Spotify Review Sentiment Analysis

Prepared for: Dr. Zhenyu Jin, Ph.D.

Professor, Towson University Marketing Intelligence Graduate Program

Table of Contents

Introduction

In recent years, the landscape of the music industry has undergone a seismic shift with the meteoric rise of music streaming services. These platforms have revolutionized how people discover, access, and engage with music, marking a departure from traditional methods of music consumption such as physical albums and digital downloads. The growing popularity of music streaming can be attributed to a confluence of technological advancements, changing consumer preferences, and evolving business models. This phenomenon has not only transformed the way individuals interact with music but has also reshaped the dynamics of the entire music ecosystem, impacting artists, record labels, and music enthusiasts alike. In this context, understanding the factors driving the ascent of music streaming services is crucial for grasping the contemporary music landscape and its implications for both industry stakeholders and consumers. This overview delves into the burgeoning popularity of music streaming services, exploring the key drivers behind their exponential growth and the profound implications they hold for the future of music consumption.

Spotify stands as a titan in the realm of music streaming, redefining the way millions engage with and enjoy music worldwide. Launched in 2008 by Swedish entrepreneurs Daniel Ek and Martin Lorentzon, Spotify quickly rose to prominence, offering users instant access to an extensive catalog of songs from both major labels and independent artists. With its user-friendly interface, personalized playlists, and seamless cross-platform integration, Spotify has become synonymous with the modern music streaming experience.

The significance of Spotify in the music streaming industry cannot be overstated. As one of the pioneers of the streaming revolution, Spotify has played a pivotal role in reshaping the music

landscape, challenging traditional distribution models, and democratizing access to music. Its innovative features, such as algorithm-driven recommendations, collaborative playlists, and artist discovery tools, have transformed the way users discover and consume music, fostering a global community of music enthusiasts.

Moreover, Spotify's impact extends beyond consumer behavior to influence the entire music ecosystem. By providing artists with a platform to reach a global audience, Spotify has empowered independent musicians, facilitated music discovery, and enabled artists to connect with fans on a more personal level. Additionally, Spotify's data-driven insights and analytics have revolutionized the music industry's approach to marketing, promotion, and artist development, offering unprecedented visibility into listener demographics, preferences, and trends.

Research question : Can machine learning models predict user satisfaction based on review sentiment when Spotify launches updates?

Understanding user satisfaction is paramount for digital platforms like Spotify, as it directly impacts user retention, engagement, and overall success. By delving into user sentiments expressed in reviews, Spotify can gain actionable insights into user preferences, concerns, and expectations regarding platform updates. Predicting user satisfaction before and after updates can empower Spotify to proactively address potential issues, fine-tune feature enhancements, and tailor their updates to better align with user preferences, ultimately fostering a more satisfying user experience.

Previous studies in sentiment analysis and user feedback prediction have laid a foundation for this research question. For instance, research by Liu et al. (2019) utilized sentiment analysis to predict user satisfaction with mobile applications, demonstrating the efficacy of machine learning models in extracting valuable insights from user reviews. Similarly, the study by Hu et al. (2020) investigated the relationship between sentiment expressed in customer reviews and user satisfaction in the context of e-commerce platforms, providing valuable insights into the predictive power of sentiment analysis for user satisfaction.

Methodology

This dataset comprises reviews of the Spotify app collected from the Google Play Store during the period from January 1, 2022, to July 9, 2022. The data collection process involved scraping reviews directly from the Google Play Store platform. With a total of 61,594 rows, this dataset offers a substantial volume of user-generated feedback and opinions regarding the Spotify application's performance, features, and overall user experience during the specified timeframe. The dataset was obtained from Kaggle, a prominent platform for accessing and sharing datasets for various research and analytical purposes.

Preprocessing Steps for Sentiment Analysis, Data Cleaning

For the sentiment analysis of Spotify reviews, a comprehensive preprocessing pipeline was implemented using a customized library tailored to the unique characteristics of the dataset. The preprocessing steps involved are below and the customized library is in the appendix.

- Text Cleaning: Removal of punctuation, special characters, and irrelevant symbols to standardize the text and enhance consistency.
- Lowercasing: Conversion of all text to lowercase to ensure uniformity and mitigate discrepancies due to case sensitivity.
- Tokenization: Segmentation of the text into individual tokens or words to facilitate further analysis and feature extraction.
- Stopword Removal: Elimination of common stopwords to focus on meaningful words and reduce noise in the dataset.

The sentiment analysis approach adopted in this study involved supervised learning using manually pre-labeled data. A dataset consisting of 1,200 pre-labeled reviews was utilized for training the sentiment prediction models. These reviews were categorized into positive and negative sentiments based on their content.

Sentiment Analysis

For our sentiment analysis project, manual labeling of data played a crucial role in our methodology. We assigned each group member a specific track, with Track 3 designated for manual labeling. The objective of this manual labeling was to establish a labeled dataset, facilitating the training of machine learning models for sentiment analysis.

- Manual Labeling Process:Track 5 was chosen for manual labeling, and each group member undertook the responsibility of labeling 400 data points within this track. This process entailed categorizing each data point as positive, negative, or neutral based on the

sentiment expressed in the text. By meticulously labeling the data, we ensured the development of a high-quality labeled dataset, serving as the foundation for training our machine learning models.

- Machine Learning Model Training:Following the manual labeling phase, we leveraged the labeled dataset to train machine learning models for sentiment analysis. We experimented with various algorithms, including Stochastic Gradient Descent (SGD), Naive Bayes, and Logistic Regression, to identify the most effective model for our task. These models were trained using the labeled dataset, utilizing features extracted from the text data to predict sentiment labels.
- Sentiment Analysis on Unlabeled Data:Subsequent to model training and evaluation, we applied the trained models to perform sentiment analysis on the remaining unlabeled data. This process involved predicting sentiment labels for each data point in the unlabeled dataset based on the insights gleaned from the trained models. By extrapolating sentiment labels to the unlabeled data, we gained valuable insights into the prevailing sentiment trends, patterns, and sentiments across the entire dataset.
- Confusion Matrix and Model Tuning: Additionally, we constructed confusion matrices to assess the performance of our models in classifying sentiment. Furthermore, we conducted model tuning to optimize the performance of our sentiment analysis models, ensuring the most accurate and reliable predictions possible.

Overview of Machine Learning Models for Sentiment Prediction

Two primary machine learning models were employed for sentiment prediction:

1. Bag-of-Words (BOW): The BOW model was used to represent the review text as a collection of words, disregarding grammar and word order but retaining multiplicity. Each review was transformed into a vector of word occurrences or frequencies, serving as input features for sentiment prediction models.
2. Naive Bayes Classifier: A Naive Bayes classifier was selected as the primary sentiment prediction model. This probabilistic classifier calculates the likelihood of a review belonging to a particular sentiment class (positive or negative) based on the occurrence of words in the review. The model was trained on the pre-labeled dataset using BOW representations of reviews as input features.

By leveraging these machine learning models and preprocessing techniques, the sentiment analysis pipeline accurately predicted the sentiment of Spotify reviews, providing valuable insights into user satisfaction levels before and after Spotify updates.

Refer the codes in the Appendix

Quantitative Analysis

Sentiment analysis is on the beginning of what can be done with the Spotify Data. Conducting sentiment analysis allows us to conduct regression to get a clear picture of the story that the data is trying to tell. We utilized t-test and linear regression in order to determine the difference between groups and answer the question of how users feel about the Spotify app after its software updates.

We initially conducted a t-test to determine if there was a significant difference between likes of positive reviews and negative reviews. The results revealed that there was a significant difference between the two groups. Negative Reviews received on average 10 likes, while positive reviews on average received 4 likes. This likely means that other users resonated more with the negative reviews about the Spotify app versus the positive reviews praising the application.

Digging deeper into the data we conducted a linear regression in order to determine what had more of an effect on the Rating (1-5) of the Spotify app. The model reveals that negative reviews decrease the app rating by 2.259 points, while total likes or “thumbs up” did not significant effect on the app rating. The model only explains 44.36% of the variance is the ratings, which means they are other factors that are effecting the rating of the app. The data we were able to obtain had a limited number of variables to work with. Overall, the regression model is a good fit for the data.

The results show that a significant number of users resonate with negative reviews, and these reviews are associated with lower ratings compared to positive reviews. The Spotify team can leverage this information to identify areas where improvements are needed, guiding their engineers to implement software patches. Maintaining a high rating in the Google Play Store is crucial for sustaining growth in the competitive streaming service market.

Conclusion

The analysis of Spotify reviews provides valuable insights into user sentiments and their impact on app ratings. Through sentiment analysis, we uncovered significant differences between positive and negative reviews, with negative reviews garnering more likes on average, indicating greater resonance among users. Additionally, our regression analysis revealed that negative reviews have a considerable negative effect on app ratings, highlighting the importance of addressing user concerns and improving user satisfaction to maintain a high app rating.

These findings underscore the importance of actively monitoring user feedback and addressing issues promptly to enhance user experience and app performance. By incorporating user feedback into product development cycles and fostering a culture of continuous improvement, Spotify can ensure a compelling and satisfying user experience for its millions of users worldwide.

Appendix

```
import pandas as pd
import numpy as np
import nltk
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore")
import os
os.chdir('C:/Users/Asus/Desktop/MI SEM II/CONSUMER SENTIMENT')
# create a data frame by reading the csv file 'movie_reviews_week10m.csv'
spotify_review = pd.read_csv('spotify_review.csv')

Spotify_review.shape
spotify_review.head(10)
import nltk
import spacy
import re
from nltk.corpus import wordnet
from nltk.tokenize.toktok import ToktokTokenizer
import myText_normalizer_Ayushi as Ayushi
review = np.array(spotify_review['Review'])
normalized_review = Ayushi.myText_normalizer(review)
for review in normalized_review[:10]:
    print(review)
spotify_review["Cleaned_Review"] = normalized_review

spotify_review.head(10)
reviews = np.array(spotify_review['Cleaned_Review'])
sentiments = np.array(spotify_review['Pre Lable'])
# dividing data into train and test reviews 70:30%

train_reviews = reviews[:840]
train_sentiments = sentiments [:840]
test_reviews = reviews[840:1199]
```

```
test_sentiments = sentiments[840:1199]
#Vectorizing with both BOW and TFIDF

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# BOW as the vectorizer
cv = CountVectorizer()
cv_train_features = cv.fit_transform(train_reviews)

# TF-IDF as the vectorizer
tv = TfidfVectorizer()
tv_train_features = tv.fit_transform(train_reviews)

# transform test reviews into features using BOW as the vectoizer
cv_test_features = cv.transform(test_reviews)

# transform test reviews into features using IF-IDF as the vectoizer
tv_test_features = tv.transform(test_reviews)
print('BOW model:> Train features shape:', cv_train_features.shape, ' Test features shape:', cv_test_features.shape)
print('TFIDF model:> Train features shape:', tv_train_features.shape, ' Test features shape:', tv_test_features.shape)
#Machine learning model

from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score

mnb = MultinomialNB()
lr = LogisticRegression()
sgd = SGDClassifier()
# train our model using Naïve Bayes model on BOW features

mnb.fit(cv_train_features, train_sentiments)

# predict the sentiments of the testing reviews
predicted_seni_bow_mnb = mnb.predict(cv_test_features)
```

```
# get the accuracy of the prediction
accuracy_1 = accuracy_score(predicted_seni_bow_mnb, test_sentiments )
precision_1 = precision_score(test_sentiments, predicted_seni_bow_mnb, pos_label='Positive')
recall_1 = recall_score(test_sentiments, predicted_seni_bow_mnb, pos_label='Positive')


print (accuracy_1)
print (precision_1)
print (recall_1)
# train our model using Naïve Bayes model on TFIDF features
mnb.fit(tv_train_features, train_sentiments)


# predict the sentiments of the testing reviews

predicted_seni_tv_mnb =mnb.predict(tv_test_features)


# get the accuracy of the prediction

accuracy_2 = accuracy_score(predicted_seni_tv_mnb, test_sentiments)
precision_2 = precision_score(predicted_seni_tv_mnb, test_sentiments, average='macro')
recall_2 = recall_score(predicted_seni_tv_mnb, test_sentiments, average='macro')

print (accuracy_2)
print (precision_2)
print (recall_2)
# train our model using logistic regression model on BOW features

lr.fit(cv_train_features, train_sentiments)

# predict the sentiments of the testing reviews
predicted_seni_bow_lr = lr.predict(cv_test_features)


# get the accuracy of the prediction
```

```
accuracy_3 = accuracy_score(predicted_seni_bow_lr, test_sentiments)
precision_3 = precision_score(predicted_seni_bow_lr, test_sentiments, average='macro')
recall_3 = recall_score(predicted_seni_bow_lr, test_sentiments, average='macro')

print (accuracy_3)
print (precision_3)
print (recall_3)
# train our model using logistic regression model on tfidf features
lr.fit(tv_train_features, train_sentiments)

# predict the sentiments of the testing reviews
predicted_seni_tv_lr = lr.predict(tv_test_features)

# get the accuracy of the prediction
accuracy_tv_lr = accuracy_score( predicted_seni_tv_lr, test_sentiments)
accuracy_tv_lr
accuracy_4 = accuracy_score(predicted_seni_tv_lr, test_sentiments)
precision_4 = precision_score(predicted_seni_tv_lr, test_sentiments, average='macro')
recall_4 = recall_score(predicted_seni_tv_lr, test_sentiments, average='macro')

print (accuracy_4)
print (precision_4)
print (recall_4)
# train our model using SGD model on BOW features

sgd.fit(cv_train_features, train_sentiments)

# predict the sentiments of the testing reviews

predicted_seni_bow_sgd = sgd.predict(tv_test_features)



# get the accuracy of the prediction

accuracy_5 = accuracy_score(predicted_seni_bow_sgd, test_sentiments)
precision_5 = precision_score(predicted_seni_bow_sgd, test_sentiments, average='macro')
```

```
recall_5 = recall_score(predicted_seni_bow_sgd, test_sentiments, average='macro')

print (accuracy_5)
print (precision_5)
print (recall_5)
# train our model using SGD model on tfidf features

sgd.fit(tv_train_features, train_sentiments)

# predict the sentiments of the testing reviews

predicted_seni_tv_sgd = sgd.predict(tv_test_features)

# get the accuracy of the prediction

accuracy_6 = accuracy_score(predicted_seni_tv_sgd ,test_sentiments)
precision_6 = precision_score(predicted_seni_tv_sgd , test_sentiments, average='macro')
recall_6 = recall_score(predicted_seni_tv_sgd, test_sentiments, average='macro')

print (accuracy_6)
print (precision_6)
print (recall_6)




data = [
   ['Naive Bayes', accuracy_1, accuracy_2, precision_1, precision_2, recall_1, recall_2],
   ['Logistic Regression', accuracy_3, accuracy_4, precision_3, precision_4, recall_3, recall_4],
   ['SGD Model', accuracy_5, accuracy_6, precision_5, precision_6, recall_5, recall_6]
]

columns = [
        'Model', 'BOW_Accuracy', 'TFDIF_Accuracy', 'BOW_Precision', 'TFDIF_Precision', 'BOW_Recall',
'TFIDF_Recall'
]
```

```
df = pd.DataFrame(data, columns=columns).T
print(df)
# using just the Accuracy method we are going to use the Naive Bayes Bag of Words model because it has the highest value
spotify_review.head(1300)

# Define unlabeled_data as the subset of dataset starting from the 1200th row
unlabeled_review = spotify_review.iloc[1200:]
cv_unlabeled_features = cv.transform(unlabeled_review['Cleaned_Review'])

predicted_sentiments_bow_mnb = mnb.predict(cv_unlabeled_features)

# Adding a column 'predicted_sentiment' to the unlabeled_data DataFrame
unlabeled_review['predicted_sentiment'] = predicted_sentiments_bow_mnb

# Returning the first few rows of the updated DataFrame
unlabeled_review.head(1300)
unlabeled_review.to_csv('updated_spotify_review.csv', index=False)

from sklearn.metrics import confusion_matrix

# Calculate the confusion matrix
conf_matrix = confusion_matrix(test_sentiments, predicted_seni_bow_mnb)

conf_matrix_df = pd.DataFrame(conf_matrix)

conf_matrix_df.index.name = 'Actual'
conf_matrix_df.columns.name = 'Predicted'
conf_matrix_df
predicted_proba = mnb.predict_proba(tv_test_features).max(axis=1)

# Create a data frame based on test data by adding columns Predicted Name and Predict confidence

# Create a DataFrame based on the test data
test_data_df = pd.DataFrame({'Review': test_reviews, 'Sentiment': test_sentiments})
```

```
test_data_df['Predict Confidence'] = predicted_proba
test_data_df

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
# Define the pipeline with CountVectorizer and Naive Bayes classifier
nb_pipeline = Pipeline([
    ('bow', CountVectorizer()),
    ('nb', MultinomialNB())
])

# Define the parameters grid
param_grid = {
    'bow__ngram_range': [(1, 1), (1, 2)],
    'nb__alpha': [0.1, 0.5, 1.0]
}

gs_nb = GridSearchCV(nb_pipeline, param_grid, cv=5, verbose=2)

gs_nb.fit(train_reviews, train_sentiments)
predicted_labels_tune = gs_nb.predict(test_reviews)
accuracy_tuned = accuracy_score(test_sentiments, predicted_labels_tune)
print("Accuracy of the tuned model:", accuracy_tuned)
predicted_proba_tune = gs_nb.predict_proba(test_reviews).max(axis = 1)
test_data_df['predict confidence Tuned'] = predicted_proba_tune
test_data_df
```

Customized Library for Normalizing the Data-

```
get_ipython().system('pip install spacy')
import nltk
import spacy
import re
from nltk.corpus import wordnet
from nltk.tokenize.toktok import ToktokTokenizer


# In[ ]:


tokenizer = ToktokTokenizer()
raw_list = nltk.corpus.stopwords.words('english')
stopwords = set(raw_list)-{"not","no"}


# In[ ]
CONTRACTION_MAP = {
    "ain't": "is not","aren't": "are not","can't": "cannot","can't've": "cannot have","'cause": "because","could've": "could have","couldn't": "could not","couldn't've": "could not have","didn't": "did not","doesn't": "does not","don't": "do not","hadn't": "had not","hadn't've": "had not have","hasn't": "has not","haven't": "have not","he'd": "he would",
     "he'd've": "he would have","he'll": "he will","he'll've": "he he will have","he's": "he is","how'd": "how did","how'd'y": "how do you","how'll": "how will","how's": "how is","I'd": "I would","I'd've": "I would have","I'll": "I will","I'll've": "I will have","I'm": "I am","I've": "I have","i'd": "i would","i'd've": "i would have","i'll": "i will","i'll've": "i will have","i'm": "i am","i've": "i have","isn't": "is not","it'd": "it would","it'd've": "it would have","it'll": "it will","it'll've": "it will have","it's": "it is","let's": "let us","ma'am": "madam","mayn't": "may not","might've": "might have","mightn't": "might not", "mightn't've": "might not have","must've": "must have","mustn't": "must not","mustn't've": "must not have","needn't": "need not","needn't've": "need not have","o'clock": "of the clock","oughtn't": "ought not","oughtn't've": "ought not have", "shan't": "shall not","sha'n't": "shall not", "shan't've": "shall not have","she'd": "she would","she'd've": "she would have","she'll": "she will", "she'll've": "she will have","she's": "she is","should've": "should have","shouldn't": "should not","shouldn't've": "should not have","so've": "so have","so's": "so as","that'd": "that would","that'd've": "that would have","that's": "that is", "there'd": "there would","there'd've": "there would have","there's": "there is",
```

```
    "they'd": "they would","they'd've": "they would have","they'll": "they will","they'll've": "they will have","they're": "they are", "they've": "they have", "to've": "to have","wasn't": "was not", "we'd": "we would", "we'd've": "we would have","we'll": "we will","we'll've": "we will have","we're": "we are","we've": "we have","weren't": "were not","what'll": "what will","what'll've": "what will have","what're": "what are","what's": "what is","what've": "what have","when's": "when is","when've": "when have","where'd": "where did","where's": "where is","where've": "where have","who'll": "who will", "who'll've": "who will have", "who's": "who is", "who've": "who have", "why's": "why is","why've": "why have", "will've": "will have", "won't": "will not","won't've": "will not have","would've": "would have","wouldn't": "would not","wouldn't've": "would not have", "y'all": "you all", "y'all'd": "you all would","y'all'd've": "you all would have","y'all're": "you all are","y'all've": "you all have","you'd": "you would","you'd've": "you would have","you'll": "you will", "you'll've": "you will have", "you're": "you are","you've": "you have"
}


# In[ ]:


def my_lowercase (text):
    text = text.lower()
    return text


def my_expanding (text, mapping):
    for key, value in mapping.items():
        text = re.sub(key, value, text)
    return text

def my_stopwords (text):
    tokens = tokenizer.tokenize(text)
    filtered_tokens = [token for token in tokens if token not in stopwords]
    text = ' '.join(filtered_tokens)
    return text


def my_specialChar_remover(text):
    pattern = r"[@$#:)'!]"
```

```
    cleaned_text = re.sub(pattern, '', text)
    return cleaned_text

def my_annoyingChar_remover (text):
    text = re.sub(r'[\t\r\n|]+', '', text)
    text = re.sub(' +',' ',text)            # ' +' matches sequences of one or more consecutive spaces.
    text = re.sub(r"[^a-zA-Z\s]", '', text)
    return text
def myText_normalizer (corpus, f1 = True, f2 = True, f3 = True, f4 = True, f5= True, f6=True):
    normalized_corpus = []
    for doc in corpus:
        if f1:
            doc = my_lowercase(doc)
        if f2:
            doc = my_expanding(doc,CONTRACTION_MAP)
        if f3:
            doc = my_stopwords (doc)
        if f4:
            doc = my_specialChar_remover(doc)
        if f5:
            doc = my_annoyingChar_remover (doc)

        normalized_corpus.append(doc)
    return normalized_corpus
```