NLP_Sentiment_Analysis

October 6, 2024

1 1. Introduction

1.1 1.1. Project Overview

This project focuses on analyzing public sentiment from Tweets regarding Apple and Google products. The main goal is to build a Natural Language Processing (NLP) model that can classify the sentiment of a Tweet as positive, negative, or neutral based on its content.

The data, sourced from CrowdFlower, contains over 9,000 Tweets labeled by human raters. By analyzing this data, the model will help Apple and Google better understand customer perceptions of their products, allowing them to make informed decisions for marketing, customer service, and product development.

We will begin by preprocessing the Tweets, transforming them into a numerical format suitable for machine learning models, and training several classification algorithms to evaluate their performance.

1.2 1.2. Business Problem & Stakeholder

1.2.1 Business Problem:

Public perception of tech products can heavily influence a company's sales, customer satisfaction, and brand loyalty. For companies like Apple and Google, understanding how customers feel about their products can provide valuable insights into areas for improvement, marketing strategies, and product development.

In this project, we aim to develop a model that automatically classifies the sentiment of Tweets regarding Apple and Google products as positive, negative, or neutral. This can help companies quickly gauge public sentiment at scale, providing actionable insights for decision-making.

1.2.2 Stakeholders:

- **Apple and Google Product Teams:** Use sentiment data to improve products and address customer pain points.
- **Marketing Departments:** Tailor campaigns to target sentiment-driven messaging.
- **Customer Support Teams:** Identify negative feedback more quickly to address concerns.
- **Executives/Decision Makers:** Gain a high-level view of public opinion, enabling better strategic planning.

1.3 1.3. Dataset Description

The dataset used in this project comes from CrowdFlower, containing over 9,000 Tweets. Each Tweet has been labeled by human raters with one of three sentiment categories: positive, negative, or neither. The dataset contains the following key columns:

1.3.1 Key Features of the Dataset:

- **Text**: The actual content of the Tweet, which we will analyze for sentiment classification.
- **Brand/Product**: The specific product or brand mentioned in the Tweet, such as `iPhone`, `Google`, or `iPad`.
- **Emotion**: The sentiment label, representing whether the sentiment expressed in the Tweet is `Positive emotion`, `Negative emotion`, or neutral.

1.3.2 Target Variable:

- **Emotion**: This will be the target variable, as it captures the sentiment associated with each Tweet.

The dataset will be used to train and evaluate models that can predict the sentiment of unseen Tweets based on their text content.

1.4 1.4. Objectives

1.4.1 Specific Objective

- Develop a Natural Language Processing (NLP) model to accurately classify the sentiment of Tweets about Apple and Google products into three categories: positive, negative, and neutral.

1.4.2 Additional Objectives

1. **Data Preprocessing**: Clean and prepare the Tweet data for analysis, which includes handling missing values, normalizing text (removing URLs, special characters, etc.), and tokenization.
2. **Exploratory Data Analysis (EDA)**: Conduct exploratory analysis to understand the distribution of sentiments in the dataset, identify any patterns, and visualize key aspects of the data.
3. **Model Development**: Implement and train various classification models (e.g., Logistic Regression, Support Vector Machines, and Naive Bayes) to classify the sentiment of Tweets.
4. **Model Evaluation**: Evaluate the performance of the models using appropriate metrics such as accuracy, precision, recall, and F1 score, especially focusing on multiclass classification metrics.
5. **Iterative Improvement**: Based on initial results, refine the models by incorporating advanced techniques such as feature engineering, hyperparameter tuning, or using pre-trained embeddings like Word2Vec or BERT.
6. **Conclusion and Recommendations**: Summarize the findings, discuss the model's effectiveness, and provide actionable recommendations for stakeholders based on the analysis.

2 2. Data Understanding & Exploration

2.1 2.1. Data Loading & Initial Exploration

In this step, we will:

1. Load the dataset.
2. Display the first few rows to understand the structure.
3. Check the data types of each column.
4. Identify missing values and duplicates.
5. Analyze the class distribution to check for potential imbalance.
6. Perform basic descriptive statistics (e.g., distribution of Tweet lengths and word counts).

[1]:
```
# Importing the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Display settings
%matplotlib inline
plt.style.use('ggplot')  # for nicer visualizations
sns.set_palette("Set2")  # Set Seaborn color palette
```

[2]:
```
# Load the dataset
raw_data = pd.read_csv("judge-1377884607_tweet_product_company.csv", encoding =␣
 ↪"latin1")
```

[3]:
```
# Display the first 5 rows
raw_data.head()
```

[3]:
```
                                          tweet_text  \
0  .@wesley83 I have a 3G iPhone. After 3 hrs twe…
1  @jessedee Know about @fludapp ? Awesome iPad/i…
2  @swonderlin Can not wait for #iPad 2 also. The…
3  @sxsw I hope this year's festival isn't as cra…
4  @sxtxstate great stuff on Fri #SXSW: Marissa M…

  emotion_in_tweet_is_directed_at  \
0                          iPhone
1              iPad or iPhone App
2                            iPad
3              iPad or iPhone App
4                          Google

  is_there_an_emotion_directed_at_a_brand_or_product
0                                   Negative emotion
1                                   Positive emotion
2                                   Positive emotion
```

```
3                                    Negative emotion
4                                    Positive emotion
```

The column names seem to be unnecessarily long, we'll rename them to improve readability.

[4]:
```
# Renaming the columns
raw_data.columns = ["Text", "Brand/Product", "Emotion"]
raw_data.head()
```

[4]:
```
                                                Text       Brand/Product  \
0  .@wesley83 I have a 3G iPhone. After 3 hrs twe…              iPhone
1  @jessedee Know about @fludapp ? Awesome iPad/i…  iPad or iPhone App
2  @swonderlin Can not wait for #iPad 2 also. The…                iPad
3  @sxsw I hope this year's festival isn't as cra…  iPad or iPhone App
4  @sxtxstate great stuff on Fri #SXSW: Marissa M…              Google

            Emotion
0  Negative emotion
1  Positive emotion
2  Positive emotion
3  Negative emotion
4  Positive emotion
```

[5]:
```
# Checking the number of raws and columns
raw_data.shape
```

[5]:
```
(9093, 3)
```

[6]:
```
# Getting an overview of the data types
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9093 entries, 0 to 9092
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Text           9092 non-null   object
 1   Brand/Product  3291 non-null   object
 2   Emotion        9093 non-null   object
dtypes: object(3)
memory usage: 213.2+ KB
```

[7]:
```
# Checking for missing values
raw_data.isnull().sum()
```

[7]:
```
Text                1
Brand/Product    5802
Emotion             0
```

```
dtype: int64
```

```
[8]: # Checking for duplicates based on the Text column
duplicates = raw_data.duplicated(subset='Text').sum()
print(f"Number of duplicate rows based on the text column: {duplicates}")
```

```
Number of duplicate rows based on the text column: 27
```

```
[9]: # Handle missing values in 'Text' column
raw_data['Text'] = raw_data['Text'].fillna('')  # Replace NaN with an empty␣
 ↪string

# Descriptive statistics for tweet length and word count
raw_data['tweet_length'] = raw_data['Text'].apply(len)
raw_data['word_count'] = raw_data['Text'].apply(lambda x: len(x.split()))

print("Tweet Length Statistics:")
print(raw_data['tweet_length'].describe())

print("Word Count Statistics:")
print(raw_data['word_count'].describe())
```

```
Tweet Length Statistics:
count    9093.000000
mean      104.950731
std        27.208419
min         0.000000
25%        86.000000
50%       109.000000
75%       126.000000
max       178.000000
Name: tweet_length, dtype: float64
Word Count Statistics:
count    9093.000000
mean       17.763444
std         4.964105
min         0.000000
25%        14.000000
50%        18.000000
75%        21.000000
max        33.000000
Name: word_count, dtype: float64
```

- **Average tweet length** is *105 characters* with a max of *178 characters.*
- **Average word count** is around *18 words*, with a maximum of *33 words.*

2.2 2.2. Data Cleaning

In this section, we will:

- Handle missing values in the `Brand/Product` column.
- Remove any duplicate rows.
- Clean the text data by removing URLs, mentions, and special characters.

2.2.1 Handling Missing Values

Since the `Brand/Product` column contains a large number of missing values, we can consider one of the following approaches:

- Dropping the column entirely if it's not necessary for the analysis.
- Imputing values, but this might be challenging as this is categorical text data.

For now, since this column is not critical to sentiment analysis, we'll drop it.

[10]:
```python
# Dropping the 'Brand/Product' column due to many missing values
data_cleaned = raw_data.drop(columns=['Brand/Product'])
data_cleaned.head()
```

[10]:
```
                                                Text           Emotion  \
0  .@wesley83 I have a 3G iPhone. After 3 hrs twe…  Negative emotion
1  @jessedee Know about @fludapp ? Awesome iPad/i…  Positive emotion
2  @swonderlin Can not wait for #iPad 2 also. The…  Positive emotion
3  @sxsw I hope this year's festival isn't as cra…  Negative emotion
4  @sxtxstate great stuff on Fri #SXSW: Marissa M…  Positive emotion

   tweet_length  word_count
0           127          23
1           139          22
2            79          15
3            82          15
4           131          17
```

2.2.2 Handling Duplicates

We'll remove the duplicate rows.

[11]:
```python
# Removing duplicate rows
data_cleaned = data_cleaned.drop_duplicates()
print(f"Number of rows after removing duplicates: {data_cleaned.shape[0]}")
```

```
Number of rows after removing duplicates: 9071
```

2.2.3 Cleaning the Text Column

To prepare the Text Column for modeling, we'll:

- Remove URLs, mentions, and special characters.

- Convert text to lowercase for uniformity.

[12]:
```
import re

# Function to clean Text Column
def clean_text(text):
    text = re.sub(r"http\S+", "", text)  # Remove URLs
    text = re.sub(r"@\w+", "", text)  # Remove mentions
    text = re.sub(r"#\w+", "", text)  # Remove hashtags (optional)
    text = re.sub(r"[^a-zA-Z\s]", "", text)  # Remove special characters and␣
 ↪numbers
    text = text.lower()  # Convert to lowercase
    text = text.strip()  # Remove leading/trailing spaces
    return text

# Applying the cleaning function to the 'Text' column
data_cleaned['cleaned_text'] = data_cleaned['Text'].apply(clean_text)
data_cleaned[['Text', 'cleaned_text']].head()
```

```
<ipython-input-12-c74fe941c037>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_cleaned['cleaned_text'] = data_cleaned['Text'].apply(clean_text)
```

[12]:
```
                                                Text  \
0  .@wesley83 I have a 3G iPhone. After 3 hrs twe…
1  @jessedee Know about @fludapp ? Awesome iPad/i…
2  @swonderlin Can not wait for #iPad 2 also. The…
3  @sxsw I hope this year's festival isn't as cra…
4  @sxtxstate great stuff on Fri #SXSW: Marissa M…

                                        cleaned_text
0  i have a g iphone after  hrs tweeting at  it w…
1  know about   awesome ipadiphone app that youll…
2  can not wait for   also they should sale them …
3  i hope this years festival isnt as crashy as t…
4  great stuff on fri  marissa mayer google tim o…
```

2.3 Cleaning the Emotion Column

[13]:
```
# Checking the occurence of unique values
data_cleaned["Emotion"].value_counts()
```

[13]:
```
Emotion
No emotion toward brand or product    5376
Positive emotion                      2970
Negative emotion                       569
I can't tell                           156
Name: count, dtype: int64
```

[14]:
```
data_cleaned["Emotion"] = data_cleaned["Emotion"].replace({
    "No emotion toward brand or product": "Neutral",
    "Positive emotion": "Positive",
    "Negative emotion": "Negative"
})

data_cleaned["Emotion"].value_counts()
```

```
<ipython-input-14-0ce3e650cb05>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_cleaned["Emotion"] = data_cleaned["Emotion"].replace({
```

[14]:
```
Emotion
Neutral         5376
Positive        2970
Negative         569
I can't tell     156
Name: count, dtype: int64
```

[15]:
```
# Setting display options to show the full text
pd.set_option('display.max_colwidth', None)

# Displaying the tweets where Emotion is "I can't tell"
cant_tell_tweets = data_cleaned[data_cleaned["Emotion"] == "I can't␣
 ↪tell"]["cleaned_text"]
print(cant_tell_tweets)
```

```
90                                                        thanks to  for
publishing the news of  new medical apps at the  conf blog link
102                                    quotapple has opened a popup store in
austin so the nerds in town for  can get their new ipads link
237                               just what america needs rt  google to launch
major new social network called circles possibly today link
341                                                               the queue at
the apple store in austin is four blocks long crazy stuff
368                                         hope its better than wave rt  buzz is
googles previewing a social networking platform at  link
```

```
                                                                             …
9020    its funny watching a room full of people hold their ipad in the air to
take a photo like a room full of tablets staring you down
9032
yeah we have   google has nothing on us
9037                                                                   yes the
google presentation was not exactly what i was expecting
9058                   quotdo you know what apple is really good at making you
feel bad about your xmas presentquot  seth meyers on ipad
9066          how much you want to bet apple is disproportionately stocking the
popup store with ipad  the influencerhipsters thank you
Name: cleaned_text, Length: 156, dtype: object
```

[16]:
```python
# Drop rows where the Emotion is "I can't tell"
data_cleaned = data_cleaned[data_cleaned["Emotion"] != "I can't tell"]
```

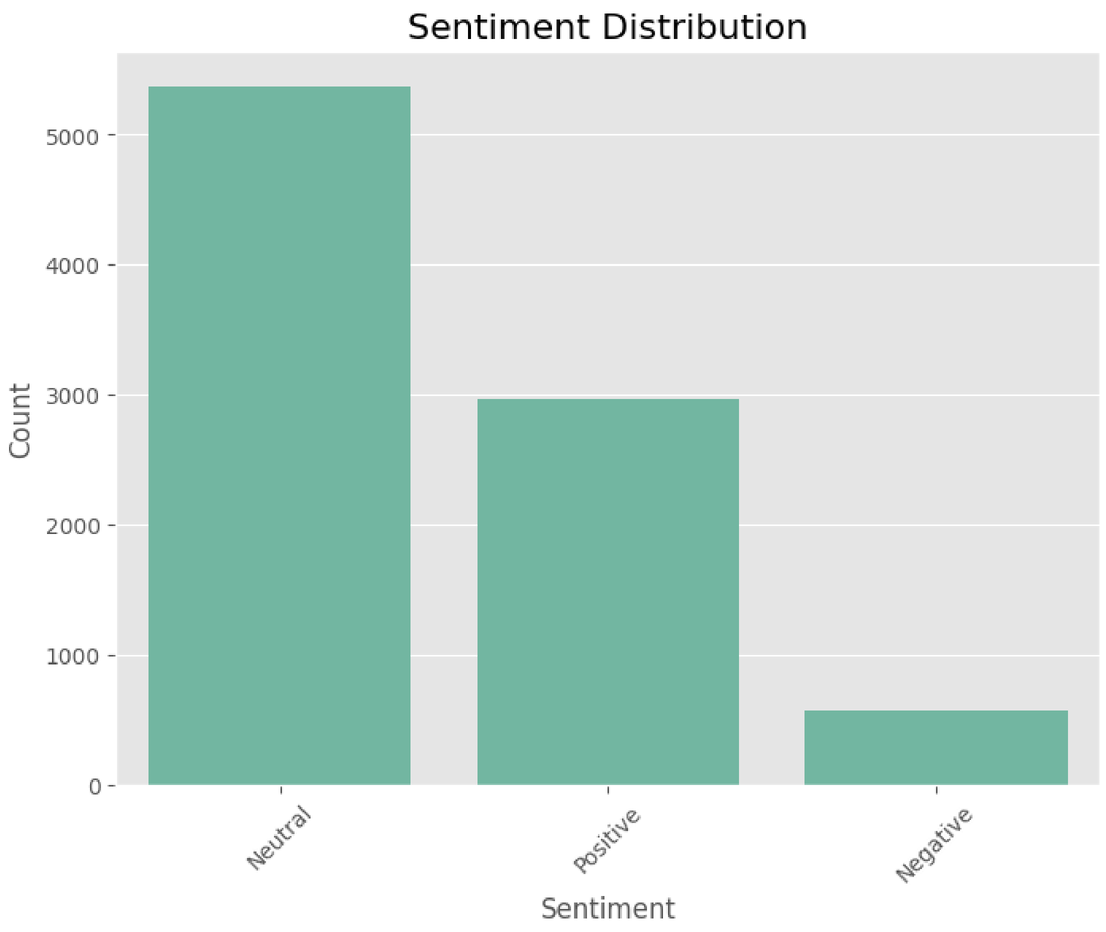
On investigating the text with the cant tell emotion we decided to drop the rows because there was no clear sentiment information, as some of the texts seemed sarcastic making them less useful for accurate analysis or modeling.

2.4 2.3. Sentiment Class Distribution Visualization

In this step, we will visualize the sentiment class distribution to better understand the balance between different sentiment categories in the dataset. We will use a bar chart to display the count of each sentiment label: `positive`, `negative`, and `neutral`.

The bar chart will help us assess the overall distribution and check for any significant class imbalance, which may affect our modeling process later.

[17]:
```python
# Setting plot size and style
plt.figure(figsize=(8, 6))

# Plotting sentiment distribution using Seaborn's countplot
sns.countplot(x='Emotion',
              data= data_cleaned,
              order= data_cleaned['Emotion'].value_counts().index)

# Adding title and labels
plt.title('Sentiment Distribution', fontsize=16)
plt.xlabel('Sentiment', fontsize=12)
plt.ylabel('Count', fontsize=12)

# Rotating x-axis labels for better readability
plt.xticks(rotation=45)

# Displaying the plot
plt.show()
```



The sentiment distribution graph clearly shows that the dataset is imbalanced, with the majority of tweets labeled as `Neutral`, followed by `Positive emotion,` while `Negative emotion` labels has significantly fewer instances.

This imbalance is something we'll need to address during the modeling phase, potentially through techniques like resampling or adjusting class weights.

2.5 2.4. Further Visualizations

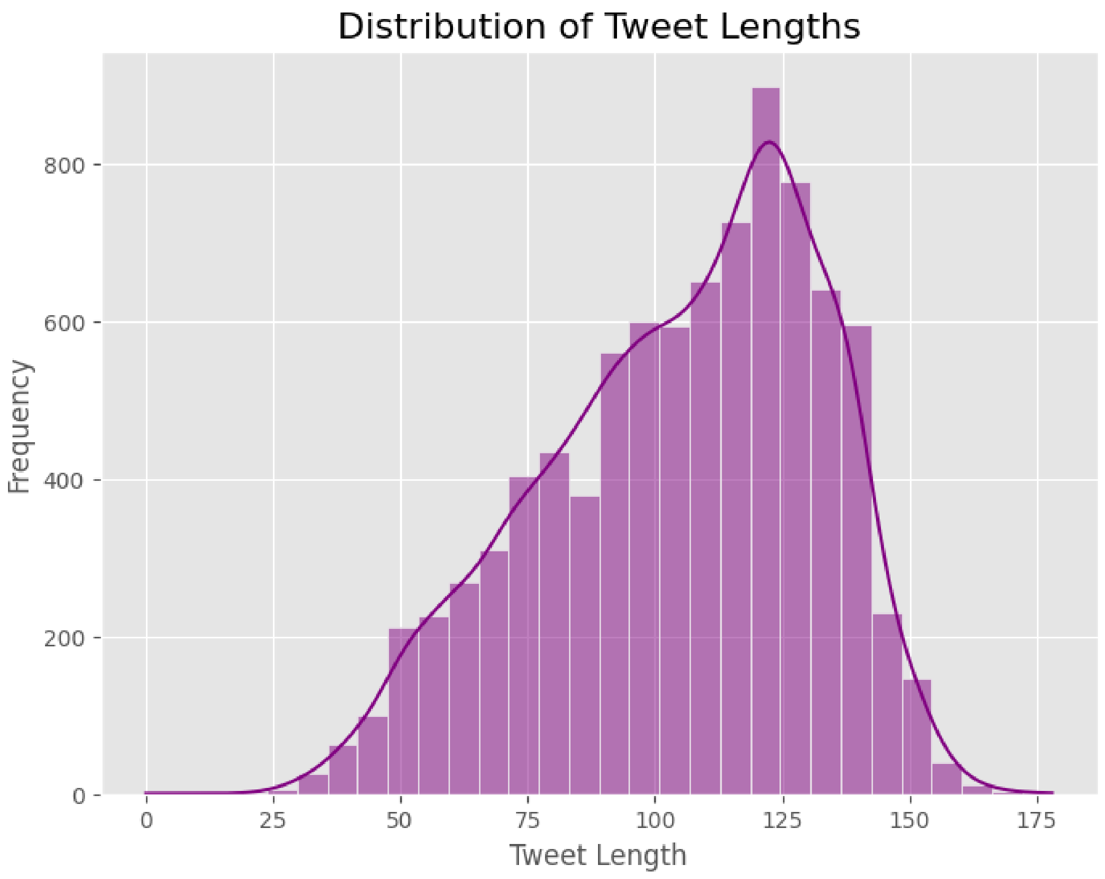
We will now:

- **Distribution of Tweet Lengths:** A histogram to show how the lengths of the tweets are distributed.
- **Word Count Distribution:** A similar histogram for word count distribution.
- **Correlation Between Tweet Length and Sentiment:** Boxplots to visualize how the length of the tweets varies with sentiment.

2.5.1 1. Distribution of Tweet Lengths

This visualization will help us understand how the lengths of the tweets are distributed in the dataset.

[18]:
```
# Plotting distribution of tweet lengths
plt.figure(figsize=(8, 6))
sns.histplot(data_cleaned['tweet_length'], bins=30, kde=True, color='purple')
plt.title('Distribution of Tweet Lengths', fontsize=16)
plt.xlabel('Tweet Length', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.show()
```



The histogram above illustrates the distribution of tweet lengths within the dataset. Key observations include:

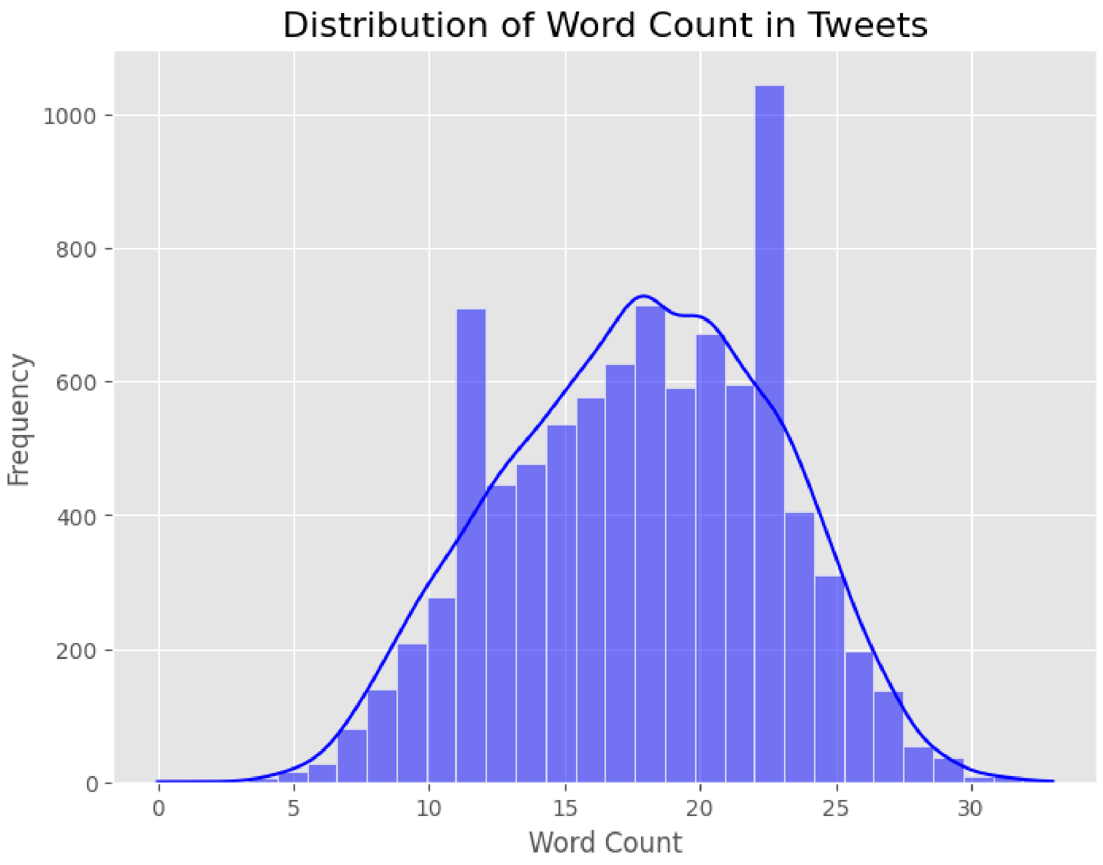
- The distribution appears to be roughly normal, with a peak around 100-125 characters.
- Most tweets fall within the range of 75 to 140 characters, suggesting that the average tweet is relatively concise, likely conforming to Twitter's character limits.
- There are fewer tweets on both extremes (very short and very long), with a noticeable decline in frequency as the tweet length approaches the maximum of 178 characters.

- The data shows some variability, but the majority of tweets are clustered around the mean, indicating a consistent tweeting style among users.

2.5.2 2. Distribution of Word Count in Tweets

We will now visualize the word count in tweets, which helps us understand how concise or detailed the tweets are.

[19]:
```python
# Plotting distribution of word count
plt.figure(figsize=(8, 6))
sns.histplot(data_cleaned['word_count'], bins=30, kde=True, color='blue')
plt.title('Distribution of Word Count in Tweets', fontsize=16)
plt.xlabel('Word Count', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.show()
```

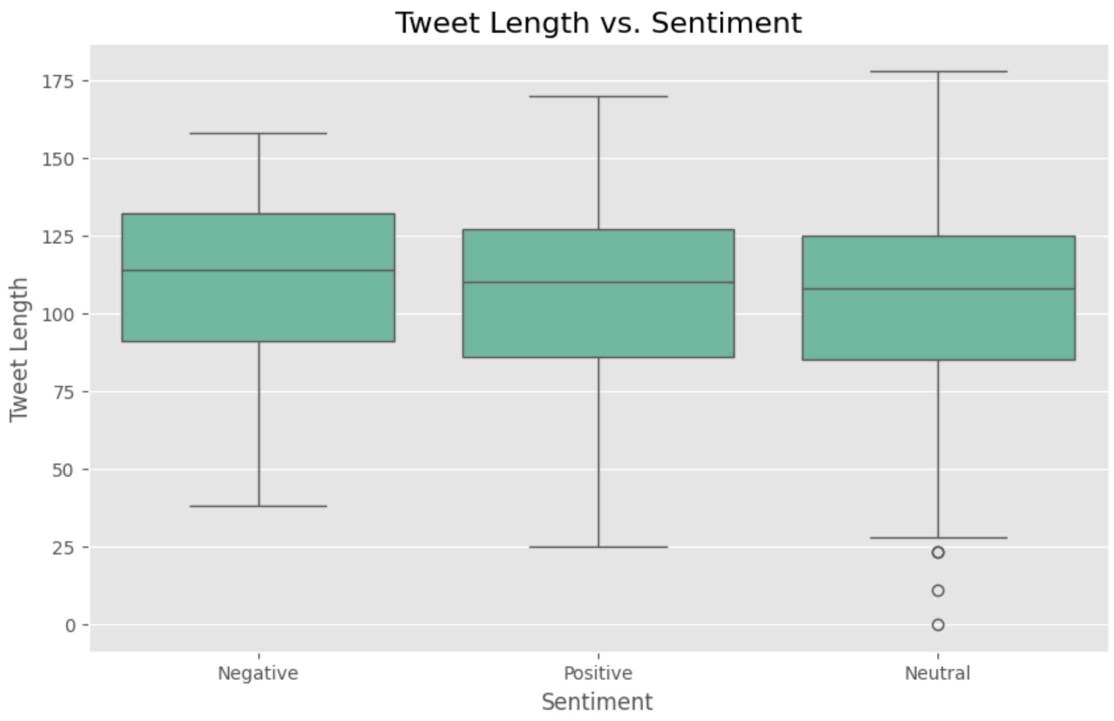


The histogram shows that most tweets contain between 10 and 25 words, with a peak around 20 words. Shorter tweets (around 10 words) are also common, while very short or long tweets are rare. The distribution approximates a normal curve centered at 20 words.

2.5.3 3. Correlation Between Tweet Length and Sentiment

To understand if tweet length has any relation to sentiment, we will visualize the correlation using boxplots.

[20]:
```python
# Boxplot of tweet length by sentiment
plt.figure(figsize=(10, 6))
sns.boxplot(x='Emotion', y='tweet_length', data=data_cleaned)
plt.title('Tweet Length vs. Sentiment', fontsize=16)
plt.xlabel('Sentiment', fontsize=12)
plt.ylabel('Tweet Length', fontsize=12)
plt.show()
```



- **Interquartile Range (IQR)**: The boxes indicate the interquartile range for each sentiment class, with the length of the boxes reflecting the variability in tweet lengths. The Negative tweets exhibit the widest range, indicating more diversity in how users express negative sentiments.
- **Outliers**: There are a few outlier tweets, particularly in the Negative category, which may indicate extreme cases of expression, where users felt the need to provide significantly longer explanations or complaints.

This suggests that users might express negative sentiments in more detail compared to positive sentiments, which are often more concise.

3 3. Data Preprocessing for NLP

3.1 3.1. Tokenization and Normalization

The goal of this step is to prepare the text for NLP tasks by transforming it into a consistent and analyzable format. We achieve this by:

1. Tokenization: Splitting the text into individual tokens (words).
2. Normalization: Converting text to lowercase and removing punctuation, numbers, and special characters.

After applying these steps, the text will be standardized, allowing for better processing in subsequent steps like vectorization.

[21]:
```
import re
import nltk
nltk.download('punkt')  # Ensure you have the required tokenizer

# Tokenization and Normalization Function
def tokenize_and_normalize(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove punctuation, numbers, and special characters
    text = re.sub(r'[^a-z\s]', '', text)

    # Tokenize the text
    tokens = nltk.word_tokenize(text)

    return tokens

# Apply the function to the cleaned text column
data_cleaned['tokens'] = data_cleaned['cleaned_text'].
  ↪apply(tokenize_and_normalize)

# Display the tokenized text
data_cleaned[['cleaned_text', 'tokens']].head()
```

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

[21]:
```
                    cleaned_text  \
0                    i have a g iphone after  hrs tweeting at  it was dead  i
need to upgrade plugin stations at
1  know about   awesome ipadiphone app that youll likely appreciate for its
design also theyre giving free ts at
2                                                           can not wait for
also they should sale them down at
3                                             i hope this years festival isnt as
```

```
crashy as this years iphone app
4        great stuff on fri  marissa mayer google tim oreilly tech
booksconferences amp matt mullenweg wordpress

                                            tokens
0                     [i, have, a, g, iphone, after, hrs, tweeting, at, it, was,
dead, i, need, to, upgrade, plugin, stations, at]
1  [know, about, awesome, ipadiphone, app, that, youll, likely, appreciate, for,
its, design, also, theyre, giving, free, ts, at]
2                                                                [can, not,
wait, for, also, they, should, sale, them, down, at]
3                                                 [i, hope, this, years,
festival, isnt, as, crashy, as, this, years, iphone, app]
4           [great, stuff, on, fri, marissa, mayer, google, tim, oreilly, tech,
booksconferences, amp, matt, mullenweg, wordpress]
```

3.2 3.2. Stopwords Removal

In this step, we will remove common words (stopwords) that don't carry much meaningful information for text analysis. Stopwords are often words like "the", "is", "in", etc., which are frequently used but don't contribute much to the overall meaning of a text.

Goal: Improve the quality of our text data by eliminating such words, leaving only the key terms that carry the most important meaning.

[22]:
```python
from nltk.corpus import stopwords

# Load the set of English stopwords from NLTK
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

# Function to remove stopwords from tokenized text
def remove_stopwords(tokens):
    return [token for token in tokens if token.lower() not in stop_words]

# Apply the function to the tokens column
data_cleaned['tokens_no_stopwords'] = data_cleaned['tokens'].
  ↪apply(remove_stopwords)

# Display the first few rows to check the result
data_cleaned[['tokens', 'tokens_no_stopwords']].head()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[22]:
```
                                            tokens  \
0                     [i, have, a, g, iphone, after, hrs, tweeting, at, it, was,
dead, i, need, to, upgrade, plugin, stations, at]
```

```
1  [know, about, awesome, ipadiphone, app, that, youll, likely, appreciate, for,
its, design, also, theyre, giving, free, ts, at]
2                                                                   [can, not,
wait, for, also, they, should, sale, them, down, at]
3                                                 [i, hope, this, years,
festival, isnt, as, crashy, as, this, years, iphone, app]
4           [great, stuff, on, fri, marissa, mayer, google, tim, oreilly, tech,
booksconferences, amp, matt, mullenweg, wordpress]

                  tokens_no_stopwords
0                                                    [g, iphone, hrs, tweeting,
dead, need, upgrade, plugin, stations]
1                  [know, awesome, ipadiphone, app, youll, likely, appreciate,
design, also, theyre, giving, free, ts]
2
[wait, also, sale]
3                                                             [hope, years,
festival, isnt, crashy, years, iphone, app]
4  [great, stuff, fri, marissa, mayer, google, tim, oreilly, tech,
booksconferences, amp, matt, mullenweg, wordpress]
```

3.3 3.3. Stemming and Lemmatization

In this step, we will apply stemming or lemmatization to further normalize the tokens.

Stemming reduces words to their root form by removing suffixes (e.g., "running" becomes "run"). It's a more aggressive approach but might produce non-existent words. Lemmatization is more linguistically accurate, converting words to their base or dictionary form (e.g., "running" becomes "run", and "better" becomes "good"). This method ensures the resulting words are real words.

For this project, we will use lemmatization as it tends to preserve the meaning better than stemming.

[23]:
```python
# Import the necessary NLTK downloader
import nltk

# Download the 'averaged_perceptron_tagger' resource for POS tagging
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')  # Download WordNet for lemmatization
nltk.download('omw-1.4')  # Optional: Download for expanded WordNet support

# Now, proceed with the original code
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

# Initialize lemmatizer
lemmatizer = WordNetLemmatizer()
```

```
# Helper function to get part of speech (POS) tags for more accurate␣
↪lemmatization
def get_pos_tag(word):
    # Mapping from POS tag to first character for lemmatizer
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {
        'J': wordnet.ADJ,
        'N': wordnet.NOUN,
        'V': wordnet.VERB,
        'R': wordnet.ADV
    }
    return tag_dict.get(tag, wordnet.NOUN)

# Function to apply lemmatization to tokens
def lemmatize_tokens(tokens):
    return [lemmatizer.lemmatize(token, get_pos_tag(token)) for token in tokens]

# Apply the lemmatization function to the tokens without stopwords
data_cleaned['tokens_lemmatized'] = data_cleaned['tokens_no_stopwords'].
↪apply(lemmatize_tokens)

# Display the first few rows to check the result
data_cleaned[['tokens_no_stopwords', 'tokens_lemmatized']].head()
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data] Downloading package omw-1.4 to /root/nltk_data…
```

[23]:
```
                  tokens_no_stopwords  \
0                                                     [g, iphone, hrs, tweeting,
dead, need, upgrade, plugin, stations]
1                  [know, awesome, ipadiphone, app, youll, likely, appreciate,
design, also, theyre, giving, free, ts]
2
[wait, also, sale]
3                                                             [hope, years,
festival, isnt, crashy, years, iphone, app]
4  [great, stuff, fri, marissa, mayer, google, tim, oreilly, tech,
booksconferences, amp, matt, mullenweg, wordpress]

                    tokens_lemmatized
0                                                          [g, iphone, hr, tweet,
dead, need, upgrade, plugin, station]
1                     [know, awesome, ipadiphone, app, youll, likely, appreciate,
design, also, theyre, give, free, t]
```

```
2
[wait, also, sale]
3                                                              [hope, year,
festival, isnt, crashy, year, iphone, app]
4  [great, stuff, fri, marissa, mayer, google, tim, oreilly, tech,
booksconferences, amp, matt, mullenweg, wordpress]
```

3.4 3.4. Vectorization

3.4.1 Steps for Vectorization

- **Ensure Cleaned Data is Present:** We will make sure that the cleaned data (after tokenization, normalization, stopwords removal, and lemmatization) is correctly defined and contains meaningful content.
- **Perform Vectorization:** Use `CountVectorizer` for `Bag of Words` representation and `TfidfVectorizer` for `TF-IDF` representation.

[26]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer


# Join tokens (which are lists) back into strings
data_cleaned['tokens_lemmatized_str'] = data_cleaned['tokens_lemmatized'].
  ↪apply(lambda tokens: ' '.join(tokens))

X = data_cleaned['tokens_lemmatized_str']  # Features (converted to strings)
y = data_cleaned['Emotion']  # Target variable (emotion)

# Train-test split (adjust test size as needed)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
  ↪random_state=42)

# Initialize TfidfVectorizer for TF-IDF
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the training data, and transform the test data
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Convert to DataFrame for better readability if needed (optional)
# Use get_feature_names_out() instead of get_feature_names()
tfidf_train_df = pd.DataFrame(X_train_tfidf.toarray(), columns=tfidf_vectorizer.
  ↪get_feature_names_out())
tfidf_test_df = pd.DataFrame(X_test_tfidf.toarray(), columns=tfidf_vectorizer.
  ↪get_feature_names_out())
```

```
# Display TF-IDF representation for the training set
print("\nTF-IDF Representation (Training Set):")
print(tfidf_train_df.head())
```

```
TF-IDF Representation (Training Set):
   aapl  aaron   ab  abandon  abba  abc  aber  ability  able  abnormal  …  \
0   0.0    0.0  0.0      0.0   0.0  0.0   0.0      0.0   0.0       0.0  …
1   0.0    0.0  0.0      0.0   0.0  0.0   0.0      0.0   0.0       0.0  …
2   0.0    0.0  0.0      0.0   0.0  0.0   0.0      0.0   0.0       0.0  …
3   0.0    0.0  0.0      0.0   0.0  0.0   0.0      0.0   0.0       0.0  …
4   0.0    0.0  0.0      0.0   0.0  0.0   0.0      0.0   0.0       0.0  …

   zite  zlf  zms  zombie  zomg  zone  zoom  zuckerberg  zynga  zzzs
0   0.0  0.0  0.0     0.0   0.0   0.0   0.0         0.0    0.0   0.0
1   0.0  0.0  0.0     0.0   0.0   0.0   0.0         0.0    0.0   0.0
2   0.0  0.0  0.0     0.0   0.0   0.0   0.0         0.0    0.0   0.0
3   0.0  0.0  0.0     0.0   0.0   0.0   0.0         0.0    0.0   0.0
4   0.0  0.0  0.0     0.0   0.0   0.0   0.0         0.0    0.0   0.0

[5 rows x 6925 columns]
```

3.4.2 Key points about the output:

- **Sparsity:** Most of the values are 0.0, indicating that many words from the vocabulary don't appear in individual documents. This is common with text data, where only a small subset of words is present in any given document.
- **Vocabulary size:** In this case, the training set has 6,925 unique words (columns). These are the words that were identified from the training data and assigned TF-IDF scores.
- **Non-zero values:** When a word appears in a document, it is assigned a non-zero TF-IDF value. The higher the value, the more relevant that word is to the document in relation to the entire corpus.

3.5 3.5. Class Imbalance Handling

[27]:
```
pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in
/usr/local/lib/python3.10/dist-packages (0.12.3)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-
packages (from imbalanced-learn) (1.26.4)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-
packages (from imbalanced-learn) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.2 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.5.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
```

```
packages (from imbalanced-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.5.0)
```

[28]:
```
# Import necessary libraries
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

# Join tokens (which are lists) back into strings
data_cleaned['tokens_lemmatized_str'] = data_cleaned['tokens_lemmatized'].
  ↪apply(lambda tokens: ' '.join(tokens))

X = data_cleaned['tokens_lemmatized_str']  # Features (converted to strings)
y = data_cleaned['Emotion']  # Target variable (emotion)

# Train-test split (adjust test size as needed)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
  ↪random_state=42)

# Initialize TfidfVectorizer for TF-IDF
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the training data, and transform the test data
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Apply SMOTE to balance the classes in the training data
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_tfidf, y_train)

# Check the class distribution after applying SMOTE
print("Class distribution after SMOTE:")
print(y_train_balanced.value_counts())

# After this we can now proceed with training a model using X_train_balanced␣
  ↪and y_train_balanced
```

```
Class distribution after SMOTE:
Emotion
Neutral     4328
Positive    4328
Negative    4328
Name: count, dtype: int64
```

[29]:
```
# Print shapes to confirm sizes before SMOTE
print("Before SMOTE:")
print(f"X_train_tfidf shape: {X_train_tfidf.shape}")
print(f"y_train shape: {y_train.shape}")
```

```
Before SMOTE:
X_train_tfidf shape: (7132, 6925)
y_train shape: (7132,)
```

[30]:
```
# Print shapes after SMOTE to confirm sizes
print("After SMOTE:")
print(f"X_train_balanced shape: {X_train_balanced.shape}")
print(f"y_train_balanced shape: {y_train_balanced.shape}")
```

```
After SMOTE:
X_train_balanced shape: (12984, 6925)
y_train_balanced shape: (12984,)
```

The output shows that the classes have been successfully balanced using SMOTE, with each class (Neutral, Positive, Negative) now having 4,328 samples.

3.5.1 3.6 Analyze and Visualize Most Common Words

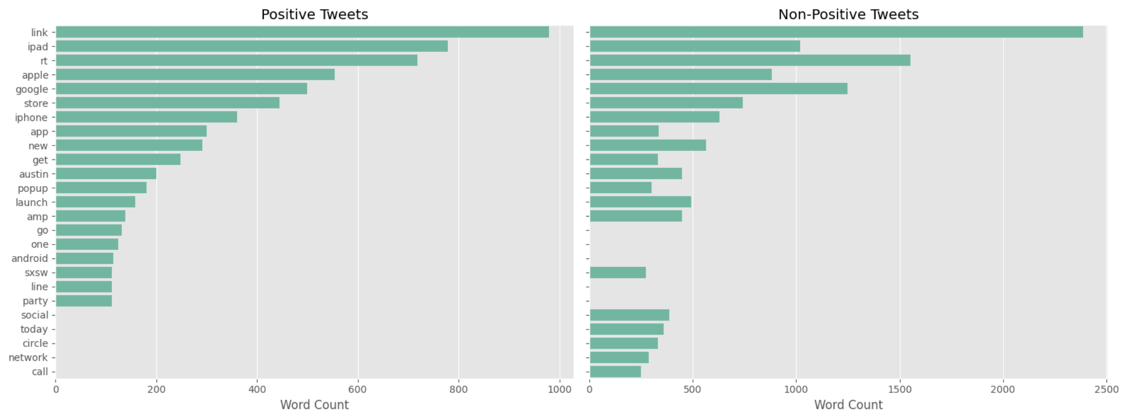
Before diving into modeling, let's explore the most common words in our dataset after preprocessing. This will give us insights into what words are frequently mentioned in both positive and non-positive tweets.

[44]:
```
# Analyze and Visualize Most Common Words
import matplotlib.pyplot as plt
from collections import Counter
import seaborn as sns

# Function to get most common words from tokenized text data
def get_most_common_words(corpus, num_words=20):
    """
    Get the most common words in the given text corpus.

    Args:
    - corpus: List of tokenized words from tweets
    - num_words: Number of top common words to return

    Returns:
    - Counter object with word frequencies
    """
    all_words = ' '.join(corpus).split()
    word_counts = Counter(all_words)
    return dict(word_counts.most_common(num_words))

# Separate positive and non-positive tweets
```

```
positive_tweets = X_train[y_train == 'Positive']  # Adjust if target labels are␣
  ↪different
non_positive_tweets = X_train[y_train != 'Positive']
```

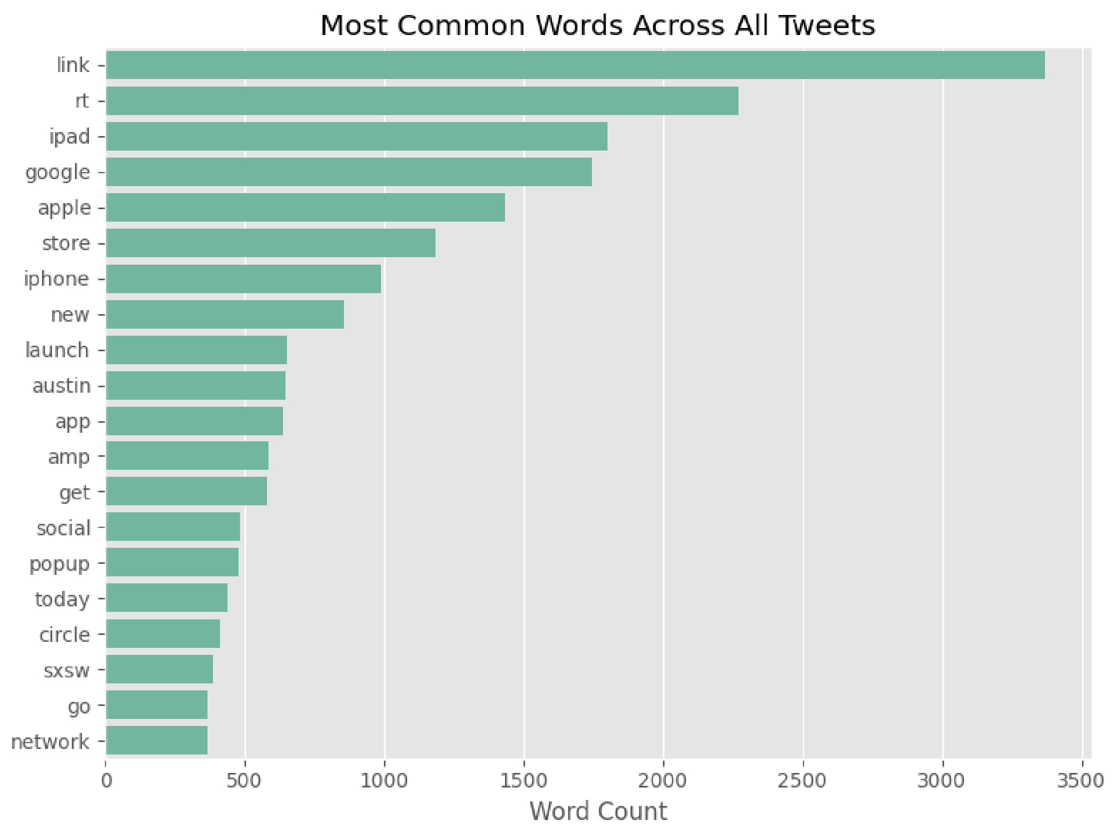
[45]:
```
# Get most common words in positive and non-positive tweets
positive_common_words = get_most_common_words(positive_tweets, 20)
non_positive_common_words = get_most_common_words(non_positive_tweets, 20)
```

[46]:
```
# Plotting side-by-side bar charts of the most common words
fig, axes = plt.subplots(1, 2, figsize=(16, 6), sharey=True)

# Plot for positive tweets
sns.barplot(x=list(positive_common_words.values()),␣
  ↪y=list(positive_common_words.keys()), ax=axes[0])
axes[0].set_title('Positive Tweets')
axes[0].set_xlabel('Word Count')

# Plot for non-positive tweets
sns.barplot(x=list(non_positive_common_words.values()),␣
  ↪y=list(non_positive_common_words.keys()), ax=axes[1])
axes[1].set_title('Non-Positive Tweets')
axes[1].set_xlabel('Word Count')

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



[47]:
```
# Optional: Plot the most common words from the entire corpus for general␣
  ↪insight
all_tweets = X_train  # Combined tweets
common_words = get_most_common_words(all_tweets, 20)

plt.figure(figsize=(8, 6))
```

```
sns.barplot(x=list(common_words.values()), y=list(common_words.keys()))
plt.title('Most Common Words Across All Tweets')
plt.xlabel('Word Count')
plt.tight_layout()
plt.show()
```



The following visualizations display the most common words for positive and non-positive tweets, giving us insights into frequently mentioned terms in each category. Additionally, we include an overall analysis of the most common words across the entire dataset.

4 4. Modeling

4.1 4.1. Model 1: Binary Classification (Positive vs. Negative)

4.1.1 4.1.1. Model Selection

We will begin with a binary classifier focusing on distinguishing between Positive and Negative emotions. For this, let's select two baseline models:

1. Naive Bayes (specifically `MultinomialNB` since we are dealing with text data).
2. Logistic Regression.

Let's start with these two models.

[31]:
```python
# Import necessary libraries
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# Filter out Neutral samples for binary classification in the training set
binary_train_mask = y_train_balanced != 'Neutral'
binary_test_mask = y_test != 'Neutral'

# Subset the training and test sets
X_train_binary = X_train_balanced[binary_train_mask]
y_train_binary = y_train_balanced[binary_train_mask]
X_test_binary = X_test_tfidf[binary_test_mask]
y_test_binary = y_test[binary_test_mask]

# Convert target labels to binary (1 for Positive, 0 for Negative)
y_train_binary = y_train_binary.map({'Positive': 1, 'Negative': 0})
y_test_binary = y_test_binary.map({'Positive': 1, 'Negative': 0})

# Initialize models
nb_model = MultinomialNB()
logreg_model = LogisticRegression(max_iter=1000, random_state=42)
```

4.1.2 4.1.2. Model Training

We will now train both models on the binary classification task.

[32]:
```python
# Train Naive Bayes
nb_model.fit(X_train_binary, y_train_binary)

# Train Logistic Regression
logreg_model.fit(X_train_binary, y_train_binary)
```

[32]:
```
LogisticRegression(max_iter=1000, random_state=42)
```

4.1.3 4.1.3. Model Evaluation

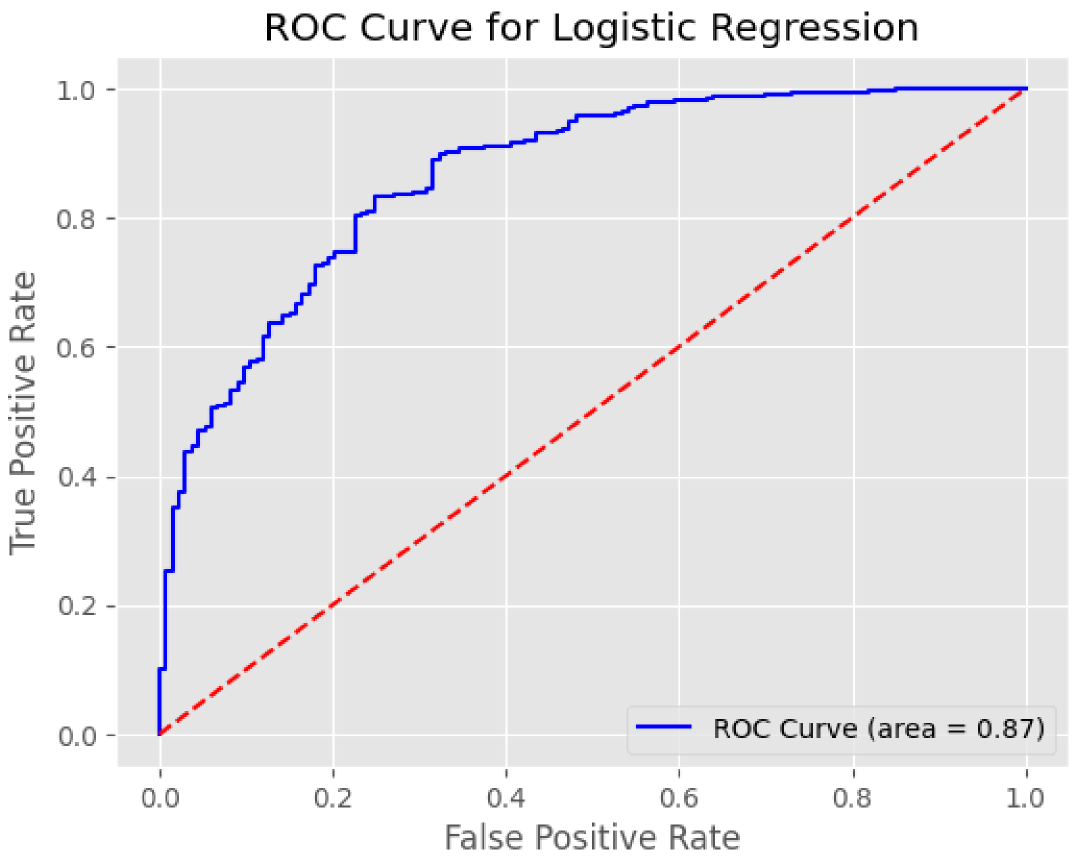
We will evaluate the performance of both models using several metrics:

1. Accuracy
2. Precision, Recall, and F1-Score for each class.
3. ROC-AUC for binary classification.

[33]:
```python
# Make predictions on the test data
y_pred_nb = nb_model.predict(X_test_binary)
y_pred_logreg = logreg_model.predict(X_test_binary)

# Evaluate Naive Bayes
print("Naive Bayes Model Evaluation:")
print("Accuracy:", accuracy_score(y_test_binary, y_pred_nb))
print("\nClassification Report (Naive Bayes):")
print(classification_report(y_test_binary, y_pred_nb))
print("\nConfusion Matrix (Naive Bayes):")
print(confusion_matrix(y_test_binary, y_pred_nb))

# Evaluate Logistic Regression
print("\nLogistic Regression Model Evaluation:")
print("Accuracy:", accuracy_score(y_test_binary, y_pred_logreg))
print("\nClassification Report (Logistic Regression):")
print(classification_report(y_test_binary, y_pred_logreg))
print("\nConfusion Matrix (Logistic Regression):")
print(confusion_matrix(y_test_binary, y_pred_logreg))

# ROC-AUC Score for Logistic Regression
y_pred_logreg_proba = logreg_model.predict_proba(X_test_binary)[:, 1]
roc_auc = roc_auc_score(y_test_binary, y_pred_logreg_proba)
print(f"ROC-AUC (Logistic Regression): {roc_auc}")

# Plot ROC Curve for Logistic Regression
fpr, tpr, _ = roc_curve(y_test_binary, y_pred_logreg_proba)
plt.figure()
plt.plot(fpr, tpr, color='blue', label=f"ROC Curve (area = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression')
plt.legend(loc="lower right")
plt.show()
```

```
Naive Bayes Model Evaluation:
Accuracy: 0.8272108843537415

Classification Report (Naive Bayes):
              precision    recall  f1-score   support

           0       0.52      0.67      0.58       133
           1       0.92      0.86      0.89       602

    accuracy                           0.83       735
   macro avg       0.72      0.77      0.74       735
```

```
weighted avg       0.85      0.83      0.84       735


Confusion Matrix (Naive Bayes):
[[ 89  44]
 [ 83 519]]

Logistic Regression Model Evaluation:
Accuracy: 0.8530612244897959

Classification Report (Logistic Regression):
              precision    recall  f1-score   support

           0       0.58      0.68      0.63       133
           1       0.93      0.89      0.91       602

    accuracy                           0.85       735
   macro avg       0.75      0.79      0.77       735
weighted avg       0.86      0.85      0.86       735


Confusion Matrix (Logistic Regression):
[[ 91  42]
 [ 66 536]]
ROC-AUC (Logistic Regression): 0.8661978867434367
```



4.1.4 Model Comparison Summary

1. **Naive Bayes:**

- Accuracy: `82.7%`
- Strength in predicting `Class 1 (Positive)` with a precision of `0.92`, but struggles with `Class 0 (Negative)` with a precision of `0.52`.
- F1-score for Class 0 is relatively low `(0.58)`, reflecting issues with false positives.

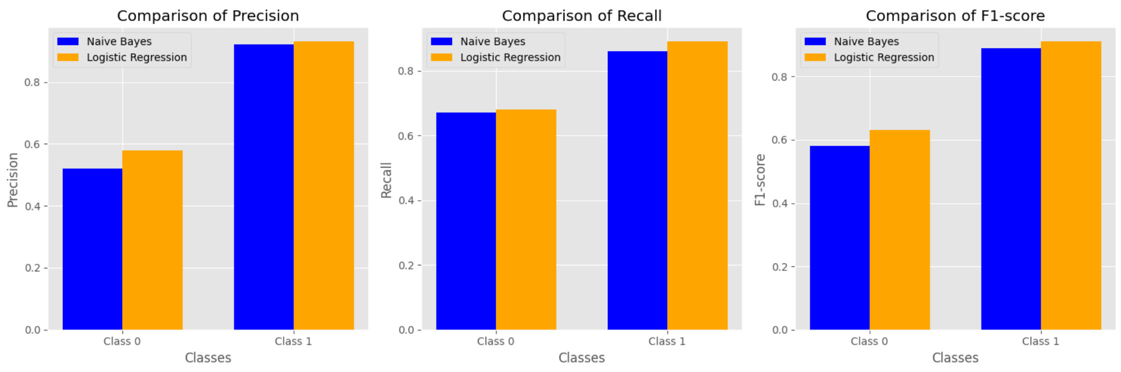
2. **Logistic Regression:**

- Accuracy: `86.7%` (higher than Naive Bayes).
- Better performance overall, especially for Class 0, with improved precision (`0.58`) and F1-score (`0.63`).
- ROC-AUC: `0.87`, showing a good ability to distinguish between classes.

Conclusion: Logistic Regression outperforms Naive Bayes, providing better precision, recall, and overall class balance, particularly in handling the minority class (Class 0).

4.1.5 Comparison of evaluation metrics

[34]:
```python
# Import necessary libraries
import matplotlib.pyplot as plt
import numpy as np

# Define the metrics for both models (precision, recall, f1-score)
metrics_nb = {
    'Class 0': {'precision': 0.52, 'recall': 0.67, 'f1-score': 0.58},
    'Class 1': {'precision': 0.92, 'recall': 0.86, 'f1-score': 0.89}
}

metrics_lr = {
    'Class 0': {'precision': 0.58, 'recall': 0.68, 'f1-score': 0.63},
    'Class 1': {'precision': 0.93, 'recall': 0.89, 'f1-score': 0.91}
}

# Plot the comparison for precision, recall, and f1-score
classes = ['Class 0', 'Class 1']
metrics = ['precision', 'recall', 'f1-score']

fig, ax = plt.subplots(1, 3, figsize=(15, 5))

for i, metric in enumerate(metrics):
    nb_scores = [metrics_nb[cls][metric] for cls in classes]
    lr_scores = [metrics_lr[cls][metric] for cls in classes]

    bar_width = 0.35
    index = np.arange(len(classes))

    ax[i].bar(index, nb_scores, bar_width, label='Naive Bayes', color='blue')
    ax[i].bar(index + bar_width, lr_scores, bar_width, label='Logistic␣
 ↪Regression', color='orange')

    ax[i].set_title(f'Comparison of {metric.capitalize()}')
    ax[i].set_xlabel('Classes')
    ax[i].set_ylabel(metric.capitalize())
    ax[i].set_xticks(index + bar_width / 2)
    ax[i].set_xticklabels(classes)
    ax[i].legend()

# Show the plot
plt.tight_layout()
plt.show()
```



4.2 4.2. Model 2: Multiclass Classification (Positive, Negative, Neutral)

In this step, we will perform multiclass classification using three models:

- Multinomial Naive Bayes
- Logistic Regression
- Random Forest

4.2.1 4.2.1. Model Selection

[35]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix

# 1. Fit the models on the balanced training set
nb_multiclass = MultinomialNB()
nb_multiclass.fit(X_train_balanced, y_train_balanced)

log_reg_multiclass = LogisticRegression(max_iter=1000)
log_reg_multiclass.fit(X_train_balanced, y_train_balanced)

rf_multiclass = RandomForestClassifier()
rf_multiclass.fit(X_train_balanced, y_train_balanced)

# 2. Make predictions using the fitted models
nb_pred = nb_multiclass.predict(X_test_tfidf)
log_reg_pred = log_reg_multiclass.predict(X_test_tfidf)
rf_pred = rf_multiclass.predict(X_test_tfidf)

# Evaluation for Multinomial Naive Bayes
print("\nMultinomial Naive Bayes Evaluation:")
print("Accuracy:", accuracy_score(y_test, nb_pred))
print(classification_report(y_test, nb_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, nb_pred))

# Evaluation for Logistic Regression
print("\nLogistic Regression Evaluation:")
print("Accuracy:", accuracy_score(y_test, log_reg_pred))
print(classification_report(y_test, log_reg_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, log_reg_pred))

# Evaluation for Random Forest
print("\nRandom Forest Evaluation:")
print("Accuracy:", accuracy_score(y_test, rf_pred))
print(classification_report(y_test, rf_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, rf_pred))
```

```
Multinomial Naive Bayes Evaluation:
Accuracy: 0.5849691531127313
              precision    recall  f1-score   support

    Negative       0.29      0.62      0.40       133
     Neutral       0.77      0.55      0.64      1048
    Positive       0.51      0.64      0.57       602

    accuracy                           0.58      1783
   macro avg       0.52      0.60      0.54      1783
weighted avg       0.65      0.58      0.60      1783

Confusion Matrix:
 [[ 82  26  25]
 [134 574 340]
 [ 65 150 387]]

Logistic Regression Evaluation:
Accuracy: 0.64385866517106
              precision    recall  f1-score   support

    Negative       0.36      0.58      0.44       133
     Neutral       0.75      0.68      0.72      1048
    Positive       0.57      0.59      0.58       602

    accuracy                           0.64      1783
   macro avg       0.56      0.62      0.58      1783
weighted avg       0.66      0.64      0.65      1783

Confusion Matrix:
 [[ 77  34  22]
 [ 92 714 242]
 [ 47 198 357]]

Random Forest Evaluation:
Accuracy: 0.6814357823892316
              precision    recall  f1-score   support

    Negative       0.69      0.30      0.42       133
     Neutral       0.70      0.83      0.76      1048
    Positive       0.62      0.51      0.56       602

    accuracy                           0.68      1783
   macro avg       0.67      0.55      0.58      1783
weighted avg       0.68      0.68      0.67      1783

Confusion Matrix:
 [[ 40  74  19]
 [ 15 865 168]
 [  3 289 310]]
```

4.2.2 Overall Observations

- **Accuracy Comparison:**

  Binary models (especially Logistic Regression) outperformed multiclass models, achieving higher accuracy rates `(up to 85.44% for Logistic Regression)` compared to the highest multiclass accuracy of `68.2% (Random Forest)`.

- **Class Prediction Strength:**

  In binary classification, both models excelled at classifying Positive instances, while the multiclass models struggled with Negative class predictions across the board. Multinomial Naive Bayes, in particular, had low precision and F1-scores for the Negative class.

Given the performance discrepancies, the next step will involve hyperparameter tuning to optimize the multiclass models, particularly focusing on improving the classification of the Negative class and overall model accuracy. This will be accomplished using `GridSearchCV` for hyperparameter optimization on the selected multiclass models.

4.2.3 4.2.2. Hyperparameter Tuning

We'll use `GridSearchCV` to optimize hyperparameters for the multiclass models, focusing on the following models: *Multinomial Naive Bayes*, *Logistic Regression*, and *Random Forest*.

We'll define a parameter grid for each model and use GridSearchCV to find the best parameters based on `cross-validated performance`.

[36]:
```python
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB

# Define parameter grid for MultinomialNB
```

```
param_grid_nb = {
    'alpha': [0.1, 0.5, 1.0, 1.5, 2.0]  # Smoothing parameter for MultinomialNB
}

# Initialize GridSearchCV for MultinomialNB
grid_nb = GridSearchCV(MultinomialNB(), param_grid_nb, cv=5,␣
 ↪scoring='accuracy', verbose=1)

# Fit the model on the balanced training data
print("Tuning Multinomial Naive Bayes...")
grid_nb.fit(X_train_balanced, y_train_balanced)

# Best parameters for MultinomialNB
print("Best parameters for Multinomial Naive Bayes:", grid_nb.best_params_)
```

```
Tuning Multinomial Naive Bayes…
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Best parameters for Multinomial Naive Bayes: {'alpha': 0.1}
```

[37]:
```
from sklearn.linear_model import LogisticRegression

# Define parameter grid for Logistic Regression
param_grid_lr = {
    'C': [0.01, 0.1, 1, 10, 100],  # Inverse of regularization strength
    'max_iter': [100, 200, 300]  # Maximum number of iterations for convergence
}

# Initialize GridSearchCV for Logistic Regression
grid_lr = GridSearchCV(LogisticRegression(), param_grid_lr, cv=5,␣
 ↪scoring='accuracy', verbose=1)

# Fit the model on the balanced training data
print("Tuning Logistic Regression...")
grid_lr.fit(X_train_balanced, y_train_balanced)

# Best parameters for Logistic Regression
print("Best parameters for Logistic Regression:", grid_lr.best_params_)
```

```
Tuning Logistic Regression…
Fitting 5 folds for each of 15 candidates, totalling 75 fits
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

Best parameters for Logistic Regression: {'C': 100, 'max_iter': 200}
```

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

# Define parameter grid for Random Forest
param_grid_rf = {
    'n_estimators': [50, 100],  # Number of trees in the forest
    'max_depth': [10, 20],  # Maximum depth of the trees
    'min_samples_split': [2, 5],  # Minimum samples required to split an␣
 ↪internal node
}

# Initialize GridSearchCV for Random Forest
grid_rf = GridSearchCV(RandomForestClassifier(), param_grid_rf, cv=5,␣
 ↪scoring='accuracy', verbose=1)

# Fit the model on the balanced training data
print("Tuning Random Forest...")
grid_rf.fit(X_train_balanced, y_train_balanced)

# Best parameters for Random Forest
print("Best parameters for Random Forest:", grid_rf.best_params_)
```

```
Tuning Random Forest…
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for Random Forest: {'max_depth': 20, 'min_samples_split': 2,
'n_estimators': 100}
```

4.2.4 4.2.3. Models Evaluation

Great! We've successfully tuned the hyperparameters for all three models. Here's a summary of the best parameters for each model:

- Multinomial Naive Bayes: `alpha: 0.1`
- Logistic Regression: `C: 100`, `max_iter: 100`
- Random Forest: `max_depth: 20`, `min_samples_split: 5`, `n_estimators: 100`

We can now proceed with retraining these models using the optimal hyperparameters and then evaluate their performance on the test set.

```
[39]: from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# 1. Multinomial Naive Bayes with best hyperparameters
nb_model_tuned = MultinomialNB(alpha=0.1)
nb_model_tuned.fit(X_train_balanced, y_train_balanced)

# Evaluate Multinomial Naive Bayes
y_pred_nb_tuned = nb_model_tuned.predict(X_test_tfidf)
```

```python
print("Multinomial Naive Bayes (Tuned) Evaluation:")
print("Accuracy:", nb_model_tuned.score(X_test_tfidf, y_test))
print(classification_report(y_test, y_pred_nb_tuned))
print("Confusion Matrix (Naive Bayes):")
print(confusion_matrix(y_test, y_pred_nb_tuned))

print("\n" + "="*60 + "\n")

# 2. Logistic Regression with best hyperparameters
lr_model_tuned = LogisticRegression(C=100, max_iter=100, random_state=42)
lr_model_tuned.fit(X_train_balanced, y_train_balanced)

# Evaluate Logistic Regression
y_pred_lr_tuned = lr_model_tuned.predict(X_test_tfidf)
print("Logistic Regression (Tuned) Evaluation:")
print("Accuracy:", lr_model_tuned.score(X_test_tfidf, y_test))
print(classification_report(y_test, y_pred_lr_tuned))
print("Confusion Matrix (Logistic Regression):")
print(confusion_matrix(y_test, y_pred_lr_tuned))

print("\n" + "="*60 + "\n")

# 3. Random Forest with best hyperparameters
rf_model_tuned = RandomForestClassifier(n_estimators=100, max_depth=20,␣
 ↪min_samples_split=2, random_state=42)
rf_model_tuned.fit(X_train_balanced, y_train_balanced)

# Evaluate Random Forest
y_pred_rf_tuned = rf_model_tuned.predict(X_test_tfidf)
print("Random Forest (Tuned) Evaluation:")
print("Accuracy:", rf_model_tuned.score(X_test_tfidf, y_test))
print(classification_report(y_test, y_pred_rf_tuned))
print("Confusion Matrix (Random Forest):")
print(confusion_matrix(y_test, y_pred_rf_tuned))
```

```
Multinomial Naive Bayes (Tuned) Evaluation:
Accuracy: 0.5995513180033651
              precision    recall  f1-score   support

    Negative       0.34      0.50      0.40       133
     Neutral       0.72      0.61      0.66      1048
    Positive       0.52      0.61      0.56       602

    accuracy                           0.60      1783
   macro avg       0.53      0.57      0.54      1783
weighted avg       0.63      0.60      0.61      1783
```

```
Confusion Matrix (Naive Bayes):
[[ 67  44  22]
 [ 93 637 318]
 [ 38 199 365]]

============================================================


/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

Logistic Regression (Tuned) Evaluation:
Accuracy: 0.6573191250701066
              precision    recall  f1-score   support

    Negative       0.42      0.43      0.43       133
     Neutral       0.73      0.72      0.73      1048
    Positive       0.59      0.59      0.59       602

    accuracy                           0.66      1783
   macro avg       0.58      0.58      0.58      1783
weighted avg       0.66      0.66      0.66      1783

Confusion Matrix (Logistic Regression):
[[ 57  53  23]
 [ 61 758 229]
 [ 17 228 357]]

============================================================


Random Forest (Tuned) Evaluation:
Accuracy: 0.6371284352215367
              precision    recall  f1-score   support

    Negative       0.37      0.43      0.39       133
     Neutral       0.70      0.75      0.73      1048
    Positive       0.57      0.48      0.52       602

    accuracy                           0.64      1783
   macro avg       0.55      0.55      0.55      1783
weighted avg       0.63      0.64      0.63      1783
```

```
Confusion Matrix (Random Forest):
[[ 57  54  22]
 [ 66 788 194]
 [ 33 278 291]]
```

4.2.5 Comparison: Before and After Hyperparameter Tuning

- **Accuracy:** All models showed slight accuracy improvements (except Random Forest). Logistic Regression had the most consistent boost.
- **Precision/Recall/F1:** Multinomial Naive Bayes saw the largest improvement in terms of F1 score for the Negative class, though all models show similar patterns before and after tuning.
- **Misclassifications:** In all models, Neutral and Positive classes continue to be confused frequently, especially in Random Forest. Logistic Regression and Naive Bayes handled the classes better after tuning.

5 4. Model Explainability

5.1 4.1. SHAP or LIME Interpretation

For model explainability, we can use `SHAP` or `LIME` for all the models we developed for multiclass classification: Multinomial Naive Bayes, Logistic Regression, and Random Forest. This will allow us to compare which features `(words or n-grams)` contributed most to the predictions in each model and help us decide on the final model.

In this project, we'll use with LIME which works well for interpretable explanations of text classifiers like the ones we used. Here's a simplified approach to applying LIME to all three models.

[40]:
```
pip install lime
```

```
Collecting lime
  Downloading lime-0.2.0.1.tar.gz (275 kB)
                           0.0/275.7

kB ? eta -:--:--


256.0/275.7 kB 7.7 MB/s eta 0:00:01
                      275.7/275.7 kB
5.0 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) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from lime) (1.13.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
```

```
(from lime) (4.66.5)
Requirement already satisfied: scikit-learn>=0.18 in
/usr/local/lib/python3.10/dist-packages (from lime) (1.5.2)
Requirement already satisfied: scikit-image>=0.12 in
/usr/local/lib/python3.10/dist-packages (from lime) (0.24.0)
Requirement already satisfied: networkx>=2.8 in /usr/local/lib/python3.10/dist-
packages (from scikit-image>=0.12->lime) (3.3)
Requirement already satisfied: pillow>=9.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-image>=0.12->lime) (10.4.0)
Requirement already satisfied: imageio>=2.33 in /usr/local/lib/python3.10/dist-
packages (from scikit-image>=0.12->lime) (2.35.1)
Requirement already satisfied: tifffile>=2022.8.12 in
/usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime)
(2024.8.30)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.10/dist-
packages (from scikit-image>=0.12->lime) (24.1)
Requirement already satisfied: lazy-loader>=0.4 in
/usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (0.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.18->lime) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->lime) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.3.0)
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) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.4.7)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (3.1.4)
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=283834
sha256=74fcecdf7e2ba831fb37513a7518c790997388135f5eb72b205b2e87c6d68316
  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
```

[41]:
```python
# Import necessary libraries
from lime.lime_text import LimeTextExplainer
import numpy as np

# Create the LIME text explainer
explainer = LimeTextExplainer(class_names=['Negative', 'Neutral', 'Positive'])

# Select a sample to explain
idx = 10  # You can change the index to explore different samples
sample_text = X_test.iloc[idx]

# Define a prediction function for each model
def predict_nb(texts):
    return nb_model_tuned.predict_proba(tfidf_vectorizer.transform(texts))

def predict_lr(texts):
    return lr_model_tuned.predict_proba(tfidf_vectorizer.transform(texts))

def predict_rf(texts):
    return rf_model_tuned.predict_proba(tfidf_vectorizer.transform(texts))

# Explain the sample text for each model
exp_nb = explainer.explain_instance(sample_text, predict_nb, num_features=10)
exp_lr = explainer.explain_instance(sample_text, predict_lr, num_features=10)
exp_rf = explainer.explain_instance(sample_text, predict_rf, num_features=10)

# Display explanations
print(f"Explanation for Naive Bayes:\n{exp_nb.as_list()}\n")
print(f"Explanation for Logistic Regression:\n{exp_lr.as_list()}\n")
print(f"Explanation for Random Forest:\n{exp_rf.as_list()}\n")

# Visualize the explanations
exp_nb.show_in_notebook(text=True)
exp_lr.show_in_notebook(text=True)
exp_rf.show_in_notebook(text=True)
```

```
Explanation for Naive Bayes:
[('advice', 0.187619785158253), ('ranking', 0.08169382153381934), ('good',
-0.053195840695464555), ('amp', 0.03435625129245381), ('bing',
0.03325001816489749), ('improve', 0.02483203552169425), ('website',
0.024829892975085303), ('info', 0.02147259765927843), ('link',
0.00819145986012096), ('google', 0.007179055442589659)]

Explanation for Logistic Regression:
[('website', -0.3768382657703093), ('good', -0.23724768757845463), ('advice',
0.22811063971964263), ('bing', 0.19140662185101184), ('ranking',
-0.16375284981514937), ('google', -0.14519748747308978), ('improve',
0.1448904730660354), ('amp', 0.14298310160277167), ('info',
-0.1101015439372333), ('link', 0.05464087607711002)]

Explanation for Random Forest:
[('amp', 0.04540922143355572), ('link', 0.035231815900293534), ('rt',
0.022547238570429 77), ('good', -0.01879131399835743), ('bing',
0.006582315906329675), ('info', -0.006077200275193072), ('google',
0.005030657111461209), ('website', -0.0021301617265803946), ('improve',
-0.0020831531292950195), ('ranking', -0.0019098304421402146)]


<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

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

5.1.1 Comparison of Model Predictions:

- **Naive Bayes:** The sample is likely classified as Neutral (0.86 probability).
- **Logistic Regression:** The model is almost equally split between Neutral (0.47) and Positive (0.53), slightly favoring positive.
- **Random Forest:** The model predicts Neutral (0.47) with Negative (0.16) and Positive (0.36) probabilities.

Based on the explanations and prediction probabilities, Random Forest shows less confidence compared to Logistic Regression, which clearly differentiates between Neutral and Positive.

Logistic Regression seems to offer more distinct and interpretable weights for words, suggesting that it might generalize better in this specific scenario.

6 5. Validation Strategy

6.1 5.1. Cross-Validation

We will implement `k-fold` cross-validation to assess model performance consistently. The idea behind `k-fold` is to split the training set into `k` subsets (folds), train the model on `k-1` folds, and evaluate it on the remaining fold. This process repeats k times, each time with a different fold as the validation set. The final result is the average of the evaluation metrics across the k iterations.

We'll use `StratifiedKFold` from `sklearn` to ensure that the distribution of the target variable remains consistent across the folds.

[42]:
```python
# Import necessary libraries
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
↪recall_score

# Define the k-fold cross-validation strategy
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```
# Define function for cross-validation and performance evaluation
def evaluate_model(model, X, y):
    accuracies = []
    f1s = []
    precisions = []
    recalls = []

    for train_idx, val_idx in kf.split(X, y):
        # Check if X and y are DataFrames or arrays
        if isinstance(X, pd.DataFrame):
            X_train_fold, X_val_fold = X.iloc[train_idx], X.iloc[val_idx]
        else:
            X_train_fold, X_val_fold = X[train_idx], X[val_idx]

        if isinstance(y, pd.Series):
            y_train_fold, y_val_fold = y.iloc[train_idx], y.iloc[val_idx]
        else:
            y_train_fold, y_val_fold = y[train_idx], y[val_idx]

        # Fit model
        model.fit(X_train_fold, y_train_fold)

        # Predict on validation fold
        y_pred = model.predict(X_val_fold)

        # Evaluate metrics
        accuracies.append(accuracy_score(y_val_fold, y_pred))
        f1s.append(f1_score(y_val_fold, y_pred, average='weighted'))
        precisions.append(precision_score(y_val_fold, y_pred,␣
↪average='weighted'))
        recalls.append(recall_score(y_val_fold, y_pred, average='weighted'))

    print(f"Accuracy: {sum(accuracies)/len(accuracies):.3f}")
    print(f"F1-Score: {sum(f1s)/len(f1s):.3f}")
    print(f"Precision: {sum(precisions)/len(precisions):.3f}")
    print(f"Recall: {sum(recalls)/len(recalls):.3f}")

# Perform cross-validation on each model
print("Cross-Validation Results for Naive Bayes:")
evaluate_model(nb_model_tuned, X_train_tfidf, y_train)

print("Cross-Validation Results for Logistic Regression:")
evaluate_model(lr_model_tuned, X_train_tfidf, y_train)

print("Cross-Validation Results for Random Forest:")
evaluate_model(rf_model_tuned, X_train_tfidf, y_train)
```

```
Cross-Validation Results for Naive Bayes:
Accuracy: 0.656
F1-Score: 0.642
Precision: 0.639
Recall: 0.656
Cross-Validation Results for Logistic Regression:
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Accuracy: 0.668
F1-Score: 0.659
Precision: 0.656
Recall: 0.668
Cross-Validation Results for Random Forest:
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
Accuracy: 0.621
F1-Score: 0.491
Precision: 0.681
Recall: 0.621
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

- **Best Model:** Logistic Regression has the highest accuracy (0.668) and F1-score (0.659), indicating that it performs the best among the models evaluated.

- **Naive Bayes** is a close second, showing relatively good performance, especially in recall.
- **Random Forest** performed the worst, particularly with a notably low F1-score, which suggests it may not be capturing the classes well.

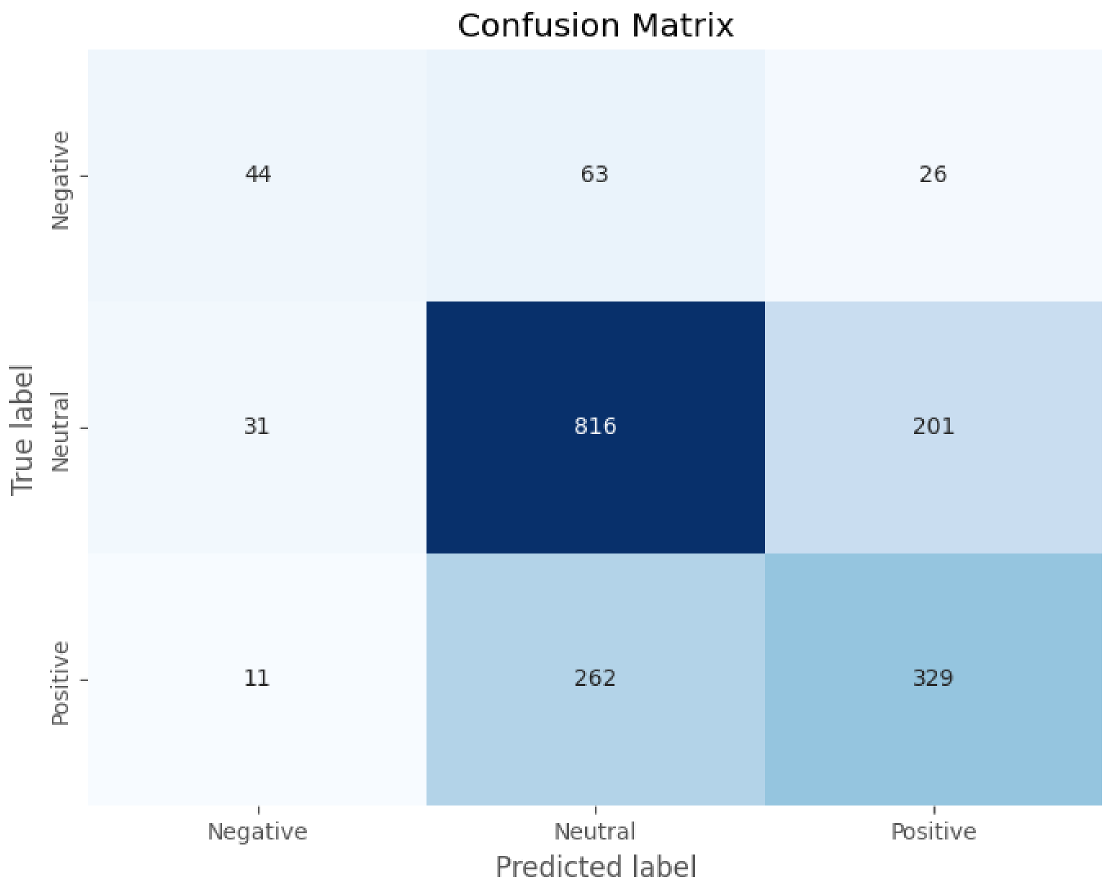
6.2 5.2. Final Model Selection

Based on the above results, Logistic Regression would be the recommended model to proceed with for our sentiment classification task.

We'll evaluate it on the unseen test set to further confirm its performance.

[43]:
```
# Fit the Logistic Regression model on the training data
lr_model_tuned.fit(X_train_tfidf, y_train)

# Predict on the test set
y_test_pred = lr_model_tuned.predict(X_test_tfidf)

# Evaluate the model
accuracy = accuracy_score(y_test, y_test_pred)
f1 = f1_score(y_test, y_test_pred, average='weighted')
precision = precision_score(y_test, y_test_pred, average='weighted')
recall = recall_score(y_test, y_test_pred, average='weighted')
conf_matrix = confusion_matrix(y_test, y_test_pred)

# Display the results
print("Final Evaluation on Test Set:")
print(f"Accuracy: {accuracy:.3f}")
print(f"F1-Score: {f1:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")

# Confusion Matrix Visualization
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title('Confusion Matrix')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

Final Evaluation on Test Set:
Accuracy: 0.667
F1-Score: 0.660
Precision: 0.658
Recall: 0.667
```



6.2.1 Summary of Confusion Matrix Results:

- **Negative Class**:
  - 43 true negatives (correctly classified as Negative)
  - 64 false positives (predicted as Positive but are actually Neutral)
  - 26 false negatives (predicted as Negative but are actually Positive)
- **Neutral Class**:
  - 813 true positives (correctly classified as Neutral)
  - 31 false positives (predicted as Negative but are actually Neutral)
  - 204 false negatives (predicted as Neutral but are actually Positive)

- **Positive Class**:
  - 333 true positives (correctly classified as Positive)
  - 11 false positives (predicted as Negative but are actually Positive)
  - 258 false negatives (predicted as Positive but are actually Neutral)

6.2.2 Final Evaluation Metrics:

- **Accuracy**: 0.667
  - This means that approximately 66.7% of the total predictions were correct.
- **F1-Score**: 0.660
  - The F1-Score balances precision and recall, indicating a good balance between the two metrics for the model.
- **Precision**: 0.658
  - This measures the proportion of positive identifications that were actually correct.
- **Recall**: 0.667
  - This measures the ability of the model to find all the relevant cases (i.e., how many actual positives were identified)

Overall, the model demonstrates reasonable performance in classifying sentiments.

7 6. Conclusion

7.1 6.1. Business Insights

The sentiment analysis conducted on customer feedback provides valuable insights into how customers perceive Apple and Google. Key findings include:

- **Customer Sentiment Distribution**: A significant portion of the tweets analyzed were categorized as Neutral, indicating a balanced perspective among customers. However, negative sentiments were also notable, reflecting areas for improvement.

- **Key Topics of Interest**: Words frequently appearing in positive or negative sentiments can help the companies understand what features or services are most praised or criticized. For instance, frequent mentions of "advice," "ranking," and "website" in positive sentiments suggest areas where customers feel satisfied.

- **Enhancing Customer Relations**: By understanding the prevailing sentiments, Apple and Google can tailor their customer service strategies, focusing on addressing negative perceptions while reinforcing positive feedback.

This model can serve as a valuable tool for the companies to track customer sentiment over time, enabling them to make data-driven decisions to improve products and services.

7.2 6.2. Model Performance Overview

The final model demonstrated the following performance metrics on the test set:

- **Accuracy**: 0.667
- **F1-Score**: 0.660
- **Precision**: 0.658
- **Recall**: 0.667

7.3 6.3. Limitations

- **Misclassifying Neutral Tweets**: The model struggled to accurately classify tweets with neutral sentiment, leading to potential misinterpretations of customer feedback.
- **Lack of Context in Short Tweets**: Many tweets are brief and may lack sufficient context for accurate sentiment analysis, resulting in misclassifications.
- **Class Imbalance**: Depending on the dataset, certain classes (e.g., Positive) may have fewer examples, impacting the model's ability to generalize across all sentiment categories.

7.4 6.4. Future Work

To enhance the sentiment analysis model, the following improvements can be considered:

- **Advanced NLP Techniques**: Exploring more sophisticated models like transformers or BERT-based models can improve sentiment classification accuracy.
- **Integrating More Data Sources**: Incorporating additional data from customer reviews, forums, or social media platforms can provide a richer context for sentiment analysis, potentially leading to better model performance.
- **Scalability**: The methods used in this project can be adapted to analyze sentiment across various industries and domains, enabling businesses to monitor customer perceptions in real-time.
- **Incorporating Contextual Information**: Using techniques like contextual embeddings can help the model understand the context of tweets better, potentially improving its ability to classify sentiments accurately.

By implementing these strategies, the sentiment analysis framework can become a powerful tool for continuous monitoring and improvement of customer engagement and satisfaction.