Building a music recommendation system (Using Spotify dataset)

Mounika Gampa (A20488077)

Ansh Shrivastava(A20481422)

1.Project Requirements:

1. Windows

2. Google colabs.

3. Python.

2.Dataset link

DataLink

3. Abstract

Our team of two has created a system that recommends music based on your mood. We used a big dataset of music from Spotify to do this. The dataset had lots of information about each song, like how fast it is, how happy or sad it sounds, and other stuff like that. We asked people to describe their mood with words like "bright," "excited," "calm," "sad," or "healing." Then we used special computer tools to find songs in the dataset that match their mood.

To make it work, we had to turn all the song information into numbers that a computer could understand. This was done using tfidf vectorizer. We then used a special math tool called cosine similarity to match up the songs in the dataset with the words people used to describe their mood. The more similar the words and the songs were, the higher the recommendation score for that song.

We believe that our music recommendation system will be helpful to people who want to find songs that match their mood and help them relax.

3.1. Introduction

In today's highly competitive online world, it is becoming increasingly important for companies to provide their users with personalized content to attract and retain their attention. Spotify, the popular music social media company, has capitalized on this by utilizing a recommendation system to provide its over 300 million active users with the best-fit songs based on their search history and playlists. This approach is part of a larger movement towards recommendation systems that make suggestions based on a person's personal viewing history or the combined viewing histories of themselves and their friends. In the case of song prediction, recommendation algorithms aim to anticipate or suggest songs that a user may like based on their data or the data of all users in the database.

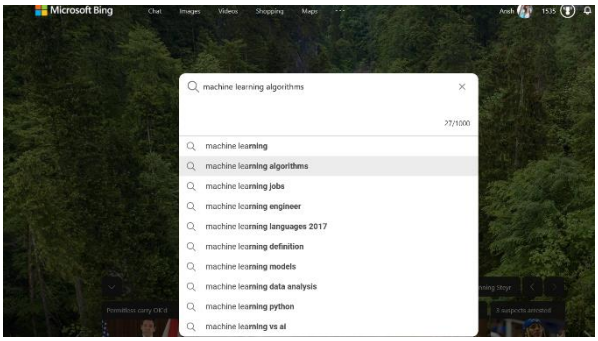
This report delves deeper into the principles and source code of recommendation systems, with a specific focus on song prediction in Spotify. The report outlines the conceptual pipeline that illustrates the process of recommending a song and delves into the specifics of how to achieve this and the many types of recommendation systems available. The report highlights that cluster-based methods have limitations in incorporating additional information, such as a

categorization predictor. In contrast, content-based filtering and collaborative filtering can be used together to create a hybrid recommendation system that utilizes both the clustering result and additional information.

Overall, this report emphasizes the importance of utilizing recommendation systems in today's digital world to provide users with personalized content that meets their needs and keeps them engaged. It also highlights the potential of hybrid recommendation systems to provide an even more tailored experience to users.

4. Example for this type of systems:

Image by name recommend images on Microsoft Bing is one of the best types of models we can use to create this kind of system and serves as a perfect example. In the Bing image recommendation system while the user wants to search the image by name then the recommended images will be displayed with different categories at the same time



Using Google Images categories to represent different moods in a music recommendation system is a unique approach. Similarly in our project users can select a mood category and receive a personalized playlist of songs curated to fit that mood. The system is flexible, allowing users to switch between mood categories to receive a new set of songs. This approach has the potential to improve user engagement and satisfaction in the music streaming industry by providing a more tailored and personalized experience.

5.Implementation

We'll start by examining the Spotify data's properties in view of the data cleaning from Part I. The data will next be processed using feature engineering so that it can be given into the algorithm used for the content-based filtering as well as the similarity metric. Finally, we will discuss briefly several metrics and thresholds related to the model.

5.1 Data Selection and Filtering

The data selection process involves two tasks, the first of which is removing duplicate music releases. Since the imported data is from Spotify playlists, it is necessary to eliminate duplicate tracks that appear in different playlists. This is done by comparing the track titles and artist names to ensure that the same track is not mistakenly removed. The process is carried out using Pandas data frame manipulation, which simplifies the procedure.

5.2 Features

The process of data selection involves two tasks. The first task is to identify and remove duplicate music releases. This is crucial since the data used in this project is imported from Spotify playlists, which may contain multiple instances of the same track. To eliminate duplicates, the names of the artists and track titles are collected and compared to each other. This process is carried out through the manipulation of pandas data frames.

Once the duplicate tracks are removed, the data is transformed using one-hot encoding, a standard technique for converting categorical data into computer-friendly features. Each category is represented as a column, and the values in each row indicate whether or not the category is present. In this way, we can create a matrix of features that represents each song in terms of its associated genres.

However, Spotify's genre distribution is highly unbalanced, with some genres being more prevalent than others. Additionally, a single artist or song may be associated with multiple genres, making it difficult to establish the significance of each genre. To address this issue, we use TF-IDF metrics to weigh the importance of each genre in the dataset.

TF-IDF, or Term Frequency-Inverse Document Frequency, is a method for measuring the significance of words in a collection of texts. In the context of this project, the songs serve as the texts, and the genres are the words. The goal of TF-IDF is to highlight the importance of a genre in the dataset by considering its frequency within each song and across all songs.

To compute the TF-IDF score for each genre, we calculate the Term Frequency (TF), which is the number of times a genre appears in a song divided by the total number of words in the song. We then calculate the Inverse Document Frequency (IDF), which is the logarithm of the total number of songs divided by the number of songs in which the genre appears. By multiplying the TF and IDF scores, we obtain the final TF-IDF score for each genre.

TF-IDF = Term Frequency x Inverse Document Frequency.

This approach is superior to one-hot encoding because it considers the significance of each genre in the dataset. Common genres that appear frequently across all songs will have lower TF-IDF scores, while less common genres will have higher scores. This helps to prevent rare genres from being overrepresented in the analysis.

5.3 Process

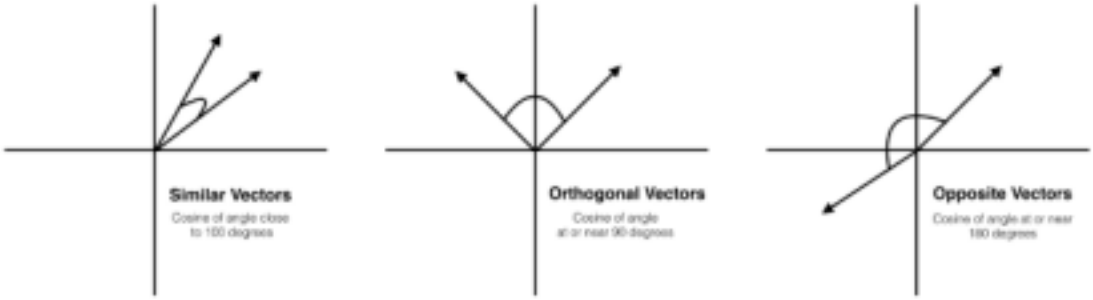
5.3.1 Summarization:

This stage involves compiling a playlist of songs into a vector that can be compared to every other song in the dataset to identify similarities. To start, we can classify songs into those that are in the playlist and those that aren't. Since we don't want to endorse any of the songs in the playlist, it's crucial to remove them. Then, using the dataset we already created in the previous step, we identify the attributes of those songs. As a result, it's critical that our dataset contains many songs to reduce the likelihood that the playlist at this stage will contain no matching songs. Finally, we combine the feature values of each song on the playlist as summarization vector.

5.3.2 Similarity and recommendation:

In the similarity and recommendation stage, the playlist summarized vector and the non-playlist tracks are used to identify how closely each song in the database matches the playlist. The comparison metric used for this purpose is cosine similarity.

Cosine similarity is a mathematical measurement that gauges the similarity between two vectors. By visualizing the music vectors as two-dimensional objects, we can compute their cosine similarity, which measures the cosine of the angle between them. A higher cosine similarity value indicates that the two songs are more similar to each other. This similarity metric is then used to recommend new songs to the user based on their similarity to the songs in the playlist. It would resemble the image below.



The two vectors are comparable once they are roughly pointing in the same direction. This is also the reason we didn't calculate the song average but instead just added them all up. Since the song vectors in our case are hyperdimensional, a graph cannot effectively depict the situation.

The Scikit Learn library has a cosine similarity function. The mathematical intuition, however, remains the same.

Formally, the mathematical formula can be expressed as:

$$\text{Cosine Sim}(A, B) = \frac{A \cdot B}{||A|| \times ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

6. Results -

6.1. To begin with, we eliminated any duplicate data entries

6.2. Next, we extracted specific columns from the data

[71]:
```
##removinf duplicate entries and also choosing calums on the basis of attributes


# Removing duplicate songs in data
def drop_duplicates(df):
    df['artists_song'] = df.apply(lambda row: row['artist_name']+row['track_name'],axis = 1)
    return df.drop_duplicates('artists_song')

songsDataFrame = drop_duplicates(DataFrame_playlist)

# Choose helpful columns
def Helpful_columns(df):
       return df[['artist_name','id','track_name','danceability', 'energy', 'key', 'loudness', 'mode',
       'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', "artist_pop", "genres", "track_pop"]]
songsDataFrame = Helpful_columns(songsDataFrame)
songsDataFrame.head()
```

6.3. The data within the “genres” column was transformed into a list structure

```
def genre_preprocess(df):
    df['genres_list'] = df['genres'].apply(lambda x: x.split(" "))
    return df
songsDataFrame = genre_preprocess(songsDataFrame)
songsDataFrame['genres_list'].head()
```

```
0     [dance_pop, hip_hop, hip_pop, pop, pop_rap, r&...
6                       [dance_pop, pop, post-teen_pop]
19                                [dance_pop, pop, r&b]
46                                     [dance_pop, pop]
55                               [pop_rap, reggae_fusion]
Name: genres_list, dtype: object
```

6.4. One-hot encoding was applied to the categorized data in order to convert it into machine–readable features.

```
def tfidf_oheprep(df, column, new_name):
    tf_df = pd.get_dummies(df[column])
    feature_names = tf_df.columns
    tf_df.columns = [new_name + "|" + str(i) for i in feature_names]
    tf_df.reset_index(drop = True, inplace = True)
    return tf_df
```

```
# encoding
subject_ohe = tfidf_oheprep(sentiment, 'subjectivity','subject')
subject_ohe.iloc[0]
```

```
subject|high      False
subject|low        True
subject|medium    False
Name: 0, dtype: bool
```

6.5. We utilized the tf-idf vectorizer as demonstrated in the following steps

```
# implementation


tfidf = TfidfVectorizer()
tfidf_Definematrix =  tfidf.fit_transform(songsDataFrame['genres_list'].apply(lambda x: " ".join(x)))
genre_df = pd.DataFrame(tfidf_Definematrix.toarray())
genre_df.columns = ['genre' + "|" + i for i in tfidf.get_feature_names_out()]
genre_df.drop(columns='genre|unknown')
genre_df.reset_index(drop = True, inplace=True)
genre_df.iloc[0]
```

```
genre|21st_century_classical    0.0
genre|432hz                     0.0
genre|_hip_hop                  0.0
genre|_roll                     0.0
genre|a_cappella                0.0
                               ...
genre|zambian_hip_hop           0.0
genre|zhongguo_feng             0.0
genre|zolo                      0.0
genre|zouk                      0.0
genre|zouk_riddim               0.0
Name: 0, Length: 2147, dtype: float64
```

6.6. To enhance the outcomes, we employed min-max scalar for the “pop” column

```
pop = songsDataFrame[["artist_pop"]].reset_index(drop = True)
scaler = MinMaxScaler()
pop_scaled = pd.DataFrame(scaler.fit_transform(pop), columns = pop.columns)
pop_scaled.head()
```

	artist_pop
0	0.74
1	0.84
2	0.86
3	0.82
4	0.75

6.7. The code below illustrates the implementation of cosine similarity

```
def Recommend_generate_playlist(df, features, nonplaylist_features):
    non_playlist_df = df[df['id'].isin(nonplaylist_features['id'].values)]
    # Find cosine similarity between the playlist and the complete song set
    non_playlist_df['sim'] = cosine_similarity(nonplaylist_features.drop('id', axis = 1).values, features.values.reshape(1, -1))[:,0]
    non_playlist_df_top_40 = non_playlist_df.sort_values('sim',ascending = False).head(40)

    return non_playlist_df_top_40
```

```
# Create a list of the top 10 suggestions
Suggestions = Recommend_generate_playlist(songsDataFrame, complete_feature_set_playlist_vector, complete_feature_set_nonplaylist)
Suggestions.head(10)
```

```
C:\Users\anshs\AppData\Local\Temp\ipykernel_26328\287525853.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  non_playlist_df['sim'] = cosine_similarity(nonplaylist_features.drop('id', axis = 1).values, features.values.reshape(1, -1))[:,0]
```

	artist_name	id	track_name	danceability	energy	key	loudness	mode	speechiness	acousticness	...	liveness	valence	te
28834	American Authors	64ybTt8CKxPdeXBNnu08Op	Believer	0.583	0.968	1	-2.909	1	0.0368	0.001410	...	0.1300	0.910	11

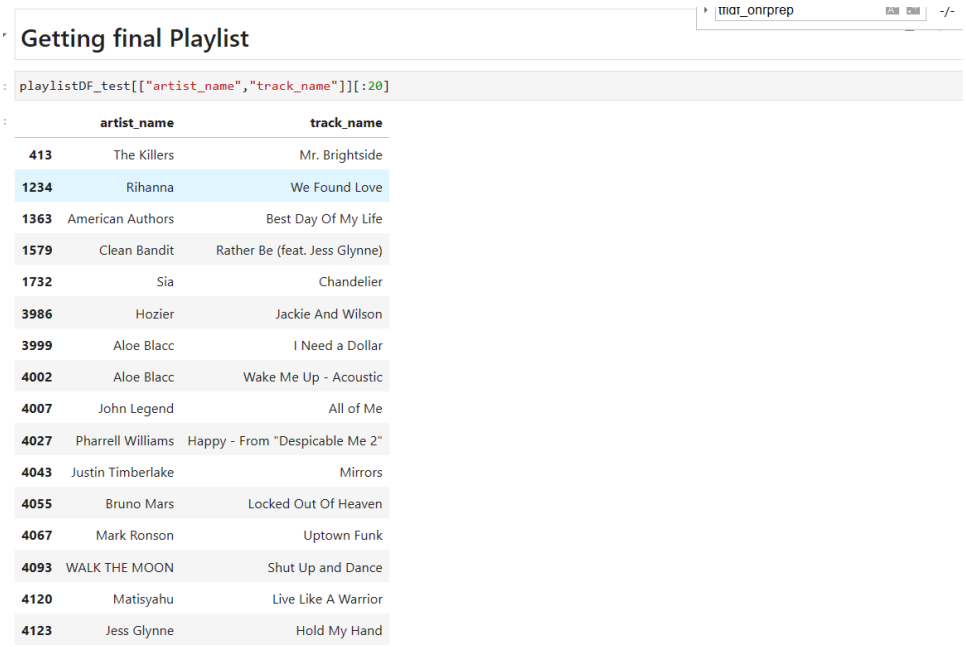
6.8. We currently utilizing the model that we developed to recommend songs based on the test playlist

```
# Create a list of the top 10 suggestions
Suggestions = Recommend_generate_playlist(songsDataFrame, complete_feature_set_playlist_vector, complete_feature_set_nonplaylist)
Suggestions.head(10)
```

```
C:\Users\anshs\AppData\Local\Temp\ipykernel_26328\287525853.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  non_playlist_df['sim'] = cosine_similarity(nonplaylist_features.drop('id', axis = 1).values, features.values.reshape(1, -1))[:,0]
```

	artist_name	id	track_name	danceability	energy	key	loudness	mode	speechiness	acousticness	...	liveness	valence	te
28834	American Authors	64ybTt8CKxPdeXBNnu08Op	Believer	0.583	0.968	1	-2.909	1	0.0368	0.001410	...	0.1300	0.910	11
51128	American Authors	1obisQNOcikRvTdStbW3pG	Go Big Or Go Home	0.665	0.875	1	-4.272	1	0.0426	0.009390	...	0.0897	0.660	12
43254	The 1975	51cd3bzVmLAjlnsSZn4ecW	She's American	0.647	0.857	1	-3.940	1	0.0547	0.167000	...	0.0763	0.550	11
28926	Neon Trees	0K1KOCeJBj3IpDYxEX9qP2	Sleeping With A Friend	0.582	0.882	2	-4.256	1	0.0355	0.001890	...	0.3200	0.507	10
54403	American Authors	4gHD93RNqEhEh2NkYzl3x6	Luck	0.554	0.806	0	-3.463	1	0.0460	0.001770	...	0.1650	0.646	14
55426	WALK THE MOON	71wT7aMCFPYfzutF66OLac	Aquaman	0.630	0.772	1	-6.986	1	0.0297	0.510000	...	0.0881	0.721	9
44455	Neon Trees	1fBl642IhJOE5U319Gy2Go	Animal	0.482	0.833	5	-5.611	1	0.0449	0.000346	...	0.3650	0.740	14



Getting final Playlist

```
playlistDF_test[["artist_name","track_name"]][:20]
```

	artist_name	track_name
413	The Killers	Mr. Brightside
1234	Rihanna	We Found Love
1363	American Authors	Best Day Of My Life
1579	Clean Bandit	Rather Be (feat. Jess Glynne)
1732	Sia	Chandelier
3986	Hozier	Jackie And Wilson
3999	Aloe Blacc	I Need a Dollar
4002	Aloe Blacc	Wake Me Up - Acoustic
4007	John Legend	All of Me
4027	Pharrell Williams	Happy - From "Despicable Me 2"
4043	Justin Timberlake	Mirrors
4055	Bruno Mars	Locked Out Of Heaven
4067	Mark Ronson	Uptown Funk
4093	WALK THE MOON	Shut Up and Dance
4120	Matisyahu	Live Like A Warrior
4123	Jess Glynne	Hold My Hand

7 Drawbacks

One drawback of our content-based filtering approach is that if we were to use a different model to cluster data, the results predicted by the model may differ. This could be due to a variety of factors such as limitations of the model, dataset size, or the absence of hyperparameter tuning. In addition, the lack of a suitable metric to train the model and evaluate its success can be problematic, which underscores the importance of having a quantitative way to measure success to improve the model.

8 Conclusion

Furthermore, the advantages of large tech companies in the domain of recommendation systems are highlighted by our work. It can be difficult to assess the success of a recommendation system in an open-source context without deploying it and gathering user feedback. Metrics such as the number of users who add recommended songs to their playlists can be useful for conducting A/B testing and determining which model or set of parameters performs the best, allowing us to update the model accordingly.

9 Github link

https://github.com/ASP22SCM22S/MachineLearningCS584