Building a music recommendation system (Using Spotify dataset)

Mounika Gampa

Ansh Shrivastava

1.Project Requirements:

1. Windows or Mac OS

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

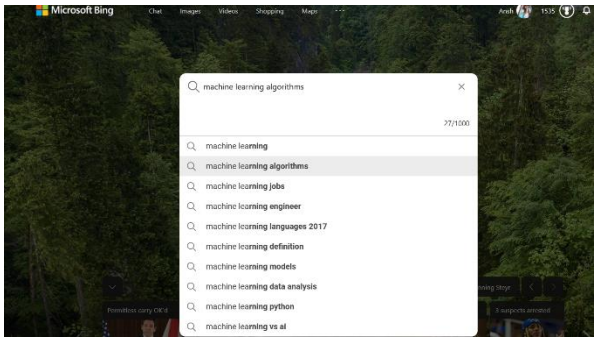
systems that make suggestions based on a person's personal viewing history or the combined viewing histories of themselves and their friends. In the case of song prediction, recommendation algorithms aim to anticipate or suggest songs that a user may like based on their data or the data of all users 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:

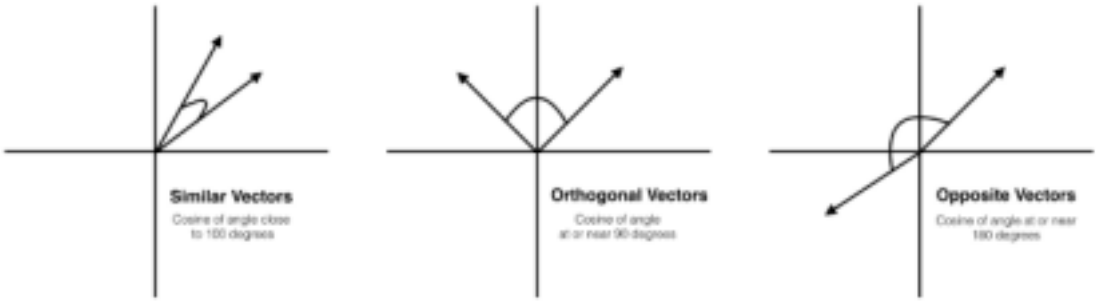
1.This stage involves compiling a playlist of songs into a vector that can be compared to every other song in the dataset to identify similarities. To start, we can classify songs into those that are in the playlist and those that aren’t. Since we don’t want to endorse any of the songs in the playlist, 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.Progress Work

6.1.The first step is to remove the duplicates of the data

```
# Drop song duplicates
def drop_duplicates(df):
    '''
    Drop duplicate songs
    '''
    df['artists_song'] = df.apply(lambda row: row['artist_name']+row['track_name'],axis = 
    return df.drop_duplicates('artists_song')

songDF = drop_duplicates(playlistDF)
```

6.2. Now take certain columns from the data.

```
# Select useful columns
def select_cols(df):
        '''
        Select useful columns
        '''
        return df[['artist_name','id','track_name','danceability', 'energy', 'key', 'loudne
        'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', '
songDF = select_cols(songDF)
songDF.head()
```

6.3.The genres column is made into the list

```
def genre_preprocess(df):
    '''
    Preprocess the genre data
    '''
    df['genres_list'] = df['genres'].apply(lambda x: x.split(" "))
    return df
songDF = genre_preprocess(songDF)
songDF['genres_list'].head()
```

6.4. Now we used the one hot encoding to transform the categorized data into machine readable features.

```
def ohe_prep(df, column, new_name):
    '''
    Create One Hot Encoded features of a specific column
    ---
    Input:
    df (pandas dataframe): Spotify Dataframe
    column (str): Column to be processed
    new_name (str): new column name to be used

    Output:
    tf_df: One-hot encoded features
    '''

    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
```

5

6.5. We implemented tf-idf vectorizer as shown below

```
# TF-IDF implementation
tfidf = TfidfVectorizer()
tfidf_matrix =  tfidf.fit_transform(songDF['genres_list'].apply(lambda x: " ".join(x)))
genre_df = pd.DataFrame(tfidf_matrix.toarray())
genre_df.columns = ['genre' + "|" + i for i in tfidf.get_feature_names()]
genre_df.drop(columns='genre|unknown') # Drop unknown genre
genre_df.reset_index(drop = True, inplace=True)
genre_df.iloc[0]
```

7. Next Step

The next step in our process is to prepare the data for our algorithm, which will enable us to predict recommended songs from our dataset. To do this, we will use the Tf-IDF vectorizer and cosine similarity metrics. The Tf-IDF vectorizer is a tool that will convert our textual data (in this case, song titles and artist names) into numerical feature vectors that can be used by our algorithm. The cosine similarity metric is a measure of similarity between two vectors, which will be used to compare the vector of the playlist we have compiled against the vectors of all other songs in our dataset.

Once we have prepared the data, we will load it into our algorithm and train the model. Our hope is that the filtered data will fit correctly into our model, and that our algorithm will produce

accurate results. This will allow us to generate recommendations for new songs to add to the playlist based on their similarity to songs already on the playlist. By following this process, we can continually refine and improve our playlist recommendation system.