss: 0.5483 - val_accuracy: 0.8224
Epoch 3/10
1100/1100 [==============================] - 12s 11ms/step - loss: 0.3120 -
accuracy: 0.9020 - val_loss: 0.5949 - val_accuracy: 0.8163
Epoch 4/10
1100/1100 [==============================] - 11s 10ms/step - loss: 0.2086 -
accuracy: 0.9347 - val_loss: 0.7067 - val_accuracy: 0.8176
Epoch 5/10
1097/1100 [============================>.] - ETA: 0s - loss: 0.1416 - accuracy:
0.9570Restoring model weights from the end of the best epoch: 2.
1100/1100 [==============================] - 11s 10ms/step - loss: 0.1420 -
accuracy: 0.9570 - val_loss: 0.7804 - val_accuracy: 0.8056
Epoch 5: early stopping
```

[ ]:
```
# Plot training & validation accuracy and loss values in a single plot
plt.figure(figsize=(12, 6))
```

```
# Plot accuracy
plt.plot(history_2.history['accuracy'], label='Train Accuracy', marker='o')
plt.plot(history_2.history['val_accuracy'], label='Validation Accuracy',␣
  ↪marker='o')

# Plot loss
plt.plot(history_2.history['loss'], label='Train Loss', marker='o')
plt.plot(history_2.history['val_loss'], label='Validation Loss', marker='o')

plt.title('Model Training Metrics')
plt.xlabel('Epoch')
plt.ylabel('Metrics Value')
plt.legend()
plt.grid(True)
plt.show()
```



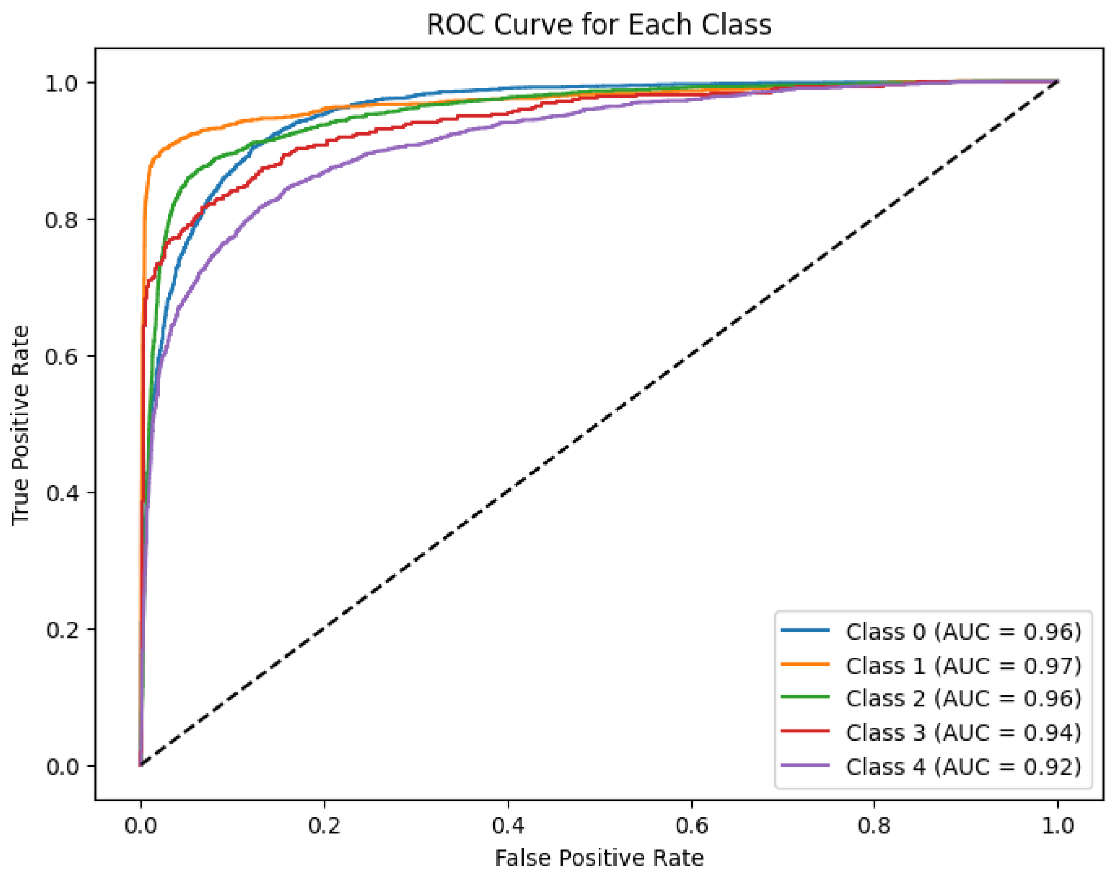
[ ]:
```
# Get predictions for test data
predictions = model.predict(test_dataset.batch(32))
predicted_classes = predictions.argmax(axis=1)
true_classes = test_labels

# 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
```

```python
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()
```

```
 51/275 [====>…] - 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
```

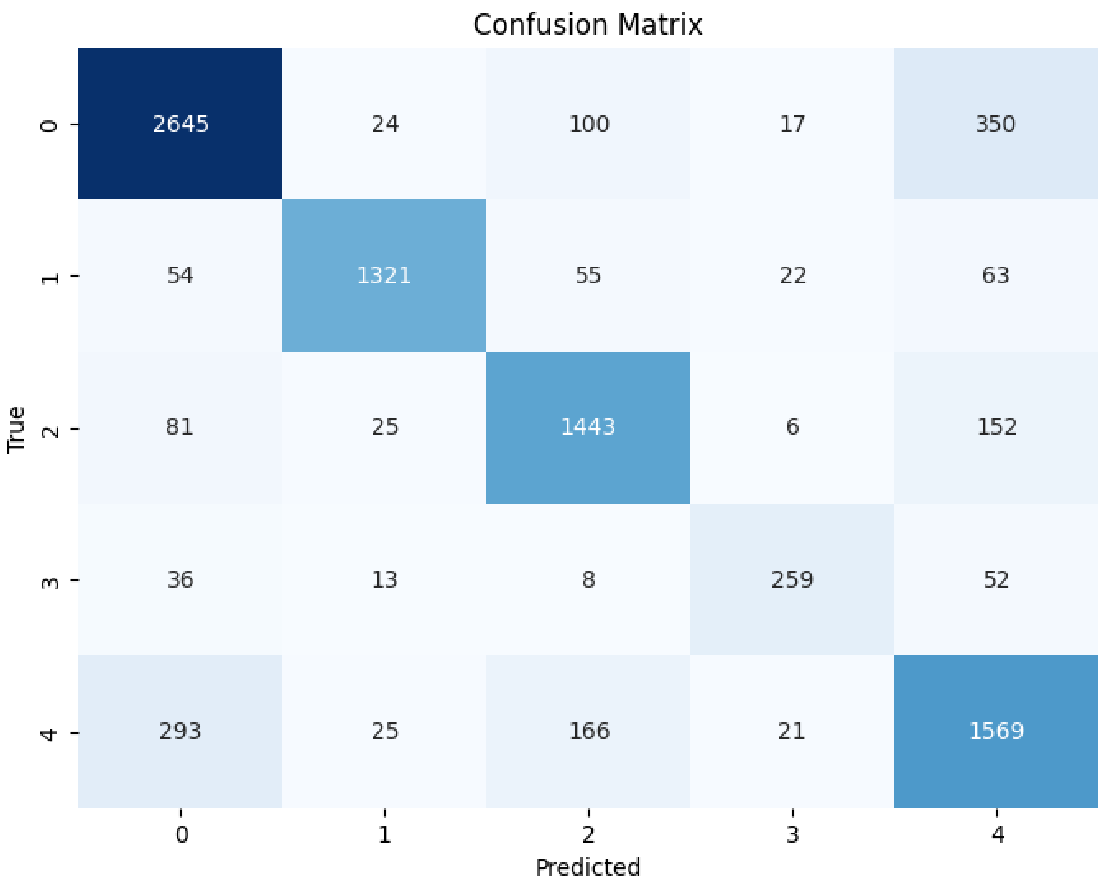


[ ]:
```
# Evaluate the model on test data
loss, accuracy = model.evaluate(test_dataset.batch(32))
print(f'Test Accuracy: {accuracy * 100:.2f}%')
print(f'Test Loss: {loss:.4f}')

# 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()
```

```python
# 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}')
```

```
/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)
```

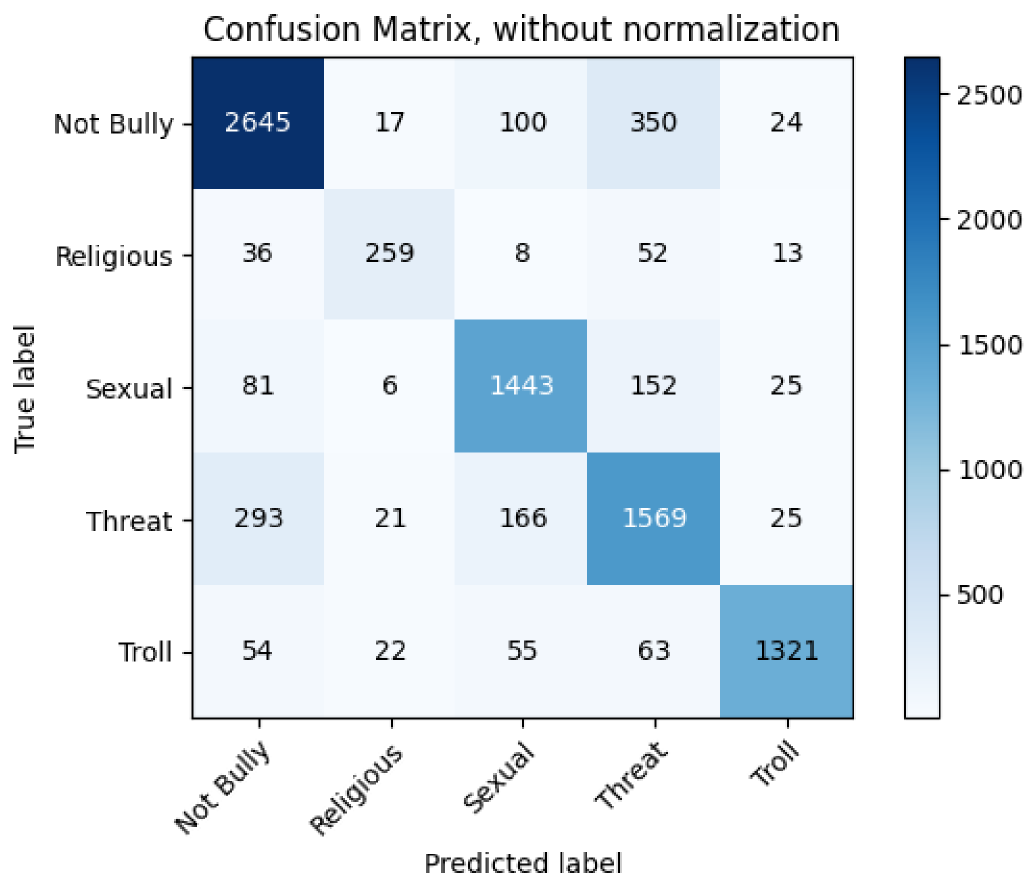
```
275/275 [==============================] - 1s 4ms/step - loss: 0.5483 -
accuracy: 0.8224
Test Accuracy: 82.24%
Test Loss: 0.5483
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.84      0.85      3136
           1       0.94      0.87      0.90      1515
           2       0.81      0.85      0.83      1707
           3       0.80      0.70      0.75       368
           4       0.72      0.76      0.74      2074

    accuracy                           0.82      8800
   macro avg       0.82      0.80      0.81      8800
weighted avg       0.83      0.82      0.82      8800

Confusion Matrix:
 [[2645   24  100   17  350]
 [  54 1321   55   22   63]
 [  81   25 1443    6  152]
 [  36   13    8  259   52]
 [ 293   25  166   21 1569]]
```




```
Precision: 0.8251
Recall: 0.8224
F1-score: 0.8233
```

[ ]:

```
Confusion matrix, without normalization
```



[ ]:
```python
# Calculate TP, FP, TN, FN for each class
for i, class_name in enumerate(class_names):
    tp = cm[i, i]
    fp = np.sum(cm[:, i]) - tp
    fn = np.sum(cm[i, :]) - tp
    tn = np.sum(cm) - tp - fp - fn

    print(f"\nClass: {class_name}")
    print(f"True Positives (TP): {tp}")
    print(f"False Positives (FP): {fp}")
    print(f"True Negatives (TN): {tn}")
    print(f"False Negatives (FN): {fn}")
```

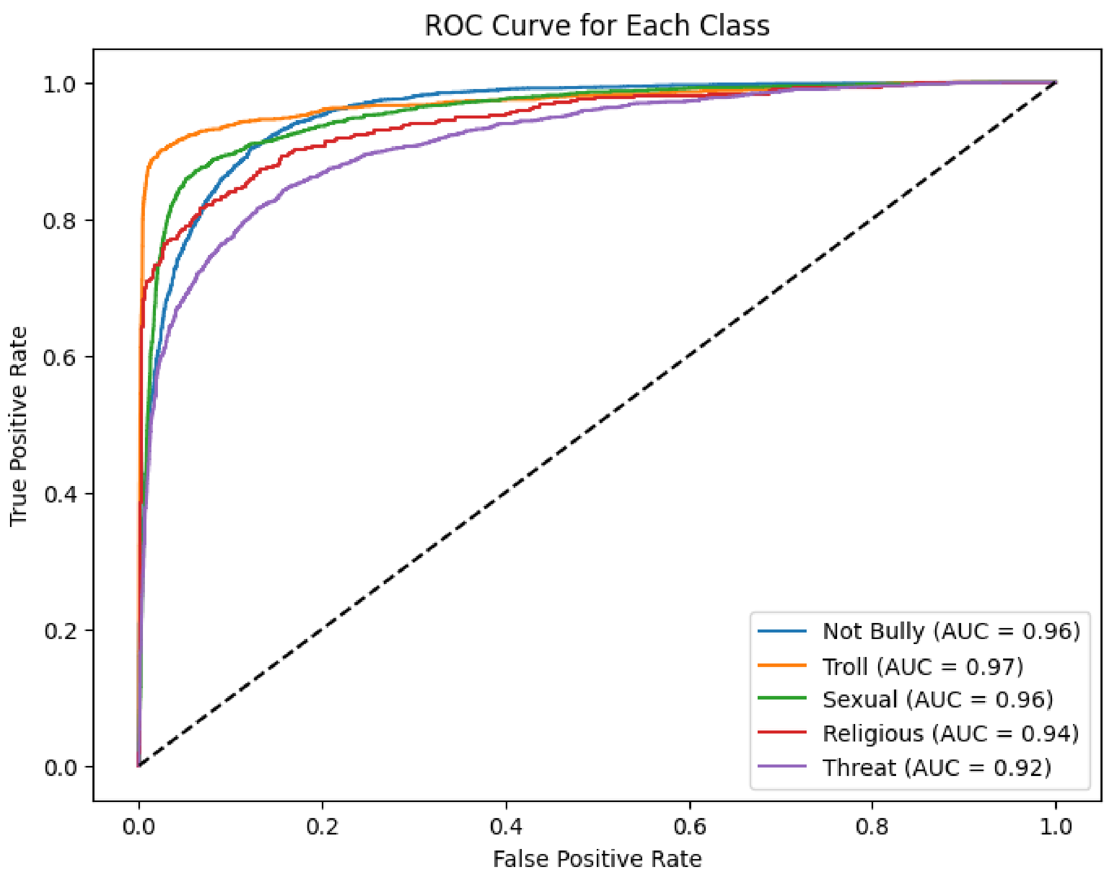
```
Class: Not Bully
True Positives (TP): 2645
False Positives (FP): 464
True Negatives (TN): 5200
False Negatives (FN): 491
```

```
Class: Troll
True Positives (TP): 259
False Positives (FP): 66
True Negatives (TN): 8366
False Negatives (FN): 109

Class: Sexual
True Positives (TP): 1443
False Positives (FP): 329
True Negatives (TN): 6764
False Negatives (FN): 264

Class: Religious
True Positives (TP): 1569
False Positives (FP): 617
True Negatives (TN): 6109
False Negatives (FN): 505

Class: Threat
True Positives (TP): 1321
False Positives (FP): 87
True Negatives (TN): 7198
False Negatives (FN): 194
```

[ ]:

```python
# Convert true classes 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_names[i]} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal reference line
plt.xlabel('False Positive Rate')
```

```python
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Each Class')
plt.legend(loc="lower right")
plt.show()
```



[ ]:

```python
```

[ ]:

```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: 1343
True Negatives: 2734
```

```
False Positives: 22
False Negatives: 57
```

[ ]:
```
# 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': LSTM_Model(tokenizer.vocab_size),
          'CNN': CNNmodel(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, callbacks=[early_stopping])
    histories[name] = history
```

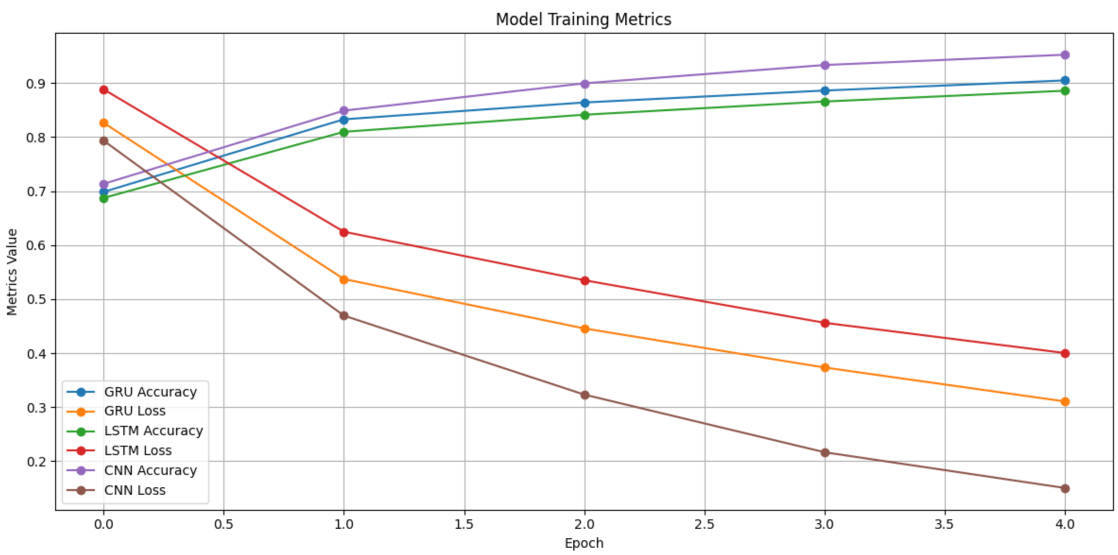
```
Training GRU model…
Epoch 1/5
1100/1100 [==============================] - ETA: 0s - loss: 0.8265 - accuracy:
0.6984

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 68s 55ms/step - loss: 0.8265 -
accuracy: 0.6984
Epoch 2/5
1099/1100 [============================>.] - ETA: 0s - loss: 0.5374 - accuracy:
0.8328

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 38s 35ms/step - loss: 0.5372 -
accuracy: 0.8328
Epoch 3/5
1100/1100 [==============================] - ETA: 0s - loss: 0.4458 - accuracy:
0.8640

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 37s 34ms/step - loss: 0.4458 -
accuracy: 0.8640
Epoch 4/5
```

```
1100/1100 [==============================] - ETA: 0s - loss: 0.3734 - accuracy:
0.8860

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 36s 33ms/step - loss: 0.3734 -
accuracy: 0.8860
Epoch 5/5
1099/1100 [============================>.] - ETA: 0s - loss: 0.3107 - accuracy:
0.9048

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 34s 31ms/step - loss: 0.3107 -
accuracy: 0.9048
Training LSTM model…
Epoch 1/5
1099/1100 [============================>.] - ETA: 0s - loss: 0.8885 - accuracy:
0.6871

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 63s 51ms/step - loss: 0.8884 -
accuracy: 0.6872
Epoch 2/5
1099/1100 [============================>.] - ETA: 0s - loss: 0.6247 - accuracy:
0.8097

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 29s 27ms/step - loss: 0.6248 -
accuracy: 0.8097
Epoch 3/5
1099/1100 [============================>.] - ETA: 0s - loss: 0.5350 - accuracy:
0.8412

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 30s 27ms/step - loss: 0.5350 -
accuracy: 0.8413
Epoch 4/5
1100/1100 [==============================] - ETA: 0s - loss: 0.4562 - accuracy:
0.8657

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 27s 25ms/step - loss: 0.4562 -
accuracy: 0.8657
```

```
Epoch 5/5
1100/1100 [==============================] - ETA: 0s - loss: 0.4003 - accuracy:
0.8857

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 28s 25ms/step - loss: 0.4003 -
accuracy: 0.8857
Training CNN model…
Epoch 1/5
1100/1100 [==============================] - ETA: 0s - loss: 0.7935 - accuracy:
0.7133

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 39s 33ms/step - loss: 0.7935 -
accuracy: 0.7133
Epoch 2/5
1094/1100 [============================>.] - ETA: 0s - loss: 0.4704 - accuracy:
0.8485

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 12s 11ms/step - loss: 0.4695 -
accuracy: 0.8488
Epoch 3/5
1100/1100 [==============================] - ETA: 0s - loss: 0.3234 - accuracy:
0.8994

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 12s 11ms/step - loss: 0.3234 -
accuracy: 0.8994
Epoch 4/5
1099/1100 [============================>.] - ETA: 0s - loss: 0.2167 - accuracy:
0.9333

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy

1100/1100 [==============================] - 10s 9ms/step - loss: 0.2167 -
accuracy: 0.9333
Epoch 5/5
1096/1100 [============================>.] - ETA: 0s - loss: 0.1505 - accuracy:
0.9523

WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,accuracy
```

```
1100/1100 [==============================] - 10s 9ms/step - loss: 0.1505 -
accuracy: 0.9524
```

[ ]:
```python
# Plot training & validation accuracy and loss values in a single plot
plt.figure(figsize=(12, 6))

for name, history in histories.items():
  plt.plot(history.history['accuracy'], label=f'{name} Accuracy', marker='o')
  plt.plot(history.history['loss'], label=f'{name} Loss', marker='o')
  plt.title('Model Training Metrics')
  plt.xlabel('Epoch')
  plt.ylabel('Metrics Value')
  plt.legend()
  plt.grid(True)

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



[ ]:
```python
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 [==============================] - 13s 10ms/step
275/275 [==============================] - 3s 11ms/step
1100/1100 [==============================] - 11s 8ms/step
275/275 [==============================] - 3s 10ms/step
```

```
1100/1100 [==============================] - 2s 2ms/step
275/275 [==============================] - 1s 2ms/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}%")
```

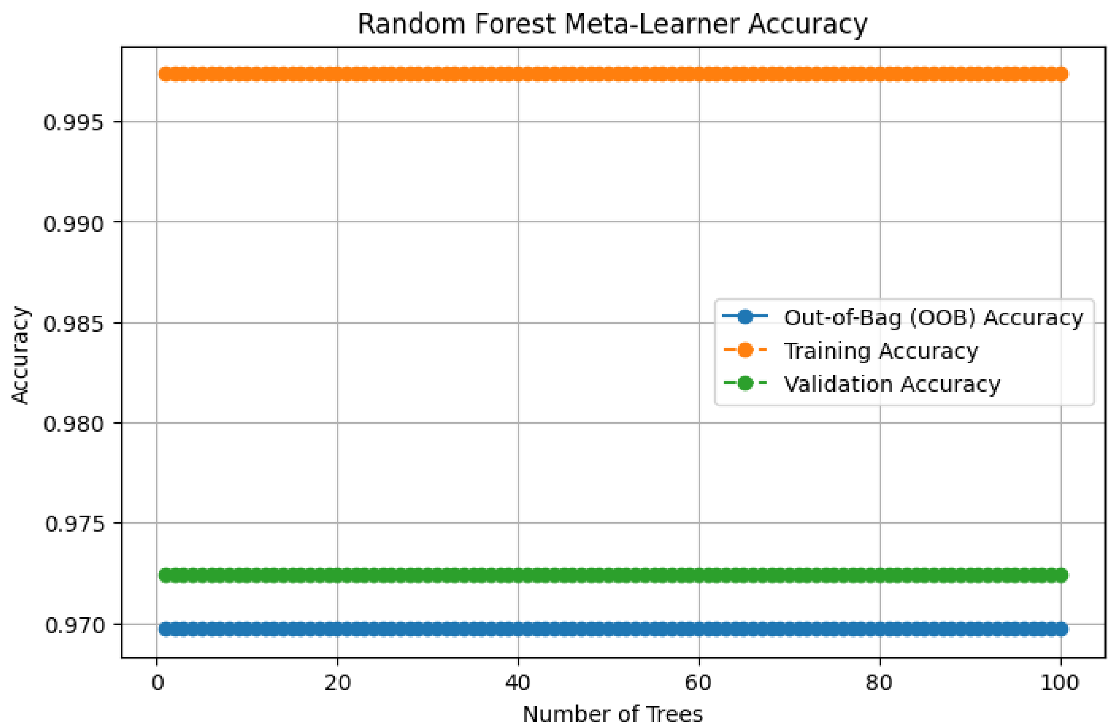
```
Accuracy of Random Forest meta-learner: 97.24%
```

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

# Predictions on the training set
rf_train_predictions = rf_meta_learner.predict(stacked_train)
train_accuracy = accuracy_score(train_labels_train, rf_train_predictions)
print(f"Training Accuracy of Random Forest meta-learner: {train_accuracy * 100:.
↪2f}%")

# Predictions on the validation set
rf_val_predictions = rf_meta_learner.predict(stacked_val)
val_accuracy = accuracy_score(train_labels_val, rf_val_predictions)
print(f"Validation Accuracy of Random Forest meta-learner: {val_accuracy * 100:.
↪2f}%")
```

```
Training Accuracy of Random Forest meta-learner: 99.74%
Validation Accuracy of Random Forest meta-learner: 97.24%
```

[ ]:

```
# Initialize the Random Forest model with oob_score=True
rf_meta_learner = RandomForestClassifier(n_estimators=100, random_state=42,␣
 ↪oob_score=True)
rf_meta_learner.fit(stacked_train, train_labels_train)

# Access the overall OOB score
oob_score = rf_meta_learner.oob_score_

# Predictions on the validation set
rf_val_predictions = rf_meta_learner.predict(stacked_val)
val_accuracy = accuracy_score(train_labels_val, rf_val_predictions)

# Plotting training and validation accuracies
plt.figure(figsize=(8, 5))
n_estimators = len(rf_meta_learner.estimators_)
epochs = range(1, n_estimators + 1)

plt.plot(epochs, [oob_score] * n_estimators, label='Out-of-Bag (OOB) Accuracy',␣
 ↪marker='o')
plt.plot(epochs, [train_accuracy] * n_estimators, label='Training Accuracy',␣
 ↪linestyle='--', marker='o')
plt.plot(epochs, [val_accuracy] * n_estimators, label='Validation Accuracy',␣
 ↪linestyle='--', marker='o')

plt.title('Random Forest Meta-Learner Accuracy')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



[ ]:
```python
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}")
```

```
Precision: 0.9808
Recall: 0.9724
F1-score: 0.9766
```

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

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

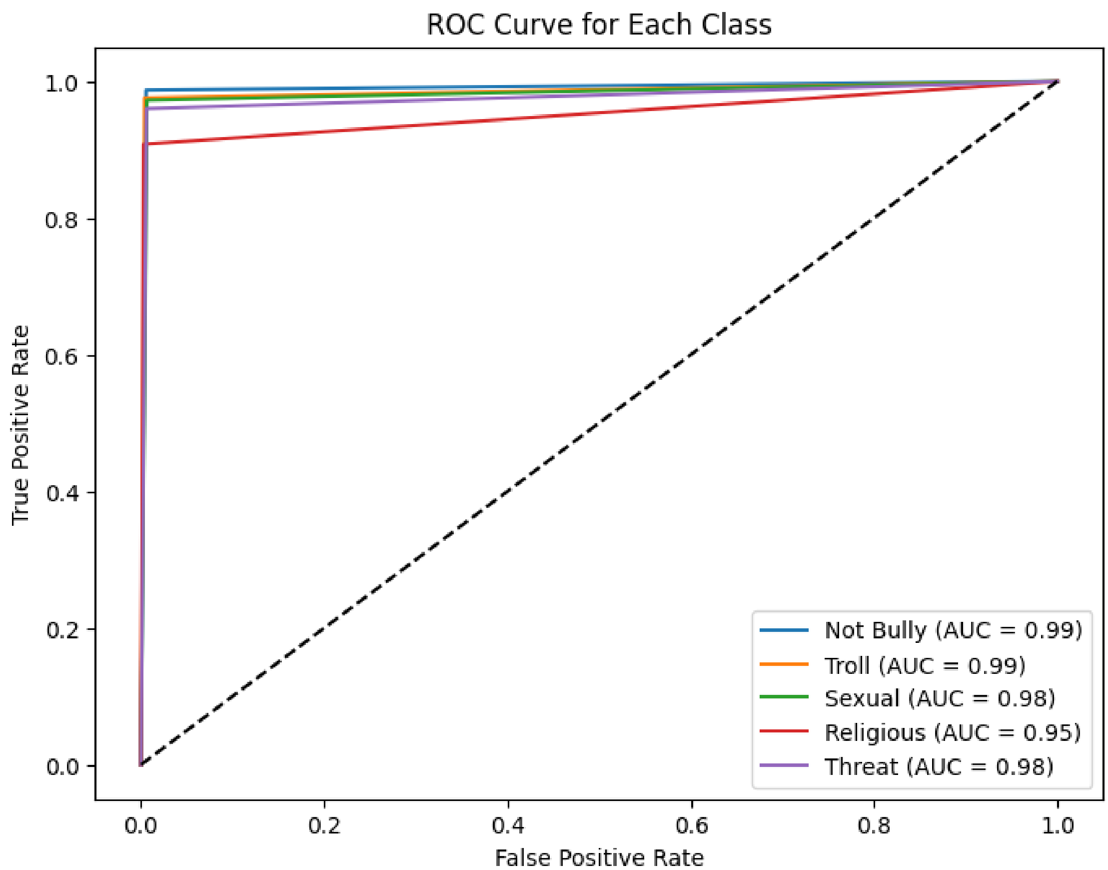
[ ]:
```python
from joblib import dump

# 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)
```

[ ]: ['/content/drive/MyDrive/Bully/rf_meta_learner.joblib']

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

[ ]:
```
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])

# Assuming class_names is a list of your class names
num_classes = ["Not Bully", "Troll", "Sexual", "Religious", "Threat"]

# 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'{num_classes[i]} (AUC = {roc_auc[i]:.2f})')

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

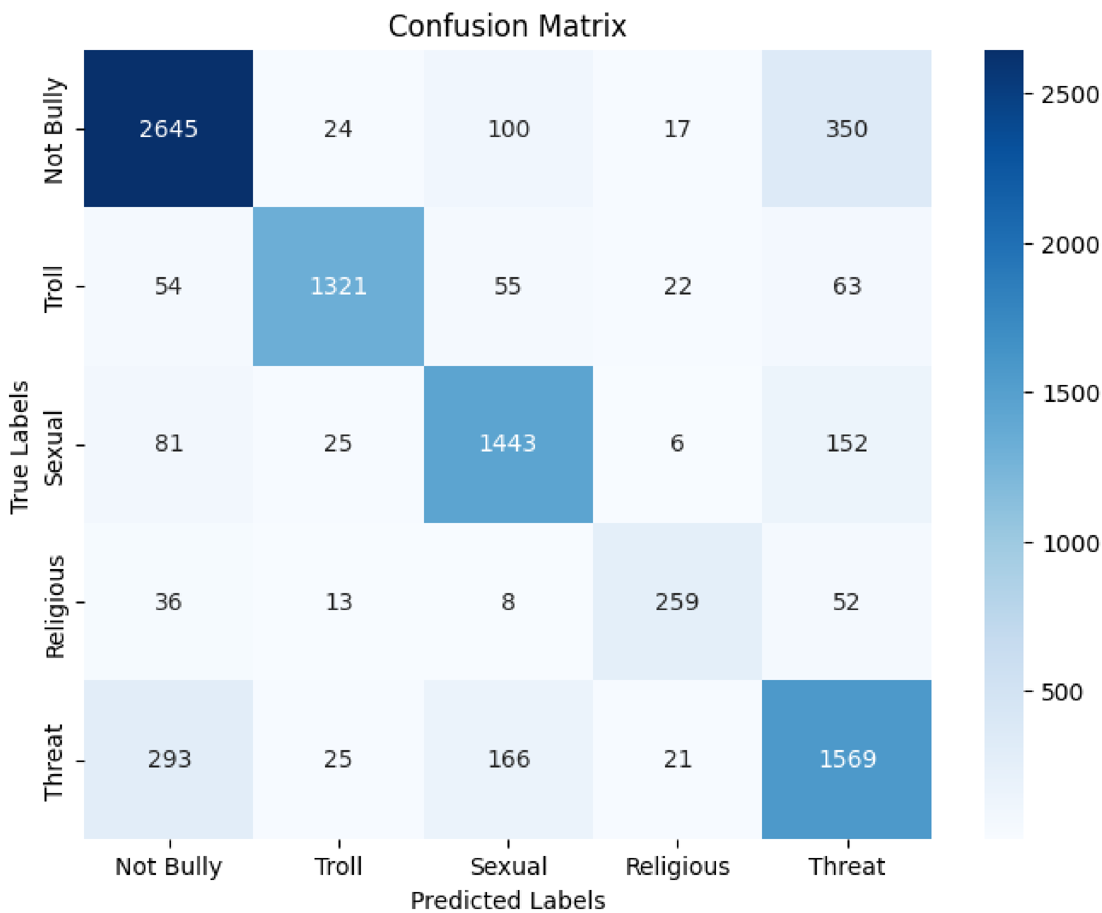


[ ]:
```python
from sklearn.preprocessing import LabelEncoder

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)
```

```
Shapes - True Labels: (7040,) Predictions: (7040,)
Confusion Matrix:
 [[2380    2    6    1   13]
 [  20 1205    8    2    0]
 [  13    8 1405    4   15]
 [  16    5    1  256    4]
 [  28    7   21   11 1609]]
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.99      0.98      2402
           1       0.98      0.98      0.98      1235
           2       0.98      0.97      0.97      1445
           3       0.93      0.91      0.92       282
           4       0.98      0.96      0.97      1676

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

[ ]:
```
num_classes = ["Not Bully", "Troll", "Sexual", "Religious", "Threat"]
# Plotting the confusion matrix with class labels
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',␣
↪xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

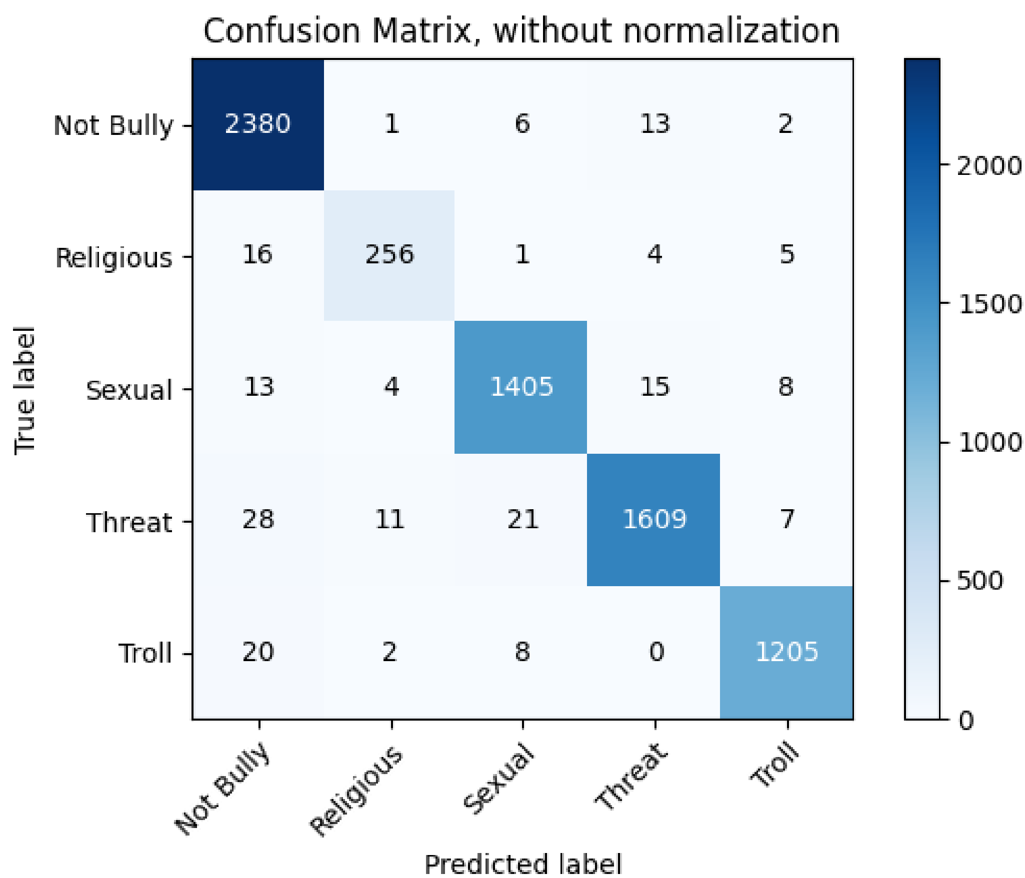


[ ]:
```python
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.utils.multiclass import unique_labels

# Convert predicted classes to class names
class_names = ["Not Bully", "Troll", "Sexual", "Religious", "Threat"]
# Round the predicted probabilities to get class labels
rounded_predictions = np.argmax(rf_meta_predictions, axis=1)

# Convert predicted classes to class names
predicted_class_names = [class_names[i] for i in rounded_predictions]

# Convert true classes to class names
true_class_names = [class_names[i] for i in train_labels_val]

# Confusion Matrix
```

```python
cm = confusion_matrix(true_class_names, predicted_class_names)

# Plot Confusion Matrix
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized Confusion Matrix'
        else:
            title = 'Confusion Matrix'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = unique_labels(y_true, y_pred)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(len(classes)),
           yticks=np.arange(len(classes)),
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(len(classes)):
        for j in range(len(classes)):
```

```
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


# Plot non-normalized confusion matrix
plot_confusion_matrix(true_class_names, predicted_class_names,␣
 ↪classes=class_names,
                      title='Confusion Matrix, without normalization')

plt.show()
```

```
Confusion matrix, without normalization
```




```
[ ]: # Calculate TP, FP, TN, FN for each class
     for i, class_name in enumerate(class_names):
         tp = cm[i, i]
```

```
    fp = np.sum(cm[:, i]) - tp
    fn = np.sum(cm[i, :]) - tp
    tn = np.sum(cm) - tp - fp - fn

    print(f"\nClass: {class_name}")
    print(f"True Positives (TP): {tp}")
    print(f"False Positives (FP): {fp}")
    print(f"True Negatives (TN): {tn}")
    print(f"False Negatives (FN): {fn}")
```

```
Class: Not Bully
True Positives (TP): 2380
False Positives (FP): 77
True Negatives (TN): 4561
False Negatives (FN): 22

Class: Troll
True Positives (TP): 256
False Positives (FP): 18
True Negatives (TN): 6740
False Negatives (FN): 26

Class: Sexual
True Positives (TP): 1405
False Positives (FP): 36
True Negatives (TN): 5559
False Negatives (FN): 40

Class: Religious
True Positives (TP): 1609
False Positives (FP): 32
True Negatives (TN): 5332
False Negatives (FN): 67

Class: Threat
True Positives (TP): 1205
False Positives (FP): 22
True Negatives (TN): 5783
False Negatives (FN): 30
```

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

```
Requirement already satisfied: lime in /usr/local/lib/python3.10/dist-packages
(0.2.0.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from lime) (3.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
```

```
(from lime) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from lime) (1.11.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from lime) (4.66.1)
Requirement already satisfied: scikit-learn>=0.18 in
/usr/local/lib/python3.10/dist-packages (from lime) (1.2.2)
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: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (23.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.18->lime) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->lime) (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) (4.46.0)
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: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib->lime) (1.16.0)
```

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

Lime

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

```
Collecting lime
```

```
  Downloading lime-0.2.0.1.tar.gz (275 kB)
                           275.7/275.7

kB 4.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from lime) (3.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from lime) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from lime) (1.11.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from lime) (4.66.1)
Requirement already satisfied: scikit-learn>=0.18 in
/usr/local/lib/python3.10/dist-packages (from lime) (1.2.2)
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: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (23.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.18->lime) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->lime) (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: 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=c0ee1710a973e9061e35f162403b78534ffeaf05e17ef7eeccd71598f95569c4
  Stored in directory: /root/.cache/pip/wheels/fd/a2/af/9ac0a1a85a27f314a06b39e1
f492bee1547d52549a4606ed89
Successfully built lime
Installing collected packages: lime
Successfully installed lime-0.2.0.1
```

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

[ ]:
```python
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")
```

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

[ ]:
```python
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
```

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

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

[ ]:
```python
# Define a sample text for explanation
sample_text = "                                  "

# Generate explanations for the sample text
exp = explainer.explain_instance(sample_text, predict_proba, num_features=10)
```

```
157/157 [==============================] - 3s 22ms/step
```

```
[ ]: import random
```

```
[ ]: test_dataset = test_dataset.take(5)

     # Extract the text samples from the dataset
     test_texts = ["".join(map(str, sample[0]['sequences'].numpy())) for sample in␣
      ↪test_dataset]


     # Randomly select 2 samples from the list of texts
     selected_samples = random.sample(test_texts, k=2) if len(test_texts) > 2 else␣
      ↪test_texts
```

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

```
[ ]: from IPython.display import display, HTML

     for i, sample in enumerate(test_dataset.take(5)):
         input_data, sample_text = sample
         sample_text = ''.join([str(text_element) for text_element in np.
      ↪nditer(sample_text)])
         explanation = explainer.explain_instance(sample_text, predict_proba,␣
      ↪num_features=10)

         # Display prediction probabilities
         display(HTML("<h3>Prediction Probabilities</h3>"))
         for label, prob in zip(explanation.class_names, explanation.predict_proba):
             print(f"{prob:.2f} - {label}")

         # Display highlighted words with scores
         display(HTML("<h3>Text with Highlighted Words</h3>"))
         words_and_scores = explanation.as_list()
         highlighted_text = ' '.join([f'<span style="background-color: rgba(0, 255,␣
      ↪0, {score:.2f})">{word}</span>' for word, score in words_and_scores])
         display(HTML(highlighted_text))

         print()
```

```
157/157 [==============================] - 2s 15ms/step
```

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

```
0.74 - not bully
0.03 - troll
0.05 - sexual
```

```
0.03 - religious
0.15 - threat

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 15ms/step

<IPython.core.display.HTML object>

0.74 - not bully
0.03 - troll
0.05 - sexual
0.03 - religious
0.15 - threat

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 11ms/step

<IPython.core.display.HTML object>

0.74 - not bully
0.03 - troll
0.05 - sexual
0.03 - religious
0.15 - threat

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 11ms/step

<IPython.core.display.HTML object>

0.74 - not bully
0.03 - troll
0.05 - sexual
0.03 - religious
0.15 - threat

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 15ms/step

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

```
0.74 - not bully
0.03 - troll
0.05 - sexual
0.03 - religious
0.15 - threat
```

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

```
<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()
```

```
157/157 [==============================] - 3s 8ms/step
Explanation for sample 1:

<IPython.core.display.HTML object>

157/157 [==============================] - 1s 8ms/step
Explanation for sample 2:

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 12ms/step
Explanation for sample 3:

<IPython.core.display.HTML object>

157/157 [==============================] - 2s 16ms/step
Explanation for sample 4:

<IPython.core.display.HTML object>

157/157 [==============================] - 1s 8ms/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 = 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
```

```
Collecting shap
  Downloading shap-0.43.0-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64
.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (532 kB)
                           532.9/532.9

kB 5.9 MB/s eta 0:00:00
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) (1.11.3)
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) (0.56.4)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-
packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba->shap) (0.39.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from numba->shap) (67.7.2)
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)
Installing collected packages: slicer, shap
Successfully installed shap-0.43.0 slicer-0.0.7
```

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

[ ]:
```
# Find the Bidirectional layer in the model
bidirectional_layer = None
for layer in model.layers:
    if isinstance(layer, Bidirectional):
        bidirectional_layer = layer
        break

if bidirectional_layer is None:
    raise ValueError("No Bidirectional layer found in the model")

# Assuming the Bidirectional layer wraps a GRU layer
gru_layer = bidirectional_layer.layer  # Access the internal GRU layer

# Create an intermediate model up to the GRU layer
intermediate_layer_model = Model(inputs=model.input, outputs=gru_layer.output)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-121-2d200f04bbe1> in <cell line: 8>()
```

```
      7
      8 if bidirectional_layer is None:
----> 9     raise ValueError("No Bidirectional layer found in the model")
     10
     11 # Assuming the Bidirectional layer wraps a GRU layer

ValueError: No Bidirectional layer found in the model
```

[ ]:
```
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})
```

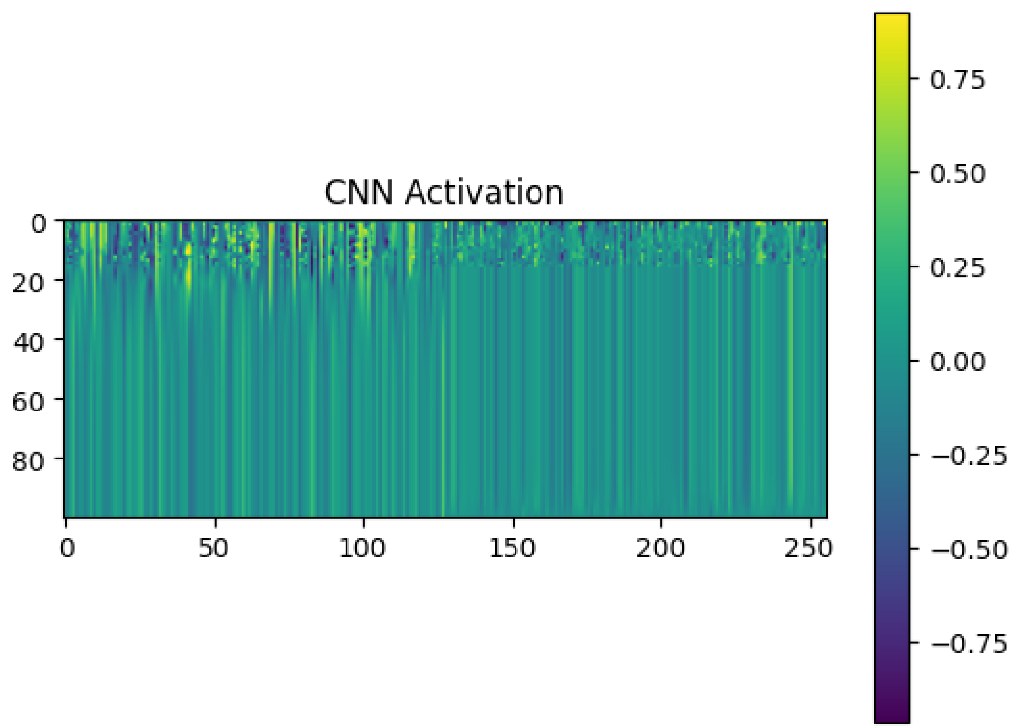
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-117-ed32c24c9714> in <cell line: 1>()
----> 1 cnn_output = intermediate_layer_model.predict({'sequences':␣
↪sequences_input})
      2 lstm_output = intermediate_layer_model.predict({'sequences':␣
↪sequences_input})
      3 ensemble_output = intermediate_layer_model.predict({'sequences':␣
↪sequences_input})

NameError: name 'intermediate_layer_model' is not defined
```

[ ]:
```
# 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()
```

```python
plt.title('CNN Activation')
plt.show()
```

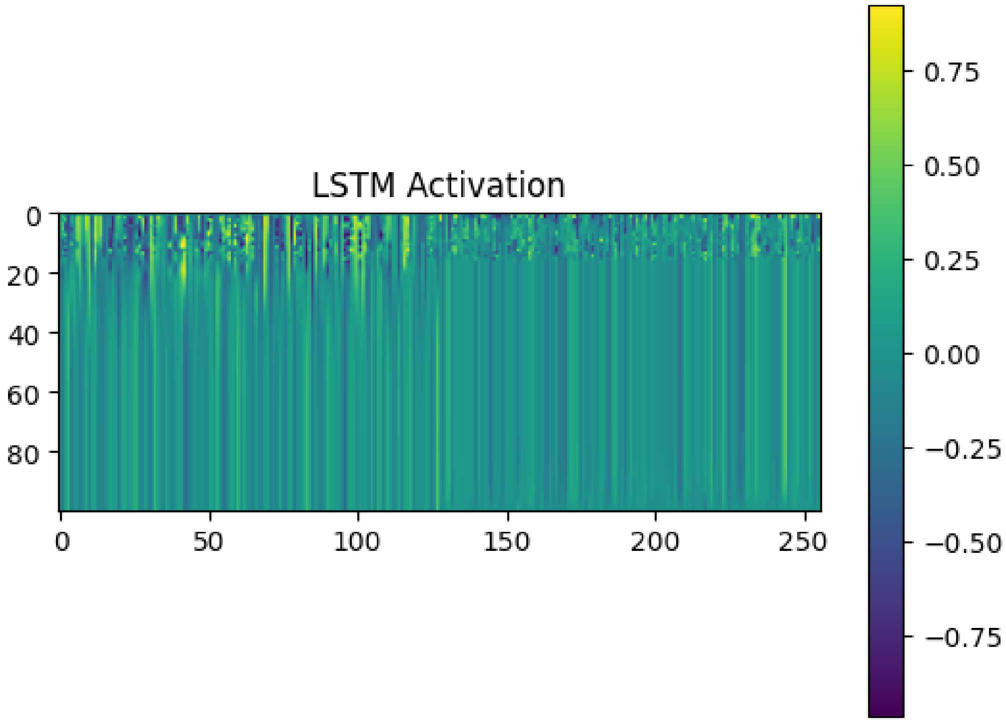
```
WARNING:tensorflow:6 out of the last 790 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x79d7b510aa70>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing 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
```



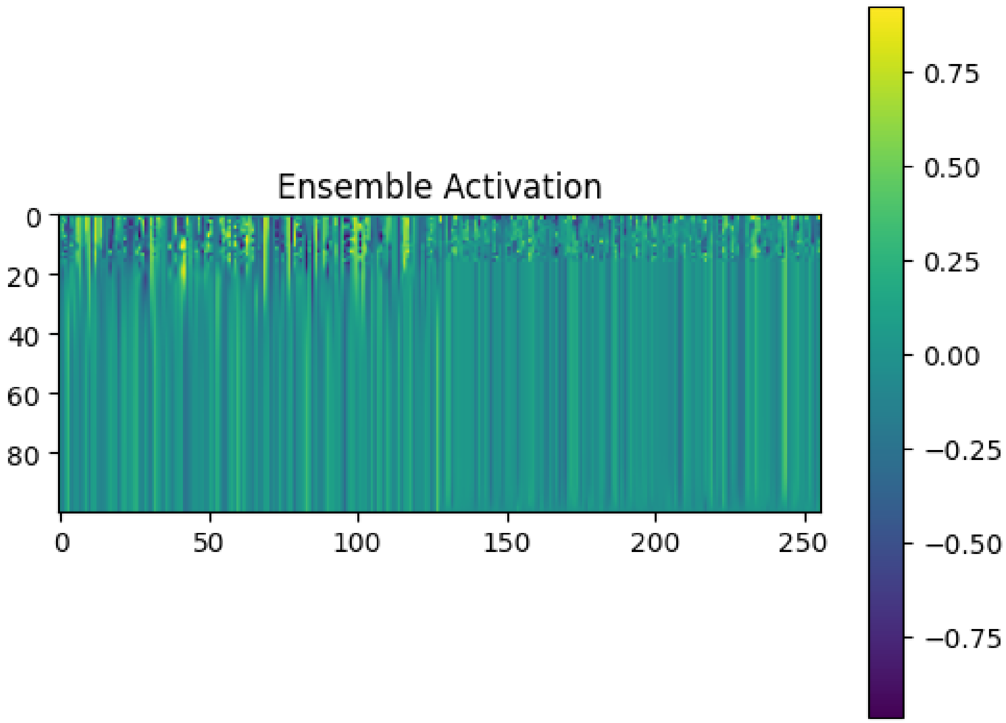
[ ]:
```python
# Print layers to identify the desired layer's index
for i, layer in enumerate(lstm_model.layers):
    print(i, layer.name, layer.__class__.__name__)

# 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
```




```
[ ]:
```