Sentiment-analysis-for-social-media-using-BERT-&-XGBoost

March 26, 2025

0.1 Module 1

#Sentiment analysis for social media using BERT Model and XGBoost

```
Mind map for this project
```

##Mount of Google Drive ###For import data files

1 Data collection process

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

new_df = pd.read_csv('/content/final_balanced_sentiment_dataset.csv')
print(new_df.shape)
```
```python
display(new_df.head())
```

```
(32273, 16)
     tweet_id airline_sentimentairline_sentiment_confidence  \
0  5.703061e+17  neutral    1.0000
1  5.703011e+17  positive   0.3486
2  5.703011e+17  neutral    0.6837
3  5.703010e+17  negative   1.0000
4  5.703008e+17  negative   1.0000

 negativereasonnegativereason_confidence     airline \
0     NaN   NaN   Virgin America
1     NaN   0.0000     Virgin America 2 NaN   NaN   Virgin
America
3     Bad Flight  0.7033     Virgin America
4     Can't Tell  1.0000     Virgin America

 airline_sentiment_gold    name negativereason_goldretweet_count \
0                    NaN    cairdin                NaN            0.0
1                    NaN   jnardino                NaN            0.0
2                    NaN yvonnalynn                NaN            0.0
3                    NaN   jnardino                NaN            0.0
```

```
4                    NaN   jnardino                NaN             0.0

                                              text tweet_coord \
0                                        What said        NaN

1  plus youve added commercials to the experience…  NaN

2  I didnt today Must mean I need to take another…  NaN

3  its really aggressive to blast obnoxious enter…  NaN

4  and its a really big bad thing about it This i…  NaN

             tweet_created tweet_location           user_timezone\
0  2015-02-24 11:35:52 -0800        NaN        Eastern Time (US &
   Canada)
1  2015-02-24 11:15:59 -0800        NaN        Pacific Time (US &
   Canada)
2  2015-02-24 11:15:48 -0800  Lets Play        Central Time (US &
   Canada)
3  2015-02-24 11:15:36 -0800        NaN        Pacific Time (US &
   Canada)
4  2015-02-24 11:14:45 -0800        NaN        Pacific Time (US &
   Canada)
  sentiment
0      irony
1      happy
2      happy
3      happy
4      sad
```

[4]:

```
!pip install emoji
```

```
Collecting emoji
  Downloading emoji-2.14.1-py3-none-any.whl.metadata (5.7 kB)
Downloading emoji-2.14.1-py3-none-any.whl (590 kB)
    590.6/590.6 kB
26.7 MB/s eta 0:00:00
Installing collected packages: emoji
Successfully installed emoji-2.14.1
```

[ ]:

```
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-
packages (3.2.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-
packages
(from nltk) (1.17.0)
```

[8]:

```python
import re
import nltk
from nltk.stem import WordNetLemmatizer
import emoji
nltk.download("wordnet")
#nltk.download()
'''import nltk
nltk.data.path.append('/kaggle/working/nltk_data')
nltk.download('wordnet', download_dir='/kaggle/working/nltk_data')'''


lemmatizer = WordNetLemmatizer()

def clean_text(text):
    # Ensure input is a string
    text = str(text)

    # Remove URLs
    text = re.sub(r'http\S+', '', text)

    # Remove mentions (@user)
    text = re.sub(r'@\S+', '', text)

    # Remove hashtags (#hashtag)
    text = re.sub(r'#\S+', '', text)

    # Remove special characters and numbers (except '!')
    text = re.sub(r'[^a-zA-Z!\s]', '', text)

    # Replace multiple '!' with a single '!'
    text = re.sub(r'!+', '!', text)
```

```
    # Remove single characters (but keep meaningful ones like 'I')
    text = re.sub(r'\s+[b-df-hj-np-tv-zB-DF-HJ-NP-TV-Z]\s+', ' ', text)   #␣
↪Keeps 'I'

    # Remove single characters at the start of words
    text = re.sub(r'\^[a-zA-Z]\s+', ' ', text)

    # Remove extra whitespace
    text = ' '.join(text.split())

    text = re.sub(r'[^a-zA-Z0-9 ]', '', text)

    # Remove prefixed 'b' (byte-string artifacts)
    text = re.sub(r'^b\s+', '', text)

    # Convert FULLY CAPITALIZED words to sentence case
    words = text.split()
    processed_words = [word.capitalize() if word.isupper() else word for word␣
↪in words]
    text = ' '.join(processed_words)
    text = " ".join([lemmatizer.lemmatize(word) for word in text.split()])  #␣
↪Lemmatize words
    text = emoji.demojize(text)  # Convert emojis to text
    text = text.strip()

    return text
new_df['text'] = new_df['text'].apply(clean_text)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data] /kaggle/working/nltk_data…
[nltk_data]Package wordnet is already up-to-date!
```

[6]:
```
new_df.head(10)
```

[6]:
```
       tweet_id airline_sentimentairline_sentiment_confidence  \
0  5.703061e+17         neutral                        1.0000
1  5.703011e+17        positive                        0.3486
2  5.703011e+17         neutral                        0.6837
3  5.703010e+17        negative                        1.0000
4  5.703008e+17        negative                        1.0000
5  5.703008e+17        negative                        1.0000
6  5.703006e+17        positive                        0.6745
7  5.703002e+17         neutral                        0.6340
8  5.703000e+17        positive                        0.6559
9  5.702955e+17        positive                        1.0000

  negativereasonnegativereason_confidence     airline  \
```

```
0           NaN                      NaN Virgin America
1           NaN                   0.0000 Virgin America
2           NaN                      NaN Virgin America
3     Bad Flight                  0.7033 Virgin America
4     Can't Tell                  1.0000 Virgin America
5     Can't Tell                  0.6842 Virgin America
6           NaN                   0.0000 Virgin America
7           NaN                      NaN Virgin America
8           NaN                      NaN Virgin America
9           NaN                      NaN Virgin America

                                                text tweet_coord  \
0                                          What said         NaN
1  plus youve added commercial to the experience …  NaN
2  I didnt today Must mean I need to take another…  NaN
3  it really aggressive to blast obnoxious entert…  NaN
4  and it a really big bad thing about it This is…  NaN
5  seriously would pay a flight for seat that did…  NaN
6  yes nearly every time I fly Vx this ear worm w…  NaN
7  Really missed a prime opportunity for Men With…  NaN
8  Well I didntbut Now I Do This is heartbreaking      NaN
9  it wa amazing and arrived an hour early Youre …  NaN
0      irony
1      happy
2      happy
3      happy
4      sad
5      sad
6      happy
7      happy
8      sad
9      happy
```

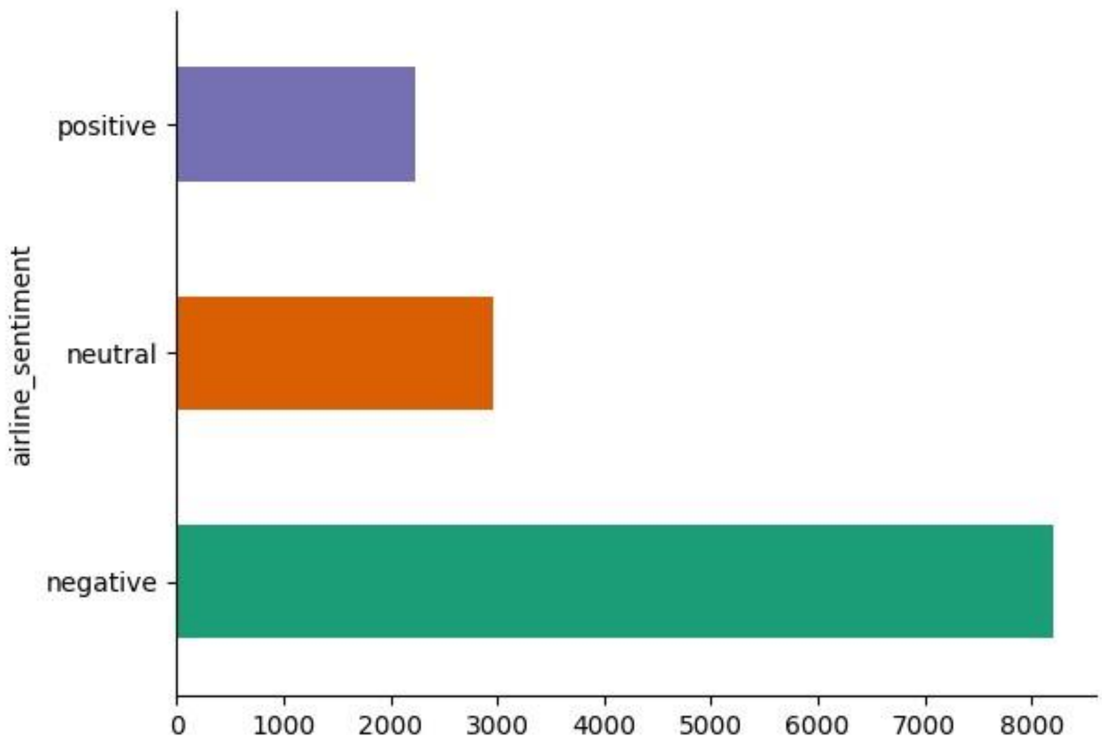
[9]:
```
all_text = ''.join(new_df['text']) # Merge all text into
one string chars = sorted(set(all_text)) # Extract unique
characters vocab_size = len(chars)
```

```
print(''.join(chars)) # Print all unique characters
print(vocab_size) # Print number of unique characters
```

```
 ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
53
```

[10]:

```
 from matplotlib import pyplot as plt import seaborn as sns
new_df.groupby('airline_sentiment').size().plot(kind='barh',
color=sns.palettes.
 ↪mpl_palette('Dark2'))
plt.gca().spines[['top',
'right',]].set_visible(False)
```

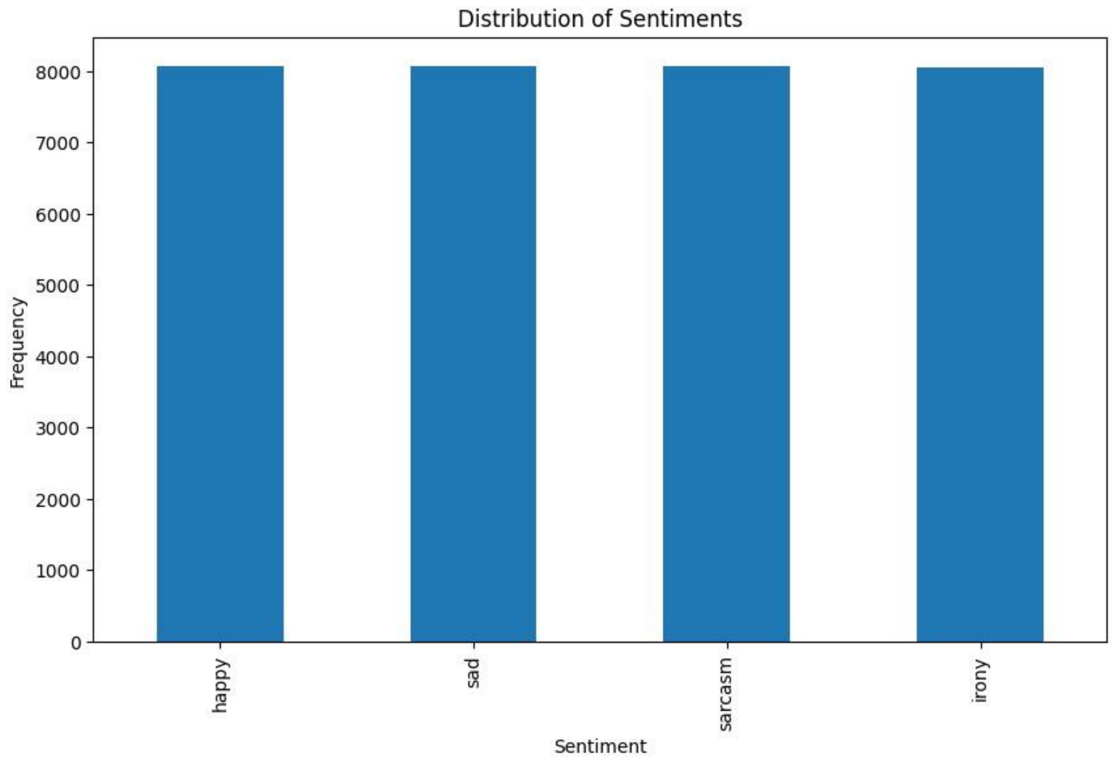


[11]:

```
# Drop rows with missing values in the 'text' column.
new_df.dropna(subset=['text'], inplace=True)

# Examine the length of the text data
new_df['text_length'] =
new_df['text'].apply(len) print("\nText
Length Statistics:") print("Average:",
new_df['text_length'].mean())
print("Minimum:",
new_df['text_length'].min())
print("Maximum:",
new_df['text_length'].max())
```

```
# Visualize the distribution of
sentiments import matplotlib.pyplot as
plt

plt.figure(figsize=(10, 6))
new_df['sentiment'].value_counts().plot(kind
='bar') plt.title('Distribution of
Sentiments') plt.xlabel('Sentiment')
plt.ylabel('Frequency') plt.show()
```

```
Text Length Statistics:
Average: 60.77299910141605
Minimum: 3
Maximum: 171
```



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

[ ]:
```
#!pip install nltk
```

[12]:
```
new_df.shape
```

[12]: (32273, 17)

[13]:
```
new_df.describe()
```

[13]:

	tweet_id	airline_sentiment_confidence	negativereason_confidence \
count	3.105100e+04	13387.000000	9455.000000
mean	5.698392e+17	0.898511	0.625692
std	7.456638e+14	0.164014	0.330754
min	5.675883e+17	0.335000	0.000000
25%	5.696184e+17	0.690700	0.357850
50%	5.703106e+17	1.000000	0.668400
75%	5.703106e+17	1.000000	1.000000
max	5.703106e+17	1.000000	1.000000

	retweet_count	text_length
count	13387.000000	32273.000000
mean	0.084858	60.772999
std	0.771892	34.572291
min	0.000000	3.000000
25%	0.000000	37.000000
50%	0.000000	46.000000
75%	0.000000	83.000000
max	44.000000	171.000000

[14]:
```
new_df.isnull().sum()
```

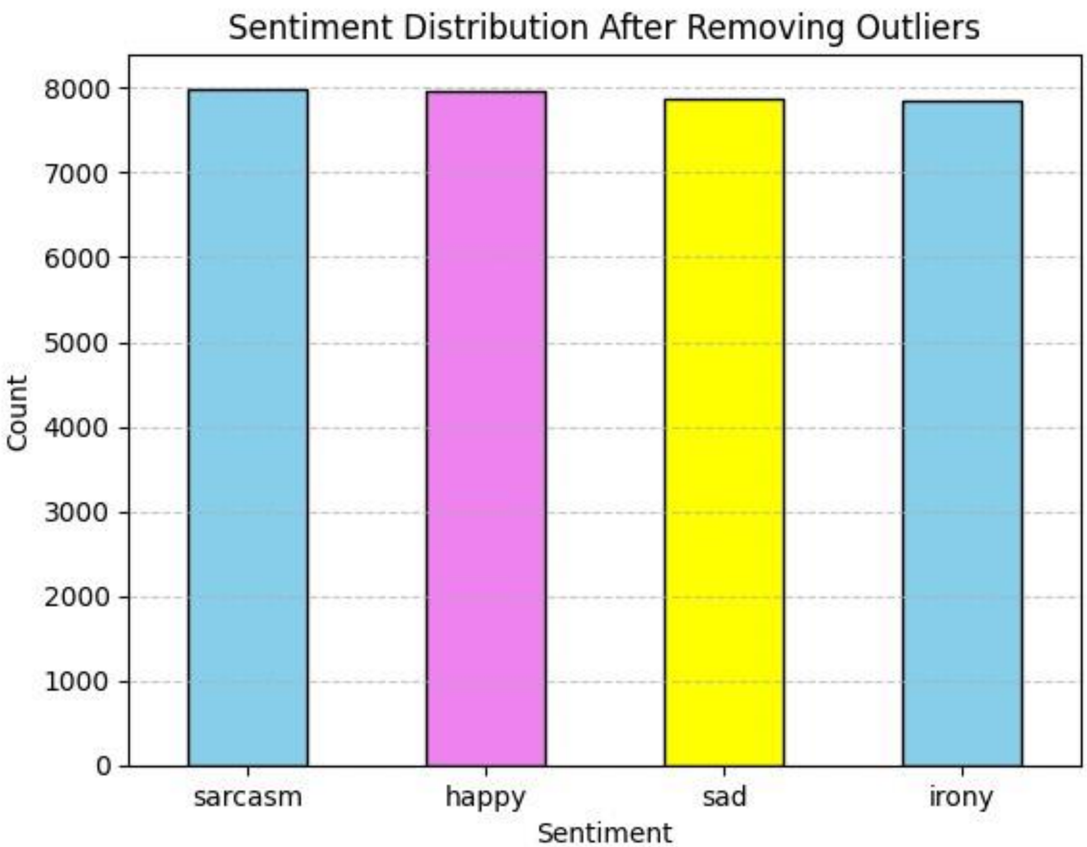
[14]:
```
tweet_id                         1222
airline_sentiment               18886
airline_sentiment_confidence    18886
negativereason                  24073
negativereason_confidence       22818
airline                         18886
airline_sentiment_gold          32235
name                            18886
negativereason_gold             32243
retweet_count                   18886
text                                0
tweet_coord                     31340
tweet_created                   18886
tweet_location                  23191
user_timezone                   23304
sentiment                           0
text_length                         0
```

```
dtype: int64
```

[15]:
```python
import pandas as pd

# Assuming new_df is already loaded
new_df['text_length'] = new_df['text'].str.len()

# Determine outlier limits (e.g., 1st and 99th percentile)
low_limit, high_limit = new_df['text_length'].quantile([0.01, 0.99])

# Filter data within limits
filtered_df = new_df[(new_df['text_length'] >= low_limit) &
 ↪(new_df['text_length'] <= high_limit)]

# Drop the extra column after filtering
filtered_df = filtered_df.drop(columns=['text_length'])

# Check sentiment balance
print(filtered_df['sentiment'].value_counts(normalize=True))

# Save cleaned dataset
filtered_df.to_csv('cleaned_sentiment_data.csv', index=False)
```

```
sentiment
sarcasm
 0.252288 happy
 0.251531 sad
 0.248185 irony
 0.247996
Name: proportion, dtype: float64
```

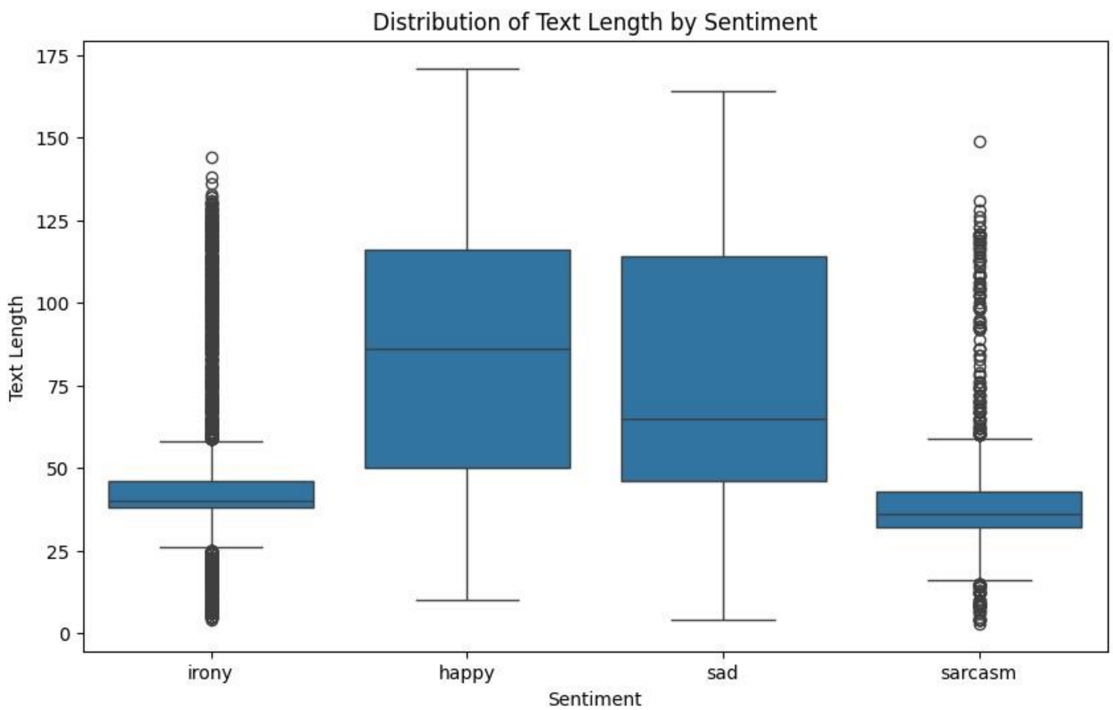
[16]:
```python
import matplotlib.pyplot as plt

# Plot sentiment distribution
filtered_df['sentiment'].value_counts().plot(kind='bar', color=['skyblue',
 ↪'violet', 'yellow'], edgecolor='black')

# Customize the chart
plt.title('Sentiment Distribution After Removing Outliers ')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Show the chart
plt.show()
```



2 Model training process

[17]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Visualize the distribution of sentiments with box plot to detect outliers
plt.figure(figsize=(10, 6))
sns.boxplot(x='sentiment',  ='text_length', data=new_df)
plt.title('Distribution of Text Length by Sentiment ')
plt.xlabel('Sentiment')
plt.ylabel('Text Length')
plt.show()

# Identify outliers based on IQR (Interquartile Range)
def identify_outliers(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
```

```python
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data < lower_bound) | (data > upper_bound)]
    return outliers

# Analyze outliers for each sentiment category
for sentiment in new_df['sentiment'].unique():
    sentiment_data = new_df[new_df['sentiment'] == sentiment]['text_length']
    outliers = identify_outliers(sentiment_data)
    print(f"Outliers for sentiment  '{sentiment}':\n{outliers}\n")

# You can choose to remove the outliers based on your analysis
# For example, you can create a new DataFrame without the outliers:
# new_df_cleaned = new_df[~new_df['text_length'].isin(outliers.index)]
```




```
Outliers for sentiment 'irony':
0          9
17         8
30        76
45         5
64        62
        …
13356     88
13359    125
13374    109
13378     99
```

```
13380   124
Name: text_length, Length: 1124, dtype: int64

Outliers for sentiment 'happy':
Series([], Name: text_length, dtype: int64)

Outliers for sentiment 'sad':
Series([], Name: text_length, dtype: int64)

Outliers for sentiment 'sarcasm':
39       84
57        6
171      15
179       8
218      12
         …
13217    70
13232    64
13239   103
13266   131
13302    93
Name: text_length, Length: 206, dtype: int64
```

[18]:
```
!pip install datasets
```

[19]:
```
import pandas as pd import numpy as np import re
import torch from sklearn.model_selection import
train_test_split from
sklearn.feature_extraction.text import
TfidfVectorizer from xgboost import XGBClassifier
from sklearn.metrics import classification_report
from transformers import BertTokenizer,
BertForSequenceClassification, Trainer,␣
↪TrainingArguments from datasets import
Dataset from torch.utils.data import
DataLoader from sklearn.metrics import
classification_report from transformers
import AutoTokenizer, AutoModel
from tqdm import tqdm from
xgboost import XGBClassifier
```

[20]:
```
# Examine the shape of the DataFrame
print("DataFrame Shape:", new_df.shape)

# Check for missing values
print("\nMissing Values:\n", new_df.isnull().sum())
```

```
# Analyze the distribution of sentiments
print("\nSentiment Distribution:\n",
new_df['sentiment'].value_counts())

# Examine the length of the text data
new_df['text_length'] =
new_df['text'].apply(len) print("\nText
Length Statistics:") print("Average:",
new_df['text_length'].mean())
print("Minimum:",
new_df['text_length'].min())
print("Maximum:",
new_df['text_length'].max())
```

```
DataFrame Shape: (32273, 17)
Missing Values:

 tweet_id                        1222
airline_sentiment              18886
airline_sentiment_confidence18886
negativereason                 24073
negativereason_confidence      22818
airline                        18886
airline_sentiment_gold         32235
name                           18886
negativereason_gold            32243
retweet_count                  18886
text                               0
tweet_coord                    31340
tweet_created                  18886
tweet_location                 23191
user_timezone                  23304
sentiment                          0
text_length                        0
dtype: int64
Sentiment Distribution:
 sentiment
happy  8076 sad
 8076 sarcasm
 8076 irony 8045
Name: count, dtype: int64

Text Length Statistics:
Average: 60.77299910141605
Minimum: 3
Maximum: 171
```

[21]:

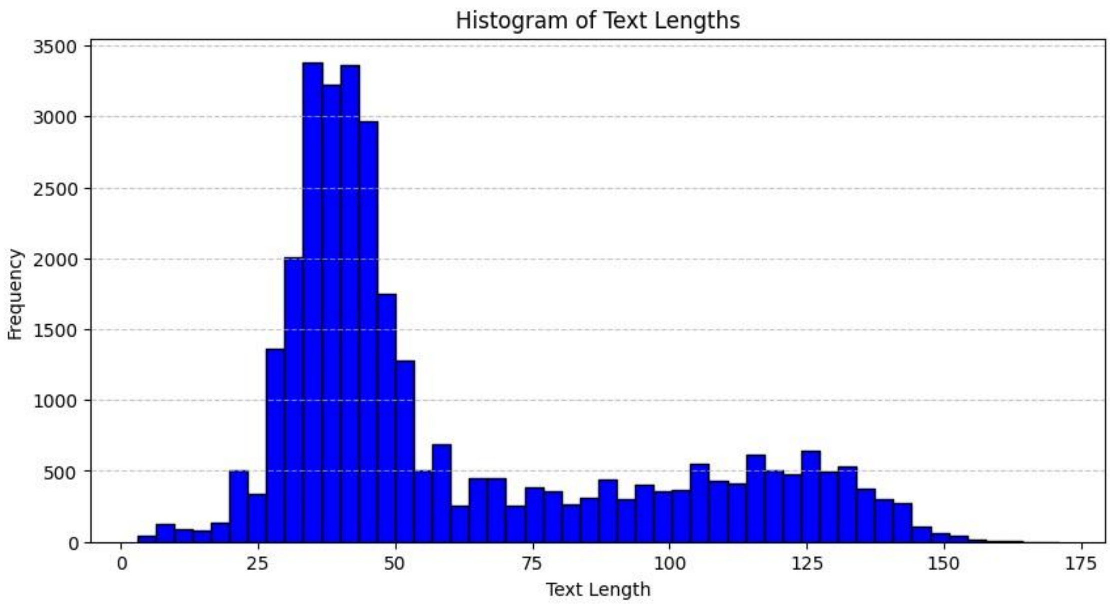
```
# Identify potential outliers or anomalies in text length
print("\nText Length Outliers (Top 10 longest):")
```

```
print(new_df.sort_values('text_length',
ascending=False)['text'].head(10)) print("\nText Length Outliers
(Top 10 shortest):") print(new_df.sort_values('text_length',
ascending=True)['text'].head(10))
```

```
Text Length Outliers (Top 10 longest):
10701My Flight Booking Problems Cld just time out w…
11177When Flight Booking Problems an intl flight on…
629   we were not given the option of using our
Unit… 12214 I tried that amp they have been
disrespectful …
9188 spent hour in line trying to get on a flight h…
6866 When I got your alert I immediately started lo…
7675 Hey guy Your Flight Booking Problems system ra…
852   Fail You Cancelled Flightled our flight frm
Gj… 4949    jumped the gun a little Cancelled
Flighting ou…
1527  every time I search a flight your site log me
… Name: text, dtype: object

Text Length Outliers (Top 10 shortest):
5096 min 3669
both
4018     sent
11788 amen 2735
thnx 2080   suck
7975     deal
13064 inch 7352
cool
7757    I cri
```

```
Name: text, dtype: object
```

[22]:
```python
plt.figure(figsize=(10, 5))
plt.hist(new_df['text_length'], bins=50, color='blue', edgecolor='black')

plt.title('Histogram of Text Lengths ')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()
```



[23]:
```python
import re
import nltk
from nltk.corpus import stopwords
from transformers import BertTokenizer

# Download stopwords if not already downloaded
nltk.download('stopwords', quiet=True)

# Initialize BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
def preprocess_text(text):
  """Preprocesses the text data."""
  text = clean_text(text)
  tokens = tokenizer.tokenize(text)
  return tokens

# Apply preprocessing to the 'text' column
new_df['processed_text'] = new_df['text'].apply(preprocess_text)

# Display the first few rows with processed text
display(new_df[['text', 'processed_text']].head())
```

```
                                              text  \
0                                        What said
1                   plus youve added commercial to
                                  the experience …
2                   I didnt today Must mean I need
                                 to take another…
3                   it really aggressive to blast
                                 obnoxious entert…
4                   and it a really big bad thing
                                 about it This is…

                                   processed_text
0                                    [what, said]
1                   [plus, you, ##ve, added,
                   commercial, to, the, …
2                   [i, didn, ##t, today, must, mean,
                   i, need, to,…
3                   [it, really, aggressive, to, blast,
                   ob, ##no, …
4                   [and, it, a, really, big, bad,
                   thing, about, i…
```

[27]:
```
# 3. Train-Test Split train_texts, test_texts, train_labels,
test_labels = train_test_split( new_df['text'],
new_df['sentiment'], test_size=0.3, random_state=42,␣
↪stratify=new_df['sentiment']
)
```

[28]:
```
# 4. Load BERT Model & Tokenizer
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name) model =
AutoModel.from_pretrained(model_name,
output_hidden_states=True) model.eval()
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

def generate_bert_embeddings_batch(texts, batch_size=32):
    """Generates BERT embeddings for a batch of texts.""" embeddings
    = [] for i in tqdm(range(0, len(texts), batch_size),
    desc="Processing Batches"):
        batch_texts = texts[i:i+batch_size]
        encoded_input = tokenizer(batch_texts,
        add_special_tokens=True,␣
 ↪padding=True, truncation=True, max_length=512, return_tensors='pt')
        encoded_input = {k: v.to(device) for k, v in
        encoded_input.items()}

        with torch.no_grad():
            outputs = model(**encoded_input)

        batch_embeddings = outputs.last_hidden_state[:, 0,
    :].cpu().numpy() embeddings.extend(batch_embeddings) return
    np.array(embeddings)
```

[29]:
```
# 5. Convert text data into BERT features
X_train = generate_bert_embeddings_batch(train_texts.tolist(),
batch_size=32)
X_test = generate_bert_embeddings_batch(test_texts.tolist(),
batch_size=32)
```

```
Processing Batches: 100%|| 706/706 [00:41<00:00, 17.15it/s]
Processing Batches: 100%|| 303/303 [00:18<00:00, 16.35it/s]
```

[30]:
```
# 6. Convert Labels to Numeric label_map = {label: i for i, label
in enumerate(new_df['sentiment'].unique())} y_train =
train_labels.map(label_map) y_test = test_labels.map(label_map)
```

[31]:
```
# 7. Train XGBoost Model xgb_model =
XGBClassifier(use_label_encoder=False,␣
 ↪eval_metric='mlogloss',n_estimators=500, #
    Increase trees learning_rate=0.03, # Reduce
    learning rate max_depth=8, # Deeper trees
    subsample=0.8, # Reduce overfitting
    colsample_bytree=0.8 # Feature selection
    )
xgb_model.fit(X_train, y_train)
```

[31]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
colsample_bylevel=None, colsample_bynode=None, colsample_bytree=0.8,
```

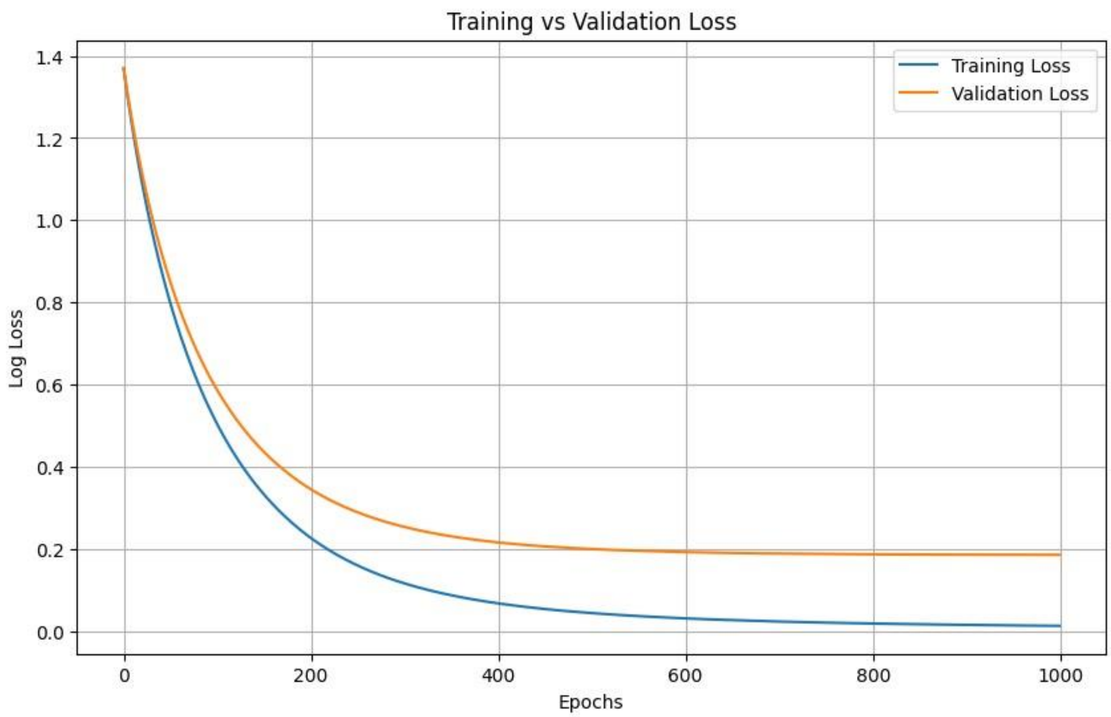
```
device=None, early_stopping_rounds=None, enable_categorical=False,
eval_metric='mlogloss', feature_types=None, gamma=None,
grow_policy=None, importance_type=None, interaction_constraints=None,
learning_rate=0.03, max_bin=None, max_cat_threshold=None,
max_cat_to_onehot=None, max_delta_step=None, max_depth=8,
max_leaves=None, min_child_weight=None, missing=nan,
monotone_constraints=None, multi_strategy=None, n_estimators=500,
n_jobs=None, num_parallel_tree=None, objective='multi:softprob', …)
```

[34]:
```python
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from xgboost import XGBClassifier
import numpy as np

# Step 1: Prepare Evaluation Sets
eval_set = [(X_train, y_train), (X_test, y_test)]

# Step 2: Initialize XGBoost Model with Enhanced Hyperparameters
xgb_model = XGBClassifier(
    use_label_encoder=False,
    eval_metric='mlogloss',
    n_estimators=1000,          # Increased trees for fine-tuning
    learning_rate=0.01,         # Lower learning rate
    max_depth=10,               # Deeper trees for capturing complex patterns
    min_child_weight=3,         # Regularization to reduce overfitting
    subsample=0.7,              # Subsample ratio for robustness
    colsample_bytree=0.7,       # Feature selection
    reg_alpha=0.1,              # L1 regularization
    reg_lambda=0.5              # L2 regularization
)

# Step 3: Fit the Model
```

```python
xgb_model.fit(X_train, y_train, eval_set=eval_set, verbose=True)

# Step 4: Plot Training vs Validation Loss
eval_results = xgb_model.evals_result_
epochs = range(len(eval_results['validation_0']['mlogloss']))

plt.figure(figsize=(10, 6))
plt.plot(epochs, eval_results['validation_0']['mlogloss'], label='Training Loss')
plt.plot(epochs, eval_results['validation_1']['mlogloss'], label='Validation Loss')
plt.title('Training vs Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Log Loss')
plt.legend()
plt.grid()
plt.show()

# Step 5: Predict and Evaluate
y_pred = xgb_model.predict(X_test)

print("XGBoost with BERT Features - Classification Report:")
print(classification_report(y_test, y_pred))

# Step 6: Confusion Matrix Visualization
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# Step 7: Hyperparameter Tuning (Optional)
# Use GridSearchCV or RandomizedSearchCV for further optimization
# Example:
# from sklearn.model_selection import GridSearchCV
# param_grid = {
#     'max_depth': [6, 8, 10],
#     'learning_rate': [0.01, 0.05, 0.1],
#     'n_estimators': [500, 1000],
#     'min_child_weight': [1, 3, 5],
# }
# grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=3, scoring='accuracy', verbose=2)
# grid_search.fit(X_train, y_train)
# print("Best Parameters:", grid_search.best_params_)
```

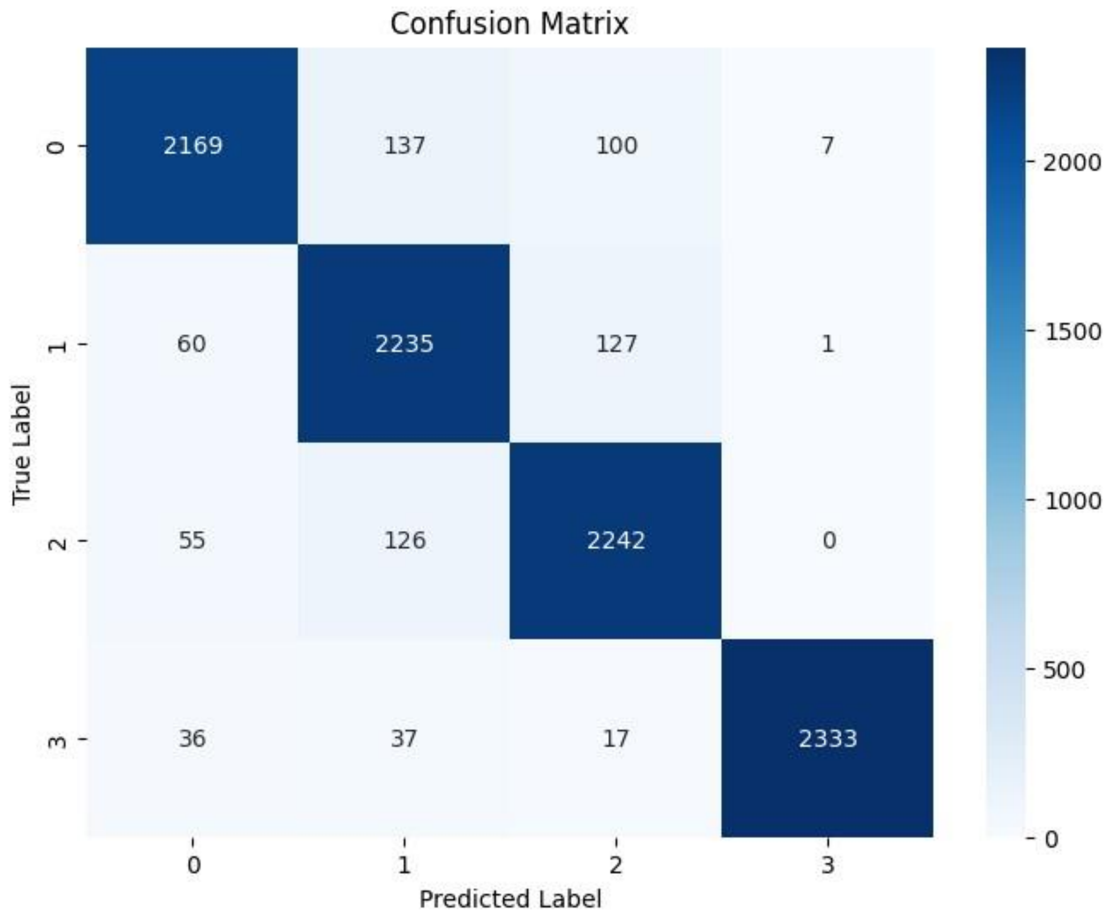
[0]	validation_0-mlogloss:1.36927	validation_1-mlogloss:1.37053
[1]	validation_0-mlogloss:1.35258	validation_1-mlogloss:1.35504
[2]	validation_0-mlogloss:1.33618	validation_1-mlogloss:1.33984
[3]	validation_0-mlogloss:1.32011	validation_1-mlogloss:1.32506
[4]	validation_0-mlogloss:1.30445	validation_1-mlogloss:1.31059
[5]	validation_0-mlogloss:1.28903	validation_1-mlogloss:1.29634
[6]	validation_0-mlogloss:1.27385	validation_1-mlogloss:1.28235
[7]	validation_0-mlogloss:1.25887	validation_1-mlogloss:1.26855
[8]	validation_0-mlogloss:1.24416	validation_1-mlogloss:1.25501
[9]	validation_0-mlogloss:1.22978	validation_1-mlogloss:1.24179
[10]	validation_0-mlogloss:1.21566	validation_1-mlogloss:1.22882
[11]	validation_0-mlogloss:1.20178	validation_1-mlogloss:1.21615
[12]	validation_0-mlogloss:1.18809	validation_1-mlogloss:1.20362
[13]	validation_0-mlogloss:1.17463	validation_1-mlogloss:1.19127
[14]	validation_0-mlogloss:1.16141	validation_1-mlogloss:1.17914
[15]	validation_0-mlogloss:1.14842	validation_1-mlogloss:1.16728
[16]	validation_0-mlogloss:1.13562	validation_1-mlogloss:1.15553
[17]	validation_0-mlogloss:1.12304	validation_1-mlogloss:1.14403
[18]	validation_0-mlogloss:1.11066	validation_1-mlogloss:1.13271
[19]	validation_0-mlogloss:1.09853	validation_1-mlogloss:1.12165
[20]	validation_0-mlogloss:1.08655	validation_1-mlogloss:1.11070
[21]	validation_0-mlogloss:1.07478	validation_1-mlogloss:1.10000
[22]	validation_0-mlogloss:1.06312	validation_1-mlogloss:1.08938
[23]	validation_0-mlogloss:1.05164	validation_1-mlogloss:1.07890

```
……… …   …..   .. .. .. .. .
 [984] validation_0-                validation_1-
       mlogloss:0.01346             mlogloss:0.18593
[985]  validation_0-                validation_1-
       mlogloss:0.01344             mlogloss:0.18592
[986]  validation_0-                validation_1-
       mlogloss:0.01342             mlogloss:0.18592
[987]  validation_0-                validation_1-
       mlogloss:0.01340             mlogloss:0.18591
[988]  validation_0-                validation_1-
       mlogloss:0.01338             mlogloss:0.18591
[989]  validation_0-                validation_1-
       mlogloss:0.01336             mlogloss:0.18591
[990]  validation_0-                validation_1-
       mlogloss:0.01333             mlogloss:0.18592
[991]  validation_0-                validation_1-
       mlogloss:0.01331             mlogloss:0.18591
[992]  validation_0-                validation_1-
       mlogloss:0.01329             mlogloss:0.18590
[993]  validation_0-                validation_1-
       mlogloss:0.01327             mlogloss:0.18591
[994]  validation_0-                validation_1-
       mlogloss:0.01325             mlogloss:0.18591
[995]  validation_0-                validation_1-
       mlogloss:0.01323             mlogloss:0.18591
[996]  validation_0-                validation_1-
       mlogloss:0.01321             mlogloss:0.18591
[997]  validation_0-                validation_1-
       mlogloss:0.01319             mlogloss:0.18594
[998]  validation_0-                validation_1-
       mlogloss:0.01317             mlogloss:0.18594
[999]  validation_0-                validation_1-
       mlogloss:0.01315             mlogloss:0.18594
```



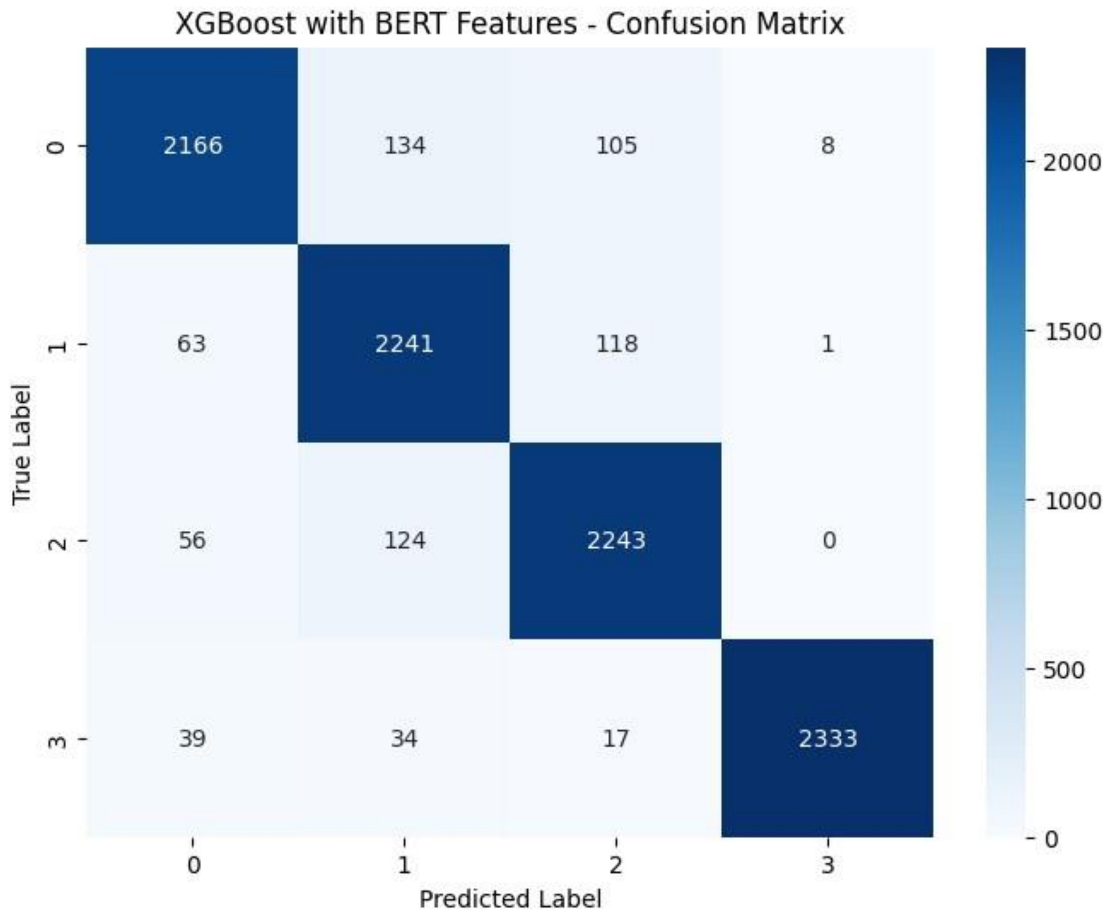
XGBoost with BERT Features - Classification Report:

	precision	recall	f1-score	support
0	0.93	0.90	0.92	2413
1	0.88	0.92	0.90	2423
2	0.90	0.93	0.91	2423
3	1.00	0.96	0.98	2423
accuracy			0.93	9682
macro avg	0.93	0.93	0.93	9682
weighted avg	0.93	0.93	0.93	9682



3 Evaluate XGBoost Model

[36]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Generate confusion matrix
cm = confusion_matrix(y_test, xgb_preds)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=np.
  ↪unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("XGBoost with BERT Features - Confusion Matrix ")
plt.show()
```



[ ]: *# 9. Train BERT Model*

```
train_dataset = Dataset.from_pandas(pd.DataFrame({"text":
train_texts, "label":␣
 ↪y_train})) test_dataset =
Dataset.from_pandas(pd.DataFrame({"text": test_texts, "label":␣
↪y_test}))

def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length",
    truncation=True)

train_dataset = train_dataset.map(tokenize_function, batched=True)
test_dataset = test_dataset.map(tokenize_function, batched=True)

bert_classification_model = BertForSequenceClassification.
 ↪from_pretrained("bert-base-uncased",
num_labels=len(label_map)) training_args =
TrainingArguments( output_dir="./results",
evaluation_strategy="epoch",␣
```

```
↪per_device_train_batch_size=8, per_device_eval_batch_size=8,␣
↪num_train_epochs=3, weight_decay=0.01
)

trainer = Trainer(
    model=bert_classification_model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset
)

trainer.train()

# 10. Evaluate BERT Model
preds = trainer.predict(test_dataset)
pred_labels = np.argmax(preds.predictions, axis=1)
print("BERT Classification Report:")
print(classification_report(y_test, pred_labels))
```

[35]:
```
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
import numpy as np

# ... (your existing code for loading, preprocessing, training, and prediction)␣
↪...

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, xgb_preds )
precision = precision_score(y_test, xgb_preds , average='weighted')  # Use␣
↪weighted average for multi-class
recall = recall_score(y_test, xgb_preds , average='weighted')
f1 = f1_score(y_test, xgb_preds , average='weighted')

print(f"Accuracy: {round(accuracy*100)}")
print(f"Precision: {round(precision*100)}")
print(f"Recall: {round(recall*100)}")
print(f"F1-score: {round(f1*100)}")
```

```
Accuracy: 93
Precision: 93
Recall: 93
F1-score: 93
```