CM3060 Natural Language Processing

Mid - Term Coursework - Sentiment Analysis on Spotify Application

Introduction

Domain - Specific Area

Sentiment analysis is a subfield of Natural Language Processing (NLP) that plays a pivotal role in extracting subjective information, including sentiments, emotions, and attitudes, from textual sources. This analysis has significant implications in today's digital landscape, where feedback and opinions abound across various online platforms. Such information provides invaluable insights into consumer attitudes toward products, services, and overall brand image, delivering substantial business value. (Coursera, 2023)

This project focuses on the domain of music streaming services, with a specific emphasis on the Spotify application. The digital music industry is characterized by its dynamic nature, rapid advancements, and intense competition. In 2022 alone, music streaming applications generated a remarkable revenue of **$17.5 billion worldwide**, setting an all-time record (Gotting M, 2023). This exponential growth reflects the increasing popularity and significance of music streaming services in the digital age.

Spotify, as the most popular streaming service, has garnered an impressive user base of over **350 million users and 150 million subscribers worldwide.** However, the competitive landscape is intense, with major players like Apple Music and Youtube vying for market share. While Apple Music may trail behind Spotify in overall user numbers, it maintains a competitive advantage in specific regions. YouTube, on the other hand, dominates in terms of total usage with two billion users but falls behind Spotify and Apple in terms of paid subscribers (Curry D, 2023).

Given the sheer magnitude of music streaming subscribers, which has surpassed **616.2 million users worldwide**, understanding user sentiment has become more critical than ever for the success of music streaming platforms, including Spotify (Gotting M, 2023). Analyzing user sentiment allows Spotify to gauge user satisfaction levels, identify areas for improvement, and make informed decisions to enhance the user experience.

In this context, sentiment analysis specifically tailored to the Spotify application holds immense potential. By delving into user reviews and feedback, Spotify can gain valuable insights into the strengths and weaknesses of its platform, enabling data-driven decision-making. This, in turn, can lead to the development and implementation of targeted marketing strategies, personalized recommendations, and tailored user experiences. Understanding user sentiment aids in benchmarking Spotify's performance against competitors, helping identify opportunities to differentiate and refine its services. This can attract and retain more users, increase user engagement, and ultimately drive revenue growth. The analysis of user sentiment contributes to the broader understanding of how music streaming platforms can deliver enhanced experiences. This insight can not only shape the strategic direction of Spotify but also advance the broader field of music streaming services.

References:

1. Coursera, "Intro to sentiment analysis", 2023.[Online].

Available: https://www.coursera.org/learn/uol-cm3060-natural-language-processing/lecture/ylmC2/intro-to-sentiment-analysis

2. Gotting M, "Music streaming worldwide - statistics & facts", 2023.[Online].

Available: https://www.statista.com/topics/6408/music-streaming/#topicOverview

3. Gotting M, "Music streaming revenue worldwide from 2005 to 2022 (in billion U.S. dollars)", 2023.[Online].

Available:https://www.statista.com/statistics/587216/music-streaming-revenue/

4. Curry D, "Music Streaming App Revenue and Usage Statistics (2023)", 2023.[Online].

Available:https://www.businessofapps.com/data/music-streaming-market/

Objectives of the Project

Assuming the role of a Spotify developer, the primary objectives of this project is to perform sentiment analysis using Natural Language Processing (NLP) techniques and build Machine Learning **classifier** models for text classification. The project aims to classify user reviews of the Spotify application as **either positive or negative**, providing valuable insights for enhancing the platform and meeting user expectations. The specific objectives include:

Data Preprocessing: Conducting a thorough analysis of the dataset to understand the different features and their requirements/importance for sentiment analysis. This phase involves identifying relevant features and any additional metadata. Understanding these features and their significance will help determine the key factors influencing user sentiment and guide the subsequent NLP-based sentiment analysis.

NLP-based Sentiment Analysis: Utilizing NLP techniques to preprocess the textual data from user reviews. Tasks such as tokenization, removing stop words, handling punctuation, and converting words to their base forms will be performed. This preprocessing stage is crucial to be able to extract the most value from the classifier models.

Building ML Classifier Models: Constructing multiple machine learning classifier models for sentiment analysis. Each model will be trained on the preprocessed dataset to classify user reviews of the Spotify application as positive or negative. This classification will help in gauging user sentiment and provide insights into areas of strength and areas requiring improvement within the application.

Performance Comparison: Evaluating and comparing the performances of the ML classifier models using suitable metrics such as **accuracy** and F1-score in this case. This analysis will help in identifying the most effective model for sentiment classification on the given dataset.

Conclusion - Reflective Evaluation: Analyzing and evaluating the results of the sentiment analysis and performance comparison. This will help to draw meaningful conclusions from the findings, including insights gained from the classification of reviews and the identification of valuable features. Reflection on the efficacy of the applied techniques and models will also be done, discussing their strengths, limitations, and potential areas for improvement.

The potential contributions of this project are multifold:

1. It provides a comprehensive analysis of user sentiment towards the Spotify application, enabling Spotify to enhance its features and services according to user preferences.
2. The comparison of ML classifier models establishes the most effective approach for sentiment analysis, offering insights into the performance and applicability of different models.
3. The identification of valuable features contributes to understanding of factors that shape positive and negative user experiences, assisting Spotify in targeted improvements.

Dataset

The dataset under consideration for this Sentiment Analysis project is titled "Spotify App Reviews 2022" which is curated and made available by Ilmi M on the Kaggle platform.

Ilmi M, 'Spotify App Reviews', 2022.[Online].
Available: https://www.kaggle.com/datasets/mfaaris/spotify-app-reviews-2022?resource=download

Description:
This dataset is an extensive compilation of over 61,000 reviews for the Spotify application, specifically sourced from the Google Play Store. The time frame for these reviews spans from the commencement of the year 2022 to July 9th of the same year. Each entry in the dataset corresponds to an individual user review and is associated with additional metadata, providing a comprehensive view of the users' sentiments and the context in which they were expressed.

The dataset is structured in a tabular format and is comprising of **Five distinct features**, each serving a unique purpose in the context of the data:

1. **Time_submitted:** This feature is represented in a numerical format, containing records of the precise time at which each review was submitted.
2. **Review:** This column is a textual field which encapsulates the actual content of the user's review.
3. **Rating:** This numerical column contains records of the rating given by the user, ranging from a minimum of 1 to a maximum of 5.
4. **Total_thumbsup:** This is another numerical column. This feature indicates the count of individuals who found the review helpful.
5. **Reply:** This textual column contains any replies from Spotify's development team to the user's review.

The combination of text and numerical data types provides a rich source of information for conducting a comprehensive sentiment analysis.
The features that will be used in this project for the analysis of positive and negative reviews are **Review** and **Rating**.

Ethics:
From an ethical and security standpoint, the dataset maintains a high level of anonymity as it does not contain any personally identifiable information. The reviews and ratings are anonymized, ensuring the privacy of the users who submitted the reviews.

License:

The dataset is licensed under the Attribution-NonCommercial 4.0 International (CC BY-NC 4.0) license. This license grants the freedom to share and adapt the dataset while requiring appropriate credit and disallowing commercial use. The licensing terms align with ethical considerations, fostering responsible usage and respecting the rights of the dataset curator.

Usage and Contributions:
The dataset has a usability score of **10.0** and has been downloaded over **4,476 times**. It has also been utilized in **18 different notebooks**. Its popularity within the data science community emphasizes its value and suitability for sentiment analysis tasks. Notably, the dataset has been frequently employed to explore the factors contributing to both 1-star and 5-star ratings. However, it is important to note that the dataset is not anticipated to receive further updates.

In summary, the "Spotify App Reviews 2022" dataset is an exceptional resource for sentiment analysis, offering a substantial collection of user reviews with diverse sentiments and associated metadata. Its structured format, anonymity, and licensing align with the ethical considerations. Leveraging this dataset will provide unique insights into user sentiment patterns, contributing to an in-depth understanding of user experiences and assisting in the enhancement of the Spotify application.

Evaluation Methodology

To evaluate the performance of the sentiment analysis models developed in this project for the Spotify application reviews, two evaluation techniques, namely accuracy and F1 score, will be primarily utilized due to their suitability and relevance to the problem at hand.

Accuracy: Accuracy is a fundamental evaluation metric that measures the overall correctness of the model's predictions. In the context of sentiment analysis, accuracy assesses the model's ability to correctly classify reviews as positive or negative. As the primary objective of the project is to determine the sentiment of Spotify reviews accurately, accuracy provides a comprehensive measure of the model's performance in achieving this objective. It is particularly useful for obtaining an overall assessment of the model's effectiveness and gauging their ability to make correct predictions.(Malato G, 2021)

F1 Score: The F1 score takes into account both precision, which measures the ability to correctly identify positive and negative reviews, and recall, which measures the ability to capture all positive and negative reviews. By considering both precision and recall, the F1 score provides a balanced assessment of the model's performance and is particularly relevant when the goal is to achieve a trade-off between precision and recall.(Korstanje J, 2021)

Other evaluation metrics such as precision, recall, and the confusion matrix, will be used as supplmentary reference metrics as they are valuable references for analyzing the model's performance. Precision represents the proportion of correctly predicted positive or negative reviews among all predicted positive or negative reviews, while recall measures the proportion of correctly predicted positive or negative reviews among all actual positive or negative reviews. The confusion matrix provides a detailed breakdown of the model's predictions, including true positives, true negatives, false positives, and false negatives. These metrics offer a more granular analysis of the model's performance and enables the identification of specific areas where the models may be making errors.(Malato G, 2021) (Narkhede S, 2018)

By primarily utilizing accuracy and F1 score, the project ensures a comprehensive evaluation of the sentiment analysis model's performance in classifying Spotify reviews. In addition to these metrics, further analysis and interpretation of the evaluation results will be conducted to draw meaningful conclusions. These conclusions will consider the model's performance in light of their objectives and potential contributions to the problem area. By critically evaluating the model's effectiveness and analyzing their limitations, recommendations for future improvements can be provided.

References:

1. Malato G, "Precision, recall, accuracy. How to choose?", 2021.[Online].

Available: https://www.yourdatateacher.com/2021/06/07/precision-recall-accuracy-how-to-choose/#:~:text=Precision%20is%20a%20measure%20of,the%20model%20has%20been%20true.

2. Korstanje J, "The F1 score", 2021.[Online].

Available: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6

3. Narkhede S, "Understanding Confusion Matrix", 2023.[Online].

Available:https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

Implementation

Installing and Importing the required Libraries

The following libraries were installed using the command pip install so that they could be imported and used.

- Pandas
- Matplotlit
- Wordcloud
- Nltk
- Scikit-learn
- Transformers
- Torch
- Tqdm
- Prettytable

In [1]:
```
# Numpy
import numpy as np

# Random module for setting random seed
import random

# Pandas
import pandas as pd

# Matplotlib
import matplotlib.pyplot as plt

# NLTK
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
nltk.download('punkt', quiet=True)  # Downloading required resources for NLTK
nltk.download('stopwords', quiet=True)
nltk.download('wordnet', quiet=True)

# RE
import re
```

```
# Collections module from Python
from collections import Counter

# Wordcloud
from wordcloud import WordCloud

# Scikit-Learn
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, confusion_matrix, make_scorer

# Transformers & Torch
from transformers import AutoModelForSequenceClassification, AutoTokenizer
import torch
from tqdm import tqdm

# Prettytable
from prettytable import PrettyTable

# Setting the random seed for reproducibility
random_seed = 33
random.seed(random_seed)
np.random.seed(random_seed)
torch.manual_seed(random_seed)
torch.cuda.manual_seed(random_seed)
```

Importing the Dataset and Data Exploration

In [2]:
```
# The Spotify App Reviews dataset is taken from Kaggle, "Spotify App Reviews", 2022.[Online]
# Available: https://www.kaggle.com/datasets/mfaaris/spotify-app-reviews-2022?resource=download
# License: Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)
# License Link: https://creativecommons.org/licenses/by-nc/4.0/
```

```
spotify_dataset = pd.read_csv("reviews.csv")
spotify_dataset.head()
```

Out[2]:

	Time_submitted	Review	Rating	Total_thumbsup	Reply
0	2022-07-09 15:00:00	Great music service, the audio is high quality...	5	2	NaN
1	2022-07-09 14:21:22	Please ignore previous negative rating. This a...	5	1	NaN
2	2022-07-09 13:27:32	This pop-up "Get the best Spotify experience o...	4	0	NaN
3	2022-07-09 13:26:45	Really buggy and terrible to use as of recently	1	1	NaN
4	2022-07-09 13:20:49	Dear Spotify why do I get songs that I didn't ...	1	1	NaN

In [3]:
```
# Checking the data types of all the columns
spotify_dataset.info()
print(spotify_dataset.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61594 entries, 0 to 61593
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Time_submitted  61594 non-null  object
 1   Review          61594 non-null  object
 2   Rating          61594 non-null  int64 
 3   Total_thumbsup  61594 non-null  int64 
 4   Reply           216 non-null    object
dtypes: int64(2), object(3)
memory usage: 2.3+ MB
Time_submitted        0
Review                0
Rating                0
Total_thumbsup        0
Reply             61378
dtype: int64
```

This dataset contains **5 columns(features)** with **61594 rows**. The **Reply** column contains **61378** rows with null values and none of the other columns contain any null values

In [4]:
```
# Checking the number of unique records from each feature
spotify_dataset.nunique()
```

Out[4]:
```
Time_submitted    61300
Review            61356
Rating                5
Total_thumbsup      532
Reply               180
dtype: int64
```

It is plausible for all columns, excluding the Review column, to contain duplicate values within the dataset. Such duplication can occur due to simultaneous review submissions, multiple individuals assigning the same rating to the application, or identical numbers of upvotes received on a review. Furthermore, automated responses generated by the application developer may contribute to duplications in certain columns.

However, it is noteworthy that the Review column itself also exhibits duplicate values. While it is conceivable for written reviews to occasionally coincide between individuals, such instances are relatively uncommon. As a result, it is prudent to examine the duplicate entries within the Review column for potential outliers. Any redundant or unnecessary duplicates can be eliminated during the data processing stage.

Data Cleaning and Preprocessing

In [5]:
```
# Removing unnecessary columns
spotify_dataset2 = spotify_dataset.drop(['Reply'], axis=1)
spotify_dataset2.head()
```

Out[5]:

	Time_submitted	Review	Rating	Total_thumbsup
0	2022-07-09 15:00:00	Great music service, the audio is high quality...	5	2
1	2022-07-09 14:21:22	Please ignore previous negative rating. This a...	5	1
2	2022-07-09 13:27:32	This pop-up "Get the best Spotify experience o...	4	0
3	2022-07-09 13:26:45	Really buggy and terrible to use as of recently	1	1
4	2022-07-09 13:20:49	Dear Spotify why do I get songs that I didn't ...	1	1

The **Reply** column is removed as it has a lot of missing data and is also not useful in this case for predicting if the review is positive or negative. As discussed above in the **Dataset** section, the features that will be in focus for this project are the **Review** and **Rating** columns.

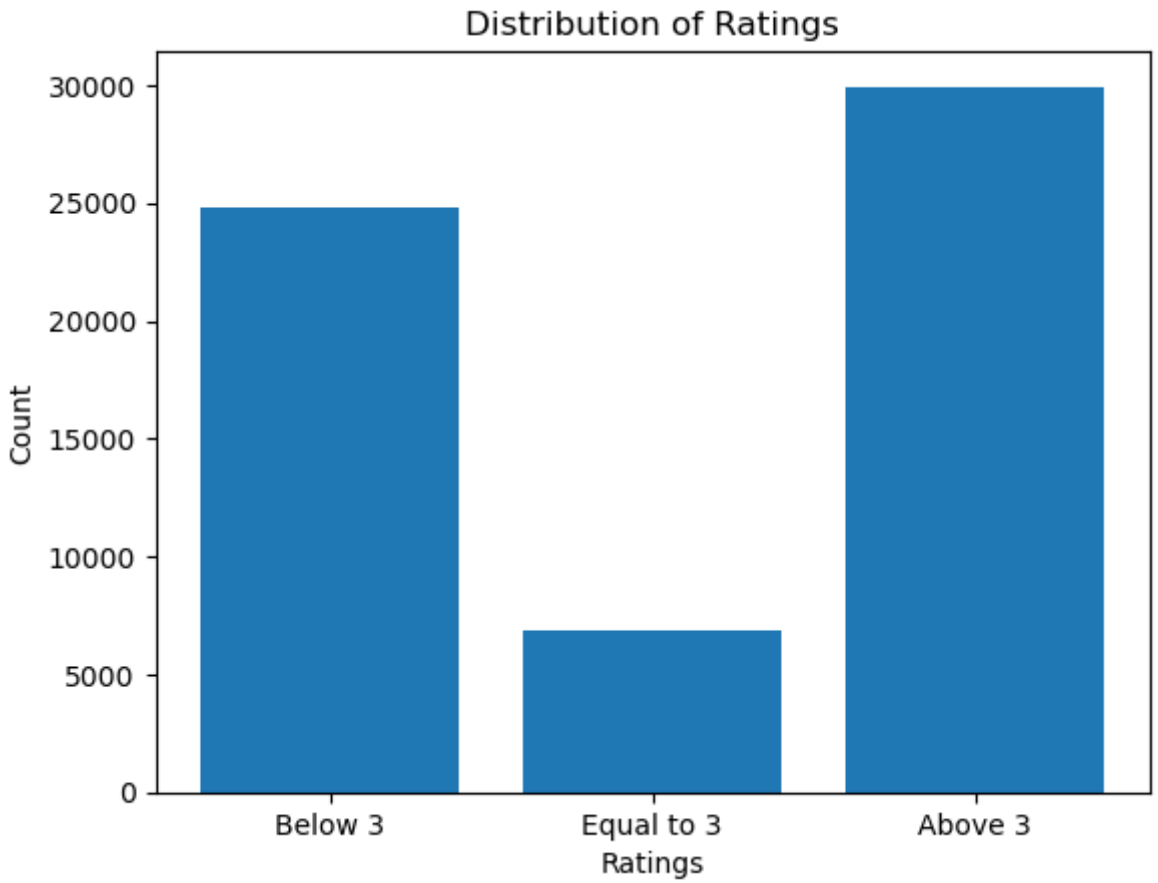
Checking the distribution of data in the Ratings column

In [6]:
```
# Checking the distribution of the ratings
below_3 = spotify_dataset2[spotify_dataset2['Rating'] < 3]
equal_to_3 = spotify_dataset2[spotify_dataset2['Rating'] == 3]
above_3 = spotify_dataset2[spotify_dataset2['Rating'] > 3]
```

In [7]:
```
# Counting the number of records in each part
below_3_count = len(below_3)
equal_to_3_count = len(equal_to_3)
above_3_count = len(above_3)

print("Ratings Below 3:", below_3_count)
print("Ratings Equal to 3:", equal_to_3_count)
print("Ratings Above 3:", above_3_count)

# Creating a bar graph to visualize the distribution
labels = ['Below 3', 'Equal to 3', 'Above 3']
counts = [below_3_count, equal_to_3_count, above_3_count]
print("-" * 70 )

plt.bar(labels, counts)
```

```
plt.xlabel('Ratings')
plt.ylabel('Count')
plt.title('Distribution of Ratings')
plt.show()
```

```
Ratings Below 3: 24771
Ratings Equal to 3: 6886
Ratings Above 3: 29937
--------------------------------------------------------------------------------
```



Filtering out rows where rating is equal to 3

In [8]:

```
# Filtering out rows with a rating of 3
spotify_dataset3 = spotify_dataset2[spotify_dataset2['Rating'] != equal_to_3['Rating'].values[0]]
spotify_dataset3.head()

# Verifying that there are no more ratings in the dataset that are equal to 3.
num_ratings_equal_to_target = len(spotify_dataset3[spotify_dataset3['Rating'] == equal_to_3['Rating'].values[0]])
print(f"The number of ratings equal to {equal_to_3['Rating'].values[0]} "
      f"in the clean dataset: {num_ratings_equal_to_target}")
print("-" * 50)
print(spotify_dataset3.head())
print("-" * 50)
print(spotify_dataset3.info())
```

```
The number of ratings equal to 3 in the clean dataset: 0
--------------------------------------------------
        Time_submitted                                             Review  \
0  2022-07-09 15:00:00  Great music service, the audio is high quality...
1  2022-07-09 14:21:22  Please ignore previous negative rating. This a...
2  2022-07-09 13:27:32  This pop-up "Get the best Spotify experience o...
3  2022-07-09 13:26:45    Really buggy and terrible to use as of recently
4  2022-07-09 13:20:49  Dear Spotify why do I get songs that I didn't ...

   Rating  Total_thumbsup
0       5               2
1       5               1
2       4               0
3       1               1
4       1               1
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
Int64Index: 54708 entries, 0 to 61593
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Time_submitted  54708 non-null  object
 1   Review          54708 non-null  object
 2   Rating          54708 non-null  int64
 3   Total_thumbsup  54708 non-null  int64
dtypes: int64(2), object(2)
memory usage: 2.1+ MB
None
```

The dataset's Reviews column has been segmented into three distinct categories for the purpose of the analysis. Ratings below 3 have been designated as negative, ratings equal to 3 as neutral, and ratings above 3 as positive. As explained above in the **Objective**, the primary focus lies on discerning positive and negative reviews, as they provide valuable insights into the strengths and weaknesses of the application. Positive reviews highlight the aspects of the application that are functioning well, while negative reviews shed light on areas that users find problematic or unsatisfactory. Hence, the dataset is being classified into positive and negative reviews, with the neutral reviews (ratings equal to 3) being omitted. Although it is possible to consider ratings equal to 3 as negative, the decision to exclude them stems from two factors:

- Some reviews corresponding to a rating of 3 are truly neutral in nature.

- The quantity of reviews with a rating of 3 is relatively low.

Hence, for this project only the positive and negative reviews will be considered. The use of netural reviews can be explored in a future study.

Checking for duplicates before Text Preprocessing

In [9]:
```
# Checking for duplicate values in the 'Review' column
duplicate_reviews = spotify_dataset3[spotify_dataset3.duplicated(['Review'], keep=False)]

# Counting the number of duplicate reviews
duplicate_counts = duplicate_reviews['Review'].value_counts()
print(duplicate_counts.head(15)) # Only printing the first 15 duplicate values to get some inference from the data
```

```
Too many ads                    32
Too much ads                    12
Amazing music app                9
Great music selection            8
Very good music app              8
Way too many ads                 7
Excellent music app              6
Good sound quality               6
Great selection of music.        5
Too many ads.                    5
Best app for songs               4
I love Spotify!                  4
Not working properly             4
Best app for listening music     4
Too many adds                    4
Name: Review, dtype: int64
```

It can be observed that certain reviews, such as "Too many ads" and "Too many ads.", may appear to be identical except for differences in punctuation. Consequently, in order to address such duplications, text processing techniques will be applied. Moreover, it is noteworthy that expressions like "Too many ads" and "Too much ads" convey the same meaning and imply a similar sentiment. However, stemming or lemmatization alone would not suffice to reduce the words "many" and "much" to a common root since they are distinct lexical units. In order to consider these phrases as equivalent, a manual compilation of a comprehensive list would be necessary, followed by replacing all instances of such phrases within the text. However, this manual approach is impractical due to the large

number of phrases involved and carries the risk of oversimplifying the text while potentially compromising nuanced meanings. Hence, for the purposes of this project, these phrases will be treated as distinct entities, preserving their individuality and potential variations in sentiment.

Text Preprocessing

In [10]:
```
# Stopwords referenced from: GeeksforGeeks, "Removing stop words with NLTK in Python", 2023.[Online]
# Available: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

# Lemmatization referenced from: R Yash, "Python | Lemmatization with NLTK", 2023.[Online]
# Available: https://www.geeksforgeeks.org/python-lemmatization-with-nltk/

# The techniques have been referenced for guidance from the links above,
# but the code is original and is implemented independently.

# Function to preprocess the text
def preprocess_text(text):
    # Converting the text to lowercase
    text = text.lower()

    # Removing the punctuation
    text = re.sub(r'[^a-z0-9]', ' ', text)

    # Tokenizing the text
    words = word_tokenize(text)

    # Removing the stopwords
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words]

    # Lemmatizing the words
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]

    # Joining the words back into a string
    text = ' '.join(words)

    # Removing leading and trailing spaces
```

```
    text = text.strip()

    # Replacing multiple spaces with a single space
    text = re.sub(' +', ' ', text)

    return text

# Creating a copy of the DataFrame
spotify_dataset3_copy = spotify_dataset3.copy()

# Applying the preprocessing to the 'Review' column of the copy
spotify_dataset3_copy['Review'] = spotify_dataset3_copy['Review'].apply(preprocess_text)

# Displaying the first few rows of the modified DataFrame
spotify_dataset3_copy.head()
```

Out[10]:

	Time_submitted	Review	Rating	Total_thumbsup
0	2022-07-09 15:00:00	great music service audio high quality app eas...	5	2
1	2022-07-09 14:21:22	please ignore previous negative rating app sup...	5	1
2	2022-07-09 13:27:32	pop get best spotify experience android 12 ann...	4	0
3	2022-07-09 13:26:45	really buggy terrible use recently	1	1
4	2022-07-09 13:20:49	dear spotify get song put playlist shuffle play	1	1

The text preprocessing techniques used above, collectively enhance the accuracy of sentiment analysis by reducing noise, handling variations in word forms, and allowing for better identification of sentiment-bearing words and phrases. The resulting preprocessed text is more amenable to classification algorithms and provides a more accurate representation of user sentiments towards the Spotify application.

Checking for duplicates after Text Preprocessing

In [11]:

```
# Checking for duplicate values in the preprocessed 'Review' column
duplicate_reviews2 = spotify_dataset3_copy[spotify_dataset3_copy.duplicated(['Review'],
                                                                            keep=False)]
```

```
# Counting the number of duplicate reviews
duplicate_counts2 = duplicate_reviews2['Review'].value_counts()
# Only printing the first 30 duplicate values to get some inference from the data
print(duplicate_counts2.head(10))
print("-" * 50)

# Counting the total number of duplicate reviews
total_duplicates = len(spotify_dataset3_copy) - spotify_dataset3_copy['Review'].nunique()
print("Total number of duplicate reviews:", total_duplicates)
```

```
many ad                     54
best music app ever         32
best music app              30
much ad                     27
love spotify                22
best music streaming app    16
good music app              13
best app music              13
great music app             13
spotify best                13
Name: Review, dtype: int64
--------------------------------------------------
Total number of duplicate reviews: 806
```

In [12]:
```
# Dropping the duplicate reviews
spotify_dataset4 = spotify_dataset3_copy.drop_duplicates(subset=['Review'],
                                                         keep='first')

# Printing the number of records before and after dropping duplicates
print("Number of records before dropping duplicates:",
      len(spotify_dataset3_copy))

print("Number of records after dropping duplicates:",
      len(spotify_dataset4))
```

```
Number of records before dropping duplicates: 54708
Number of records after dropping duplicates: 53902
```

Given that the dataset contained a relatively small number of duplicate reviews (806) after text preprocessing, a decision was made to remove these duplicate values, considering the considerable size of the dataset and the need for data integrity.

In [13]:
```
# Creating a new DataFrame
spotify_dataset5 = spotify_dataset4.copy()

# Replacing ratings above 3 with "positive" and ratings below 3 with "negative"
spotify_dataset5['Rating'] = spotify_dataset5['Rating'].apply(lambda x: 'positive' if x > 3 else 'negative')
spotify_dataset5.head()
```

Out[13]:

	Time_submitted	Review	Rating	Total_thumbsup
0	2022-07-09 15:00:00	great music service audio high quality app eas...	positive	2
1	2022-07-09 14:21:22	please ignore previous negative rating app sup...	positive	1
2	2022-07-09 13:27:32	pop get best spotify experience android 12 ann...	positive	0
3	2022-07-09 13:26:45	really buggy terrible use recently	negative	1
4	2022-07-09 13:20:49	dear spotify get song put playlist shuffle play	negative	1

In order to achieve the desired binary classification, the ratings were transformed into either positive or negative categories, aligning with the intended sentiment analysis objectives. This conversion allows for a simplified and more straightforward classification task, facilitating the analysis of user sentiments towards the Spotify application.

In [14]:
```
# Converting the Time_submitted coulmn into datetime format
spotify_dataset5['Time_submitted'] = pd.to_datetime(spotify_dataset5['Time_submitted'])

# Getting the maximum date in 'Time_submitted'
max_date = spotify_dataset5['Time_submitted'].max()

# Normalizing the 'Total_thumbsup' column by dividing by the maximum value
spotify_dataset5['Total_thumbsup_norm'] = \
spotify_dataset5['Total_thumbsup'] / spotify_dataset5['Total_thumbsup'].max()

# Normalizing the'Time_submitted' column by converting to the number of days since the maximum date,
# and then dividing by the maximum number of days
spotify_dataset5['Time_submitted_norm'] = (max_date - spotify_dataset5['Time_submitted']).dt.days / \
(max_date - spotify_dataset5['Time_submitted']).dt.days.max()

# Creating a new column called Score which is the combination of the Time_submitted and "Total_thumbsup" columns.
```

```
spotify_dataset5['Score'] =  (2 * spotify_dataset5['Total_thumbsup_norm']) +\
spotify_dataset5['Time_submitted_norm']
```

In [15]:
```
spotify_dataset5.head()
```

Out[15]:

	Time_submitted	Review	Rating	Total_thumbsup	Total_thumbsup_norm	Time_submitted_norm	Score
0	2022-07-09 15:00:00	great music service audio high quality app eas...	positive	2	0.000244	0.0	0.000488
1	2022-07-09 14:21:22	please ignore previous negative rating app sup...	positive	1	0.000122	0.0	0.000244
2	2022-07-09 13:27:32	pop get best spotify experience android 12 ann...	positive	0	0.000000	0.0	0.000000
3	2022-07-09 13:26:45	really buggy terrible use recently	negative	1	0.000122	0.0	0.000244
4	2022-07-09 13:20:49	dear spotify get song put playlist shuffle play	negative	1	0.000122	0.0	0.000244

To address the need for prioritizing reviews based on their relevance and significance, a sorting approach was considered to provide valuable insights into the strengths and weaknesses of the Spotify application. Initially, the intention was to sort the dataframe in **descending order of the Total_thumbsup column**, which represents the count of individuals who found the reviews helpful. This would have allowed for the identification of reviews that received high appreciation from users, providing an indication of the application's positive aspects or areas requiring improvement.

Additionally, it was crucial to consider the temporal relevance of reviews and account for the potential impact of outdated information. To achieve this, a **descending order based on the Time_submitted column was desired to access the latest reviews**. This approach aimed to mitigate potential discrepancies arising from outdated features or resolved issues in subsequent versions.

However, it was observed that sorting the dataframe based on both columns sequentially resulted in the overwritten ordering, limiting the desired insights. To resolve this challenge, a new column called **Score** was introduced. The Score column was created by **combining the normalized values of the Time_submitted and Total_thumbsup columns**. The Time_submitted column was first

converted to the datetime format for performing date operations. The latest date of review submission was determined and used to normalize the Time_submitted values. Similarly, the Total_thumbsup column was normalized by dividing each value by the maximum value. While combining these columns, a weight of **twice the magnitude was assigned to the Total_thumbsup column** to prioritize its contribution to the overall Score. This approach facilitated an effective combination of temporal relevance and user appreciation, enabling a more comprehensive evaluation of the reviews' value.

Cleaning up the dataset before getting the final dataset that will be used for modelling

In [16]:
```
# Creating the clean_dataset that will only contain the Review, Rating, and Score columns
clean_dataset = spotify_dataset5[['Review',
                                  'Rating',
                                  'Score']]

# Ordering the clean_dataset in descending order of the Score column
clean_dataset = clean_dataset.sort_values(by='Score',
                                          ascending=False)
clean_dataset.head()
```

Out[16]:

	Review	Rating	Score
36968	app good got explore many new song however too...	positive	2.460317
37523	forever glitchy app play button current song p...	negative	1.930970
35203	fantastic app always great experience app thor...	positive	1.615568
31673	2 year use far favorite app play music easy us...	positive	1.587239
58520	best music app find problem ad frequent long a...	positive	1.483710

In [17]:
```
# Creating the final_dataset with only the Review and Rating columns, and only the first 30000 rows
final_dataset = clean_dataset[['Review',
                               'Rating']].iloc[:30000]
final_dataset.head()
```

Out[17]:

	Review	Rating
36968	app good got explore many new song however too...	positive
37523	forever glitchy app play button current song p...	negative
35203	fantastic app always great experience app thor...	positive
31673	2 year use far favorite app play music easy us...	positive
58520	best music app find problem ad frequent long a...	positive

To streamline the dataset and focus on the essential features for subsequent analysis, a clean dataset was created by removing redundant columns such as Total_thumbsup and Total_thumbsup_norm. This step aimed to enhance the dataset's efficiency and clarity by retaining only the Review and Rating features. These features were deemed necessary for training the machine learning model to classify reviews as either positive or negative.

Furthermore, to manage the dataset effectively and expedite subsequent processes, including training and evaluation, a subset of the clean dataset consisting of the first 30,000 records was selected. This decision was driven by various factors, including the need for faster processing, facilitating easier experimentation, reducing memory usage, and expediting the debugging process. By working with a smaller subset, computational resources could be optimized, and potential issues or modifications could be addressed promptly.

The chosen subset ensured a balance between maintaining a representative sample size and mitigating computational complexities, thereby enabling efficient exploration and analysis of the dataset.

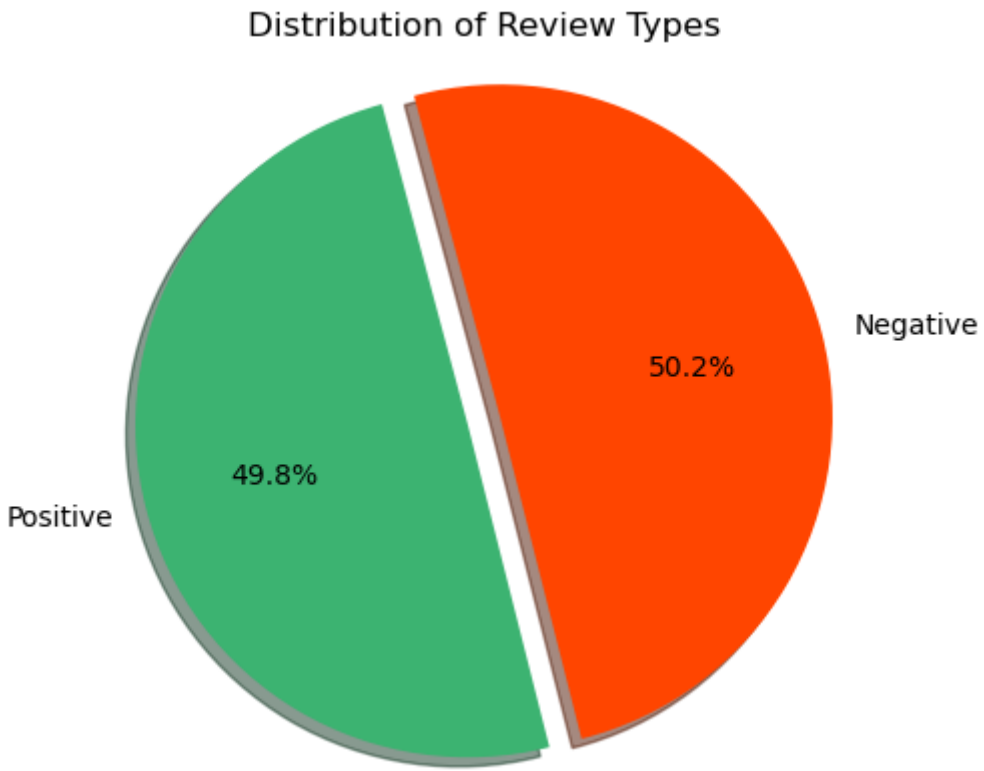
Exploratory Data Analysis

Checking the distribution of positive and negative reviews in the final dataset

In [18]:
```
# Counting the number of positive and negative reviews
positive_reviews = final_dataset[final_dataset['Rating'] == 'positive']
negative_reviews = final_dataset[final_dataset['Rating'] == 'negative']

# Counting the number of records in each part
positive_reviews_count = len(positive_reviews)
```

```
negative_reviews_count = len(negative_reviews)

print("Positive Reviews:",
      positive_reviews_count)
print("Negative Reviews:",
      negative_reviews_count)

# Initializing the variables
labels = ['Positive',
          'Negative']

counts = [positive_reviews_count,
          negative_reviews_count]

colors = ['mediumseagreen',
          'orangered']

explosion = [0.1,
             0]

plt.pie(counts,
        labels=labels,
        colors=colors,
        autopct='%1.1f%%',
        startangle = 105,
        shadow = True,
        explode = explosion)
plt.title('Distribution of Review Types')
plt.axis('equal')
plt.show()
```

```
Positive Reviews: 14933
Negative Reviews: 15067
```

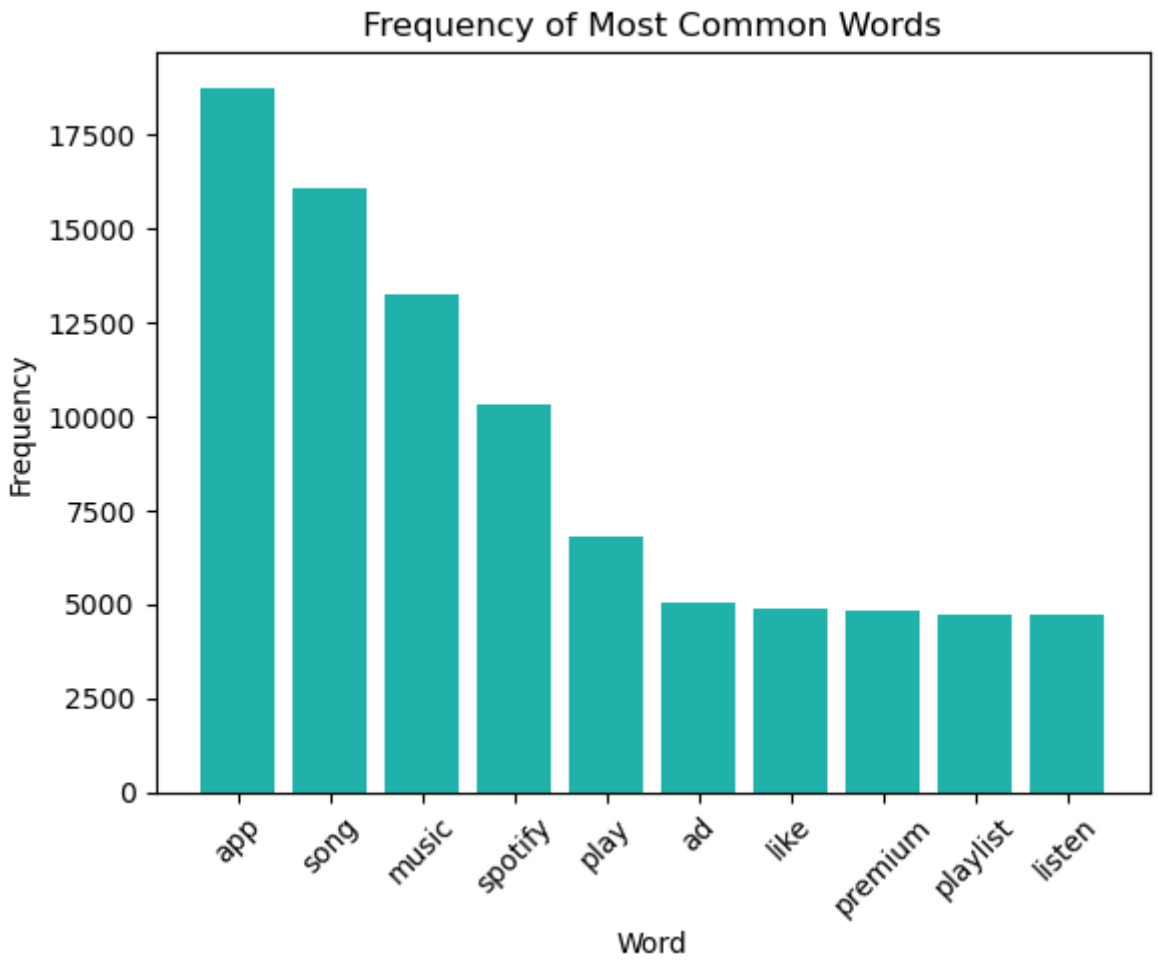


From the Pie plot above, it can be observed that the distribution of positive and negative reviews in the dataset is almost equal. There are **14,933 positive reviews** and **15,067 negative reviews**. This balance in the dataset is good for training a machine learning model as it prevents the model from being biased towards a particular class.

Getting the most common words

In [19]:
```
# Referenced from: nikhilaggarwal3, "Python Collections Module", 2023.[Online]
# Available: https://www.geeksforgeeks.org/python-collections-module/

# The techniques have been referenced for guidance from the link above,
# but the code is original and is implemented independently.

# Tokenizing the reviews
```

```
tokens = final_dataset['Review'].apply(nltk.word_tokenize)

# Flattening the list of tokens and counting the frequency of each word
word_frequency = Counter([word for sublist in tokens for word in sublist])

# Getting the 10 most common words
most_common_words = word_frequency.most_common(10)

# Creating a bar graph to visualize the frequency
labels, counts = zip(*most_common_words)

plt.bar(labels, counts, color='lightseagreen')
plt.xlabel('Word')
plt.ylabel('Frequency')
plt.title('Frequency of Most Common Words')
plt.xticks(rotation=45)
plt.show()
```



In [20]:
```
# Concatenating all reviews into one string
all_reviews = ' '.join(final_dataset['Review'])

# Generating a word cloud image
wordcloud = WordCloud(background_color='white').generate(all_reviews)

# Displaying the generated image
plt.imshow(wordcloud,
           interpolation='bilinear')
```

```
plt.axis('off')
plt.show()
```

From the given dataset, after removing the stopwords such as 'the', 'is', 'at', 'which', and 'on' during text preprocessing, the data given in the bar graph and the wordcloud seems to be the expected output.

Data Manipulation

In [21]:
```
# Splitting the data into training and testing sets
X_train,X_test, y_train, y_test = train_test_split(final_dataset['Review'],
                                                   final_dataset['Rating'],
                                                   test_size=0.2,
                                                   random_state=33)
```

Using Tfidf Vectorizer

In [22]:
```
# Referenced from: scikit-learn, "sklearn.feature_extraction.text.TfidfVectorizer", 2023.[Online]
# Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

# The documentation has been referenced for guidance from the link above,
# but the code is original and is implemented independently.
```

```
# Initializing a TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fitting and transforming the vectorizer on the training data
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

# Transforming the testing data with the vectorizer
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

The choice of the TF-IDF vectorizer was motivated by its ability to effectively represent the textual data in this sentiment analysis project. TF-IDF (Term Frequency-Inverse Document Frequency) assigns weights to each term based on its frequency in a document and its rarity across the entire dataset. This approach is advantageous because it emphasizes terms that are highly relevant to a particular document while downplaying common terms. In comparison to other vectorizers such as the Bag-of-Words model, TF-IDF considers the importance of terms in the context of the entire corpus, resulting in more meaningful and discriminative feature representations.

References:

1. Dutta M, "Bag-of-words vs TFIDF vectorization –A Hands-on Tutorial", 2021.[Online].

Available: https://www.analyticsvidhya.com/blog/2021/07/bag-of-words-vs-tfidf-vectorization-a-hands-on-tutorial/

2. Kaplan D, "Machine Learning 101: CountVectorizer Vs TFIDFVectorizer", 2022.[Online].

Available: https://enjoymachinelearning.com/blog/countvectorizer-vs-tfidfvectorizer/#:~:text=CountVectorizer%20simply%20counts%20the%20number,is%20to%20the%20whole%20corpus.

Label Encoding the Rating column

In [23]:
```
# Referenced from: scikit-learn, "sklearn.preprocessing.LabelEncoder", 2023.[Online]
# Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

# The documentation has been referenced for guidance from the link above,
# but the code is original and is implemented independently.

#Encoding the Rating Column into 0s and 1s for the model to process
```

```
# Initializing the LabelEncoder
le = LabelEncoder()

# Fitting and transforming the Rating column
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)
```

As the Rating column contained text values for the rating, it was Label encoded to transform the text values into numerical values for the ML classification models to be able to process those values.

Building the Baseline model

Using a pre-trained RoBERTa transformer from Huggingface as the baseline

The pretrained model is taken from Huggingface:
Siebert C, "siebert/sentiment-roberta-large-english", 2023.[Online].
Available: https://huggingface.co/siebert/sentiment-roberta-large-english

Citation:
Hartmann J and Heitmann M and Siebert C and Schamp C, "More than a Feeling: Accuracy and Application of Sentiment Analysis", 2023. [Online].
Available: https://www.sciencedirect.com/science/article/pii/S0167811622000477

In [24]:
```
# Creating a new function for the pretrained model evaluation
def evaluate_model_bert(y_true, y_pred):

    evaluation_metrics = {}

    # Calculate evaluation metrics
    evaluation_metrics['Accuracy'] = accuracy_score(y_true,
                                                    y_pred)

    evaluation_metrics['Precision'] = precision_score(y_true,
                                                      y_pred,
                                                      pos_label='positive')
```

```
    evaluation_metrics['Recall'] = recall_score(y_true,
                                                y_pred,
                                                pos_label='positive')

    evaluation_metrics['F1 Score'] = f1_score(y_true,
                                              y_pred,
                                              pos_label='positive')

    evaluation_metrics['Confusion Matrix'] = confusion_matrix(y_true, y_pred)

    return evaluation_metrics
```

In [25]:

```
# Referenced from: ptrblck, "Training loop getting slower and slower", 2021.[Online]
# Available: https://discuss.pytorch.org/t/training-loop-getting-slower-and-slower/110982

# Referenced from: Deckers M., "Using CUDA with pytorch?", 2018.[Online]
# Available: https://stackoverflow.com/questions/50954479/using-cuda-with-pytorch

# Referenced from: tqdm, "tqdm", 2023.[Online]
# Available: https://github.com/tqdm/tqdm

# Referenced from: Mulla R, "Python Sentiment Analysis Project with NLTK and 🤗 Transformers. Classify Amazon Revie
# Available: https://www.youtube.com/watch?v=QpzMWQvxXWk

# The techniques, code and steps have been referenced for guidance and adapted from the links above.

max_length = 512
dim = 1

# Loading the pre-trained model and tokenizer
model_name = "siebert/sentiment-roberta-large-english"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

# Setting up batch processing
batch_size = 64

# Using the Review column from the final_dataset
reviews = final_dataset['Review'].tolist()
```

```python
# Function to process a batch of reviews
def process_batch(batch):
    tokens = tokenizer(batch,
                       truncation = True,
                       padding = True,
                       max_length = max_length,
                       return_tensors = 'pt')
    with torch.no_grad():
        outputs = model(**tokens)
    predicted_labels = torch.argmax(outputs.logits,
                                    dim = dim).tolist()
    sentiments = ['positive' if label == 1 else 'negative' for label in predicted_labels]
    return sentiments

# Splitting the reviews into batches
batches = [reviews[i:i + batch_size] for i in range(0,
                                                    len(reviews),
                                                    batch_size)]

# Running the pytorch library on the CPU
device = torch.device("cpu")
model.to(device)

# Processing the batches
sentiments = []
with torch.no_grad():
    for batch in tqdm(batches):
        tokens = tokenizer(batch,
                           truncation = True,
                           padding=True,
                           max_length = max_length,
                           return_tensors='pt').to(device)

        outputs = model(**tokens)
        predicted_labels = torch.argmax(outputs.logits, dim=1).tolist()
        batch_sentiments = ['positive' if label == 1 else 'negative' for label in predicted_labels]
        sentiments.extend(batch_sentiments)

# Calling the evaluate_model function to obtain evaluation metrics
```

```
bert_eval = evaluate_model_bert(final_dataset['Rating'],
                                sentiments)

# Printing the evaluation metrics
for metric, value in bert_eval.items():
    print(f'{metric}: {value}')
```

```
100%|██████████| 469/469 [49:42<00:00,  6.36s/it]
Accuracy: 0.8393
Precision: 0.8415754627753006
Recall: 0.8341927275162392
F1 Score: 0.8378678325205986
Confusion Matrix: [[12722  2345]
 [ 2476 12457]]
```

The output of the Roberta pretrained sentiment analysis model is given below:(The code takes approximately 45 mins to run)

Accuracy: 0.8393

Precision: 0.8415754627753006

Recall: 0.8341927275162392

F1 Score: 0.8378678325205986

Confusion Matrix:

[[12722 2345]

[2476 12457]]

The pre-trained RoBERTa model has been selected as the baseline model for this task due to its advanced language understanding capabilities and its proficiency in capturing intricate patterns and nuances in textual data. As a state-of-the-art language model, the siebert RoBERTa model from huggingface has undergone extensive training on a large corpus of text, equipping it with the ability to acquire representations that demonstrate effective generalization across various natural language processing tasks.

By employing the pre-trained RoBERTa model as the baseline, a convincing foundation is established for evaluating the performance of more complex models within the sentiment analysis framework. This choice of baseline allows for a meaningful comparison and assessment of the improvements achieved by the subsequent complex models.

The evaluation results, particularly the accuracy and F1 score, affirm the proficiency of the RoBERTa model in extracting sentiment information from the Spotify reviews. Through a comparative analysis against the baseline, the performance enhancements provided by

the alternative complex models can be measured and their additional value in terms of sentiment analysis on the Spotify application can be ascertained.

In summary, the pre-trained RoBERTa model serves as a robust and reliable baseline, providing a reasonable starting point for the development and evaluation of more sophisticated models. These subsequent models aim to surpass the performance achieved by the RoBERTa model and deliver even greater precision and insight in sentiment analysis for Spotify reviews.

Building the complex classifier models

In [26]:
```
# Creating a function for the model evaluation
def evaluate_model(y_true, y_pred):

    evaluation_metrics = {}

    # Calculate evaluation metrics
    evaluation_metrics['Accuracy'] = accuracy_score(y_true,
                                                    y_pred)

    evaluation_metrics['Precision'] = precision_score(y_true,
                                                      y_pred)

    evaluation_metrics['Recall'] = recall_score(y_true,
                                                y_pred)

    evaluation_metrics['F1 Score'] = f1_score(y_true,
                                              y_pred)

    evaluation_metrics['Confusion Matrix'] = confusion_matrix(y_true,
                                                              y_pred)

    return evaluation_metrics
```

Multinomial Naive Bayes

In [27]:
```
# Referenced from: scikit-learn, "sklearn.naive_bayes.MultinomialNB", 2023.[Online]
# Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
```

```
# The documentation has been referenced for guidance from the link above,
# but the code is original and is implemented independently.

# Initializing a MultinomialNB
mnb_tfidf = MultinomialNB()

# Fitting the classifier
mnb_tfidf.fit(X_train_tfidf, y_train)

# Predicting the test set results
y_pred = mnb_tfidf.predict(X_test_tfidf)

# Evaluating the model
mnb_tfidf_eval = evaluate_model(y_test, y_pred)

# Printing the evaluation metrics
for metric, value in mnb_tfidf_eval.items():
    print(f'{metric}: {value}')
```

```
Accuracy: 0.8676666666666667
Precision: 0.9131059245960502
Recall: 0.8216478190630049
F1 Score: 0.8649659863945577
Confusion Matrix: [[2663  242]
 [ 552 2543]]
```

The Multinomial Naive Bayes model performs significantly better than the baseline model, with an accuracy of 86.8% and F1 Score of 86%. Both precision and recall are also much higher, indicating that the model is doing a good job of correctly identifying the sentiment of the reviews.

Logistic Regression

In [28]:
```
# Referenced from: scikit-learn, "sklearn.linear_model.LogisticRegression", 2023.[Online]
# Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

# The documentation has been referenced for guidance from the link above,
# but the code is original and is implemented independently.
```

```
# Initializing Logistic Regression
lr_tfidf = LogisticRegression(random_state=random_seed)
# Referenced from: scikit-learn, "sklearn.linear_model.LogisticRegression", 2023.[Online]
# Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

# The documentation has been referenced for guidance from the link above,
# but the code is original and is implemented independently.

# Initializing Logistic Regression
lr_tfidf = LogisticRegression(random_state=random_seed)

# Fitting the classifier
lr_tfidf.fit(X_train_tfidf, y_train)

# Predicting the test set results
y_pred = lr_tfidf.predict(X_test_tfidf)

# Evaluating the model
lr_tfidf_eval = evaluate_model(y_test, y_pred)

# Printing the evaluation metrics
for metric, value in lr_tfidf_eval.items():
    print(f'{metric}: {value}')
```

```
Accuracy: 0.8851666666666667
Precision: 0.9159751037344398
Recall: 0.8558966074313409
F1 Score: 0.8849173208618674
Confusion Matrix: [[2662  243]
 [ 446 2649]]
```

The Logistic Regression model performs slightly better than the Multinomial Naive Bayes model in comparison to all the evaluation metrics, indicating that the model is doing a slightly better job of correctly identifying the sentiment of the reviews.

Random Forest Classifier

In [29]:

```
# Referenced from: scikit-learn, "sklearn.ensemble.RandomForestClassifier", 2023.[Online]
# Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
```

```
# The documentation has been referenced for guidance from the link above,
# but the code is original and is implemented independently.

# Initializing Random Forest Classifier
rf_tfidf = RandomForestClassifier(random_state=random_seed)

# Fitting the classifier
rf_tfidf.fit(X_train_tfidf, y_train)

# Predicting the test set results
y_pred = rf_tfidf.predict(X_test_tfidf)

# Evaluating the model
rf_tfidf_eval = evaluate_model(y_test, y_pred)

# Printing the evaluation metrics
for metric, value in rf_tfidf_eval.items():
    print(f'{metric}: {value}')
```

```
Accuracy: 0.8611666666666666
Precision: 0.9022048364153628
Recall: 0.8197092084006462
F1 Score: 0.8589808701540546
Confusion Matrix: [[2630  275]
 [ 558 2537]]
```

The Random Forest classifier is slightly outperformed by both the Multinomial Naive Bayes and Logistic Regression models.

K-Fold Cross Validation on the Logistic Regression model

In [30]:

```
# Referenced from: sklearn, "sklearn.model_selection.cross_val_score", 2023.[Online]
# Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

# Referenced from: Ilyasov E, "How to compute precision,recall and f1 score of an
# imbalanced dataset for K fold cross validation?", 2023.[Online]
# Available: https://stackoverflow.com/questions/46598301/
# how-to-compute-precision-recall-and-f1-score-of-an-imbalanced-dataset-for-k-fold

# The code for F1-Score has been referenced and adapted from the Stackoverflow link above.
```

```python
# Initializing Logistic Regression
lr_tfidf = LogisticRegression(random_state=random_seed)

# Performing k-fold cross-validation for accuracy
cv_scores_accuracy = cross_val_score(lr_tfidf, X_train_tfidf, y_train, cv=5, scoring='accuracy')

# Performing k-fold cross-validation for F1-score
scorer = make_scorer(f1_score, average='binary')
cv_scores_f1 = cross_val_score(lr_tfidf, X_train_tfidf, y_train, cv=5, scoring=scorer)

# Printing the cross-validation scores for accuracy
print("Cross-Validation Accuracy Scores:", cv_scores_accuracy)
print("Mean Accuracy:", cv_scores_accuracy.mean())
print("Standard Deviation:", cv_scores_accuracy.std())
print("-" * 90)

# Printing the cross-validation scores for F1-score
print("Cross-Validation F1-Score Scores:", cv_scores_f1)
print("Mean F1-Score:", cv_scores_f1.mean())
print("Standard Deviation:", cv_scores_f1.std())
```

```
Cross-Validation Accuracy Scores: [0.87645833 0.88333333 0.87583333 0.874375   0.880625  ]
Mean Accuracy: 0.878125
Standard Deviation: 0.00333072814861854
------------------------------------------------------------------------------------------
Cross-Validation F1-Score Scores: [0.87320932 0.87810187 0.87155172 0.86973428 0.87626862]
Mean F1-Score: 0.8737731653419498
Standard Deviation: 0.0030505835978032448
```

The Logistic Regression model was selected as the best model based on its superior performance in terms of Accuracy and F1 score compared to the other models. Performing K-fold cross validation on the Logistic Regression model allows for a robust and reliable evaluation of its performance. Cross validation helps mitigate the potential bias and variance that can arise from using a single train-test split. By dividing the data into multiple folds and systematically evaluating the model on different subsets of the data, K-fold cross validation provides a more representative estimate of the model's generalization performance.

The cross-validation Accuracy scores have a mean Accuracy of 0.88 and a standard deviation of 0.0033. These scores indicate that the model achieves high Accuracy consistently across the different folds, providing confidence in its ability to classify sentiment accurately.

Similarly, the cross-validation F1-score scores have a mean F1-score of 0.87 and a standard deviation of 0.0030. The F1-score reflects the model's balance between precision and recall, and the consistent F1-scores across the folds indicate its robustness in capturing both positive and negative sentiments effectively.

Overall, the cross-validation results reinforce the initial evaluation of the Logistic Regression model, confirming its strong performance in sentiment analysis on Spotify reviews. The high mean Accuracy and F1-score, along with low standard deviations, indicate the model's reliability and stability in classifying sentiments across different subsets of the data.

Model Summary

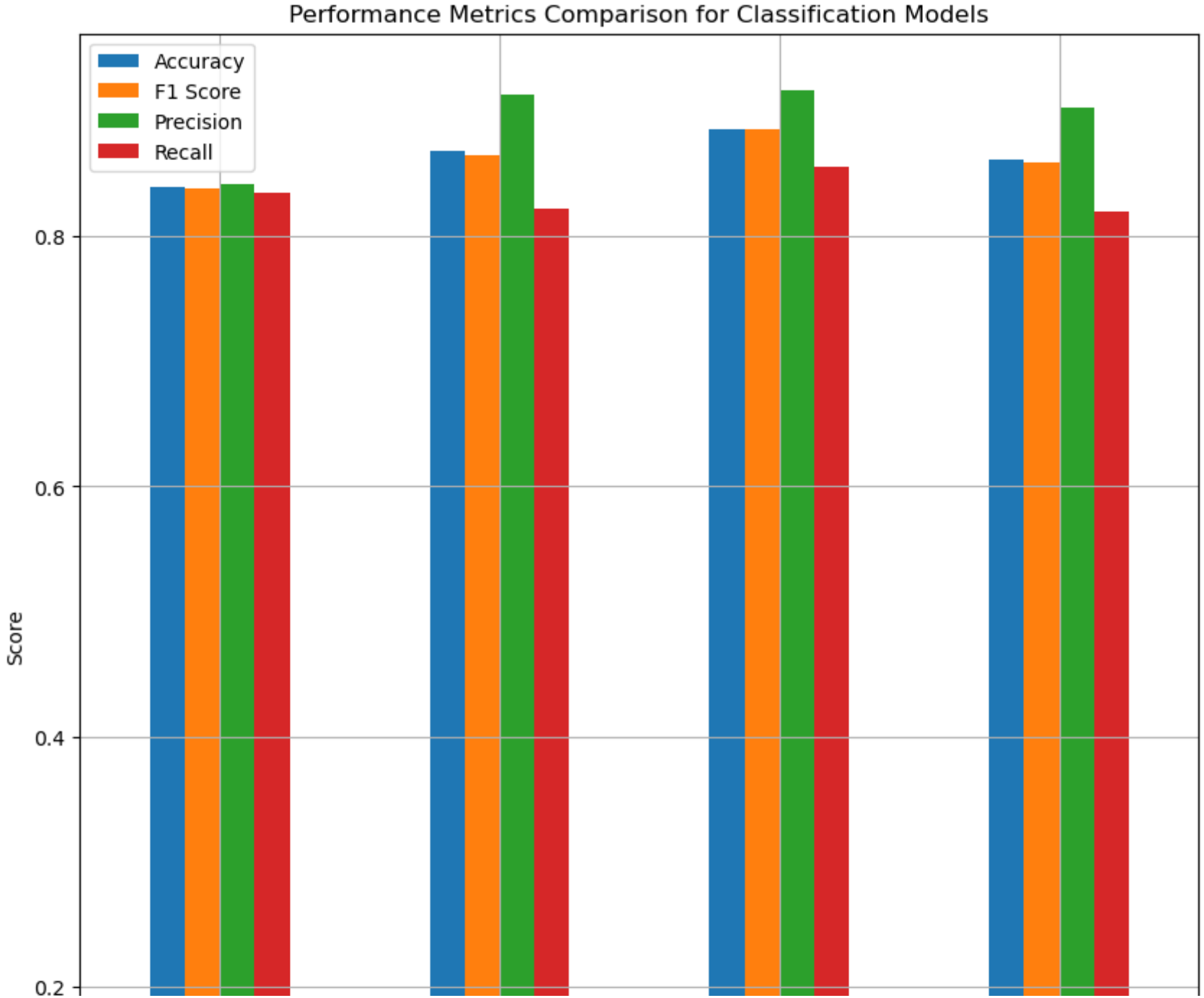
In [31]:
```
# Referenced from: prettytable, "prettytable 3.8.0", 2023.[Online]
# Available: https://pypi.org/project/prettytable/

# The documentation has been referenced for guidance from the link above,
# but the code is original and is implemented independently.

# Creating the table
score_summary = PrettyTable()

# Setting the table format
score_summary.float_format = ".2"  # Set the float format to display two decimal places

# Creating the table header
score_summary.field_names = ["ML Classification Models",
                             "Accuracy",
                             "Precision",
                             "Recall",
                             "F1 Score",
                             "Confusion Matrix"]

# Adding the models with their scores
score_summary.add_rows(
    [
       ["Baseline(Pre-Trained RoBERTa)", round(bert_eval['Accuracy'], 2),
         round(bert_eval['Precision'], 2),
         round(bert_eval['Recall'], 2),
         round(bert_eval['F1 Score'], 2),
```

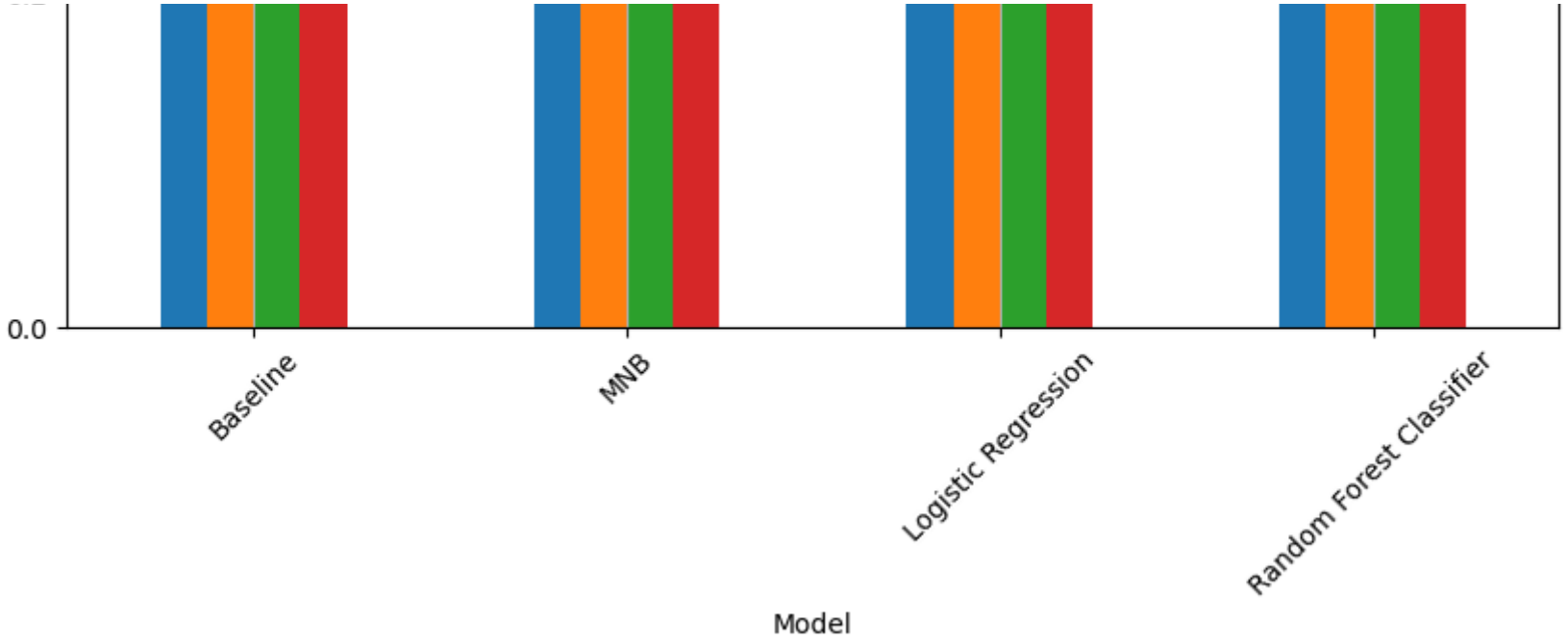
```
         str(bert_eval['Confusion Matrix'])
        ],
        ["MultinomialNB", round(mnb_tfidf_eval['Accuracy'], 2),
         round(mnb_tfidf_eval['Precision'], 2),
         round(mnb_tfidf_eval['Recall'], 2),
         round(mnb_tfidf_eval['F1 Score'], 2),
         str(mnb_tfidf_eval['Confusion Matrix'])
        ],
        ["Logistic Regression", round(lr_tfidf_eval['Accuracy'], 2),
         round(lr_tfidf_eval['Precision'], 2),
         round(lr_tfidf_eval['Recall'], 2),
         round(lr_tfidf_eval['F1 Score'], 2),
         str(lr_tfidf_eval['Confusion Matrix'])
        ],
        ["Random Forest", round(rf_tfidf_eval['Accuracy'], 2),
         round(rf_tfidf_eval['Precision'], 2),
         round(rf_tfidf_eval['Recall'], 2),
         round(rf_tfidf_eval['F1 Score'], 2),
         str(rf_tfidf_eval['Confusion Matrix'])
        ]
    ]
)
# Printing the table
print(score_summary)
```

```
+-------------------------------+----------+-----------+--------+----------+------------------+
|    ML Classification Models   | Accuracy | Precision | Recall | F1 Score | Confusion Matrix |
+-------------------------------+----------+-----------+--------+----------+------------------+
| Baseline(Pre-Trained RoBERTa) |   0.84   |    0.84   |  0.83  |   0.84   |  [[12722  2345]  |
|                               |          |           |        |          |  [ 2476 12457]]  |
|         MultinomialNB         |   0.87   |    0.91   |  0.82  |   0.86   |   [[2663  242]   |
|                               |          |           |        |          |   [ 552 2543]]   |
|      Logistic Regression      |   0.89   |    0.92   |  0.86  |   0.88   |   [[2662  243]   |
|                               |          |           |        |          |   [ 446 2649]]   |
|         Random Forest         |   0.86   |    0.9    |  0.82  |   0.86   |   [[2630  275]   |
|                               |          |           |        |          |   [ 558 2537]]   |
+-------------------------------+----------+-----------+--------+----------+------------------+
```

In [32]:
```
# Initializing a dataframe with the evaluation results
summary = pd.DataFrame({
    'Model': ['Baseline',
```

```
            'MNB',
            'Logistic Regression',
            'Random Forest Classifier'],

    'Accuracy': [bert_eval['Accuracy'],
                 mnb_tfidf_eval['Accuracy'],
                 lr_tfidf_eval['Accuracy'],
                 rf_tfidf_eval['Accuracy']
                ],

    'F1 Score': [bert_eval['F1 Score'],
                 mnb_tfidf_eval['F1 Score'],
                 lr_tfidf_eval['F1 Score'],
                 rf_tfidf_eval['F1 Score']
                ],

    'Precision': [bert_eval['Precision'],
                  mnb_tfidf_eval['Precision'],
                  lr_tfidf_eval['Precision'],
                  rf_tfidf_eval['Precision']
                 ],

    'Recall': [bert_eval['Recall'],
               mnb_tfidf_eval['Recall'],
               lr_tfidf_eval['Recall'],
               rf_tfidf_eval['Recall']
              ]
})

# Plotting the comparison bar chart
summary.set_index('Model').plot(kind='bar', figsize=(10,6))
plt.title('Performance Metrics Comparison for Classification Models')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.grid(visible=True)
plt.legend(loc='upper left')
plt.subplots_adjust(top=1.5)
plt.show()
```


Performance Metrics Comparison for Classification Models
Accuracy
F1 Score
Precision
Recall
Score
0.8
0.6
0.4
0.2



Based on the evaluation results considering accuracy and F1 score metrics, both Logistic Regression and Multinomial Naive Bayes models have emerged as the best performing models in this sentiment analysis project. These models have demonstrated their effectiveness in accurately classifying positive and negative sentiments in Spotify application reviews. Their performance surpasses that of the baseline model and indicates their capability to capture meaningful patterns in the data.

However, it is worth noting that deep learning models could be considered for future studies in sentiment analysis. Deep learning models, such as Recurrent Neural Networks (RNNs) or transformers, have shown promising results in various NLP tasks, including sentiment analysis. These models have the ability to capture complex dependencies and contextual information in textual data, potentially leading to further improvements in sentiment analysis accuracy. (Kurniasari L and Setyanto A)

Additionally, RNNs are well-suited for capturing the temporal characteristics of sequential data, including time series. This makes them valuable in sentiment analysis tasks where the order of words or reviews is significant. By modeling temporal dependencies and context, RNNs can enhance sentiment analysis performance in dynamic situations where sentiment evolves over time. Considering their ability to handle sequential data, RNNs should be considered in future studies of sentiment analysis. (Ghelani S)

References:

1. Kurniasari L and Setyanto A, "Sentiment Analysis using Recurrent Neural Network", 2020.[Online].

Available: https://iopscience.iop.org/article/10.1088/1742-6596/1471/1/012018

2. Ghelani S, "Text Classification — RNN's or CNN's?", 2019.[Online].

Available: https://towardsdatascience.com/text-classification-rnns-or-cnn-s-98c86a0dd361

Conclusion

In summary, the sentiment analysis model developed in this project can be further enhanced by fine-tuning hyperparameters and exploring alternative machine learning models such as XGBoost or Support Vector Machines. Deep learning models like RNNs or transformers also hold potential for capturing complex dependencies in textual data, although they require more computational resources and training data.

An alternative approach for this project involves incorporating domain-specific knowledge or external resources like sentiment lexicons or pre-trained word embeddings. These resources can improve the model's understanding of domain-specific language and sentiment expressions, although their availability and quality may vary across domains.

The contributions of this project extend beyond the context of Spotify app reviews. The methodologies and techniques employed are transferable to other domains, enabling sentiment analysis in e-commerce, hospitality, social media, news, and more. The codebase, algorithms, and methodologies can be shared and replicated using different programming languages, libraries, and algorithms, such as Python-based NLTK or spaCy.

The project's impact lies in its ability to provide valuable insights within the problem area of sentiment analysis. By analyzing customer feedback, it facilitates targeted improvements, decision-making, and user satisfaction enhancement in various industries leading to improved market capture and customer retention, ultimately leading to higher profits. The project's approach serves as a foundation for sentiment analysis, showcasing the effectiveness of NLP techniques, machine learning models, and evaluation metrics in understanding user sentiment which is key for business growth.

Replicating the approach requires careful consideration of domain-specific characteristics. Adjustments to preprocessing steps, feature engineering, or model selection may be necessary to account for unique language, user behavior, or data availability. Nonetheless, the project's methodologies provide a framework for sentiment analysis, fostering further research and practical applications.

Overall, this project extends the boundaries of sentiment analysis, offering adaptable methodologies and techniques applicable across domains. Its contributions to the problem area, transferability, and potential for replication make it a valuable resource for sentiment analysis research and implementation.

References:

Introduction

1. Coursera, "Intro to sentiment analysis", 2023.[Online].
   Available: https://www.coursera.org/learn/uol-cm3060-natural-language-processing/lecture/ylmC2/intro-to-sentiment-analysis

2. Gotting M, "Music streaming worldwide - statistics & facts", 2023.[Online].
   Available: https://www.statista.com/topics/6408/music-streaming/#topicOverview

3. Gotting M, "Music streaming revenue worldwide from 2005 to 2022 (in billion U.S. dollars)", 2023[Online].
   Available: https://www.statista.com/statistics/587216/music-streaming-revenue/

4. Curry D, "Music Streaming App Revenue and Usage Statistics (2023)", 2023.[Online].
   Available:https://www.businessofapps.com/data/music-streaming-market/

5. Malato G, "Precision, recall, accuracy. How to choose?", 2021.[Online].
   Available: https://www.yourdatateacher.com/2021/06/07/precision-recall-accuracy-how-to-choose/#:~:text=Precision%20is%20a%20measure%20of,the%20model%20has%20been%20true.

6. Korstanje J, "The F1 score", 2021.[Online].
   Available: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6

7. Narkhede S, "Understanding Confusion Matrix", 2023.[Online].
   Available: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

Implementation

1. GeeksforGeeks, "Removing stop words with NLTK in Python", 2023.[Online].
   Available: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

2. R Yash, "Python | Lemmatization with NLTK", 2023.[Online].
   Available: https://www.geeksforgeeks.org/python-lemmatization-with-nltk/

3. nikhilaggarwal3, "Python Collections Module", 2023.[Online].
   Available: https://www.geeksforgeeks.org/python-collections-module/

4. scikit-learn, "sklearn.feature_extraction.text.TfidfVectorizer", 2023.[Online].
   Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer

5. Dutta M, "Bag-of-words vs TFIDF vectorization –A Hands-on Tutorial", 2021.[Online].
   Available: https://www.analyticsvidhya.com/blog/2021/07/bag-of-words-vs-tfidf-vectorization-a-hands-on-tutorial/

6. Kaplan D, "Machine Learning 101: CountVectorizer Vs TFIDFVectorizer", 2022.[Online].
   Available: https://enjoymachinelearning.com/blog/countvectorizer-vs-tfidfvectorizer/#:~:text=CountVectorizer%20simply%20counts%20the%20number,is%20to%20the%20whole%20corpus.

7. scikit-learn, "sklearn.preprocessing.LabelEncoder", 2023.[Online].
   Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

8. Siebert C, "siebert/sentiment-roberta-large-english", 2023.[Online].
   Available: https://huggingface.co/siebert/sentiment-roberta-large-english

9. ptrblck, "Training loop getting slower and slower", 2021.[Online].
   Available: https://discuss.pytorch.org/t/training-loop-getting-slower-and-slower/110982

10. Deckers M., "Using CUDA with pytorch?", 2018.[Online].
    Available: https://stackoverflow.com/questions/50954479/using-cuda-with-pytorch

11. tqdm, "tqdm", 2023.[Online].
    Available: https://github.com/tqdm/tqdm

12. Mulla R, "Python Sentiment Analysis Project with NLTK and 🤗 Transformers. Classify Amazon Reviews!!", 2022.[Online].
    Available: https://www.youtube.com/watch?v=QpzMWQvxXWk

13. scikit-learn, "sklearn.naive_bayes.MultinomialNB", 2023.[Online].
    Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

14. scikit-learn, "sklearn.linear_model.LogisticRegression", 2023.[Online].
    Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

15. scikit-learn, "sklearn.ensemble.RandomForestClassifier", 2023.[Online].
    Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

16. sklearn, "sklearn.model_selection.cross_val_score", 2023.[Online]
    Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

17. Ilyasov E, "How to compute precision,recall and f1 score of an imbalanced dataset for K fold cross validation?", 2023.[Online]
    Available: https://stackoverflow.com/questions/46598301/how-to-compute-precision-recall-and-f1-score-of-an-imbalanced-dataset-for-k-fold

18. prettytable, "prettytable 3.8.0", 2023.[Online].
    Available: https://pypi.org/project/prettytable/

19. Kurniasari L and Setyanto A, "Sentiment Analysis using Recurrent Neural Network", 2020.[Online].
    Available: https://iopscience.iop.org/article/10.1088/1742-6596/1471/1/012018

20. Ghelani S, "Text Classification — RNN’s or CNN’s?", 2019.[Online].
Available: https://towardsdatascience.com/text-classification-rnns-or-cnn-s-98c86a0dd361