Spotify Recommender System

November 25, 2023

1 Spotify Recommender System

A Music Recommendation System is an application of Data Science that aims to assist users in discovering new and relevant musical content based on their preferences and listening behaviour. Personalized music recommendations have become an essential tool in the digital music landscape, enabling music streaming platforms like Spotify and Apple Music to offer personalized and engaging experiences to their users.

2 How Does a Music Recommendation System Work?

Music Recommendation Systems operate through intricate algorithms that analyze vast amounts of data about users' musical interactions, such as their listening history, liked tracks, skipped songs, and even explicit user preferences conveyed through ratings or feedback. These data points are instrumental in constructing comprehensive user profiles, delineating individual tastes and preferences.

In the initial phase, the system employs various data preprocessing techniques to cleanse and organize the information efficiently. Subsequently, the system uses recommendation algorithms, such as collaborative filtering, content-based filtering, and hybrid approaches, to generate music recommendations.

As users continually interact with the system, it accumulates additional data, refining and updating their profiles in real time. Consequently, the recommendations become increasingly precise and aligned with the user's evolving musical preferences.

[1]:
```python
import requests
import base64

CLIENT_ID = 'abf01fec947441699d9b3bf5a0532a3d'
CLIENT_SECRET = '5b79853ff77a4621aca6cd7d789e54f8'

# Base64 encode the client ID and client secret
client_credentials = f"{CLIENT_ID}:{CLIENT_SECRET}"
client_credentials_base64 = base64.b64encode(client_credentials.encode())

# Request the access token
token_url = 'https://accounts.spotify.com/api/token'
headers = {
```

```
    'Authorization': f'Basic {client_credentials_base64.decode()}'
}
data = {
    'grant_type': 'client_credentials'
}
response = requests.post(token_url, data=data, headers=headers)

if response.status_code == 200:
    access_token = response.json()['access_token']
    print("Access token obtained successfully.")
else:
    print("Error obtaining access token.")
    exit()
```

```
Access token obtained successfully.
```

In the above code, The CLIENT_ID and CLIENT_SECRET variables hold my credentials (you need to add your credentials in these variables) that uniquely identify the application making requests to the Spotify API. These credentials are obtained when a developer registers their application with Spotify's developer dashboard. The Client ID identifies the application, while the Client Secret is a confidential key used for authentication.

The client ID and secret are combined in the client_credentials variable, separated by a colon (:). Then, this string is encoded using Base64 encoding to create a secure representation of the credentials. We then proceed to request an access token from the Spotify API.

It sends a POST request to the token_url (https://accounts.spotify.com/api/token) with the client credentials in the Authorization header, which is required for client authentication. The grant_type parameter is set to 'client_credentials' to indicate that the application is requesting an access token for the client credentials flow.

With the access token, the application can now make authorized requests to retrieve music data, such as tracks, albums, artists, and user information, which is fundamental for building a music recommendation system using the Spotify API and Python.

Now, I'll write a function to get music data from any playlist on Spotify. For this task, you need to install the Spotipy library, which is a Python library providing access to Spotify's web API. Here's how to install it on your system by writing the command mentioned below in your command prompt or terminal:

[7]:
```
import pandas as pd
import spotipy
from spotipy.oauth2 import SpotifyOAuth
def get_trending_playlist_data(playlist_id, access_token):
    # Set up Spotipy with the access token
    sp = spotipy.Spotify(auth=access_token)

    # Get the tracks from the playlist
    playlist_tracks = sp.playlist_tracks(playlist_id, fields='items(track(id,␣
 ↪name, artists, album(id, name)))')
```

```
    # Extract relevant information and store in a list of dictionaries
    music_data = []
    for track_info in playlist_tracks['items']:
        track = track_info['track']
        track_name = track['name']
        artists = ', '.join([artist['name'] for artist in track['artists']])
        album_name = track['album']['name']
        album_id = track['album']['id']
        track_id = track['id']

        # Get audio features for the track
        audio_features = sp.audio_features(track_id)[0] if track_id != 'Not available' else None

        # Get release date of the album
        try:
            album_info = sp.album(album_id) if album_id != 'Not available' else None
            release_date = album_info['release_date'] if album_info else None
        except:
            release_date = None

        # Get popularity of the track
        try:
            track_info = sp.track(track_id) if track_id != 'Not available' else None
            popularity = track_info['popularity'] if track_info else None
        except:
            popularity = None

        # Add additional track information to the track data
        track_data = {
            'Track Name': track_name,
            'Artists': artists,
            'Album Name': album_name,
            'Album ID': album_id,
            'Track ID': track_id,
            'Popularity': popularity,
            'Release Date': release_date,
            'Duration (ms)': audio_features['duration_ms'] if audio_features else None,
            'Explicit': track_info.get('explicit', None),
            'External URLs': track_info.get('external_urls', {}).get('spotify', None),
            'Danceability': audio_features['danceability'] if audio_features else None,
```

```
            'Energy': audio_features['energy'] if audio_features else None,
            'Key': audio_features['key'] if audio_features else None,
            'Loudness': audio_features['loudness'] if audio_features else None,
            'Mode': audio_features['mode'] if audio_features else None,
            'Speechiness': audio_features['speechiness'] if audio_features else␣
↪None,
            'Acousticness': audio_features['acousticness'] if audio_features␣
↪else None,
            'Instrumentalness': audio_features['instrumentalness'] if␣
↪audio_features else None,
            'Liveness': audio_features['liveness'] if audio_features else None,
            'Valence': audio_features['valence'] if audio_features else None,
            'Tempo': audio_features['tempo'] if audio_features else None,
            # Add more attributes as needed
        }

        music_data.append(track_data)

    # Create a pandas DataFrame from the list of dictionaries
    df = pd.DataFrame(music_data)

    return df
```

The function begins by initializing the Spotipy client with the provided access_token, which serves as the authentication token to interact with the Spotify Web API. The access_token allows the function to make authorized requests to access Spotify's resources. The function then uses the Spotipy client to fetch information about the tracks in the specified playlist (identified by its playlist_id). The sp.playlist_tracks method retrieves the playlist tracks. The fields parameter is used to specify the specific track information that is required, such as track ID, name, artists, album ID, and album name.

The function then extracts relevant information from the retrieved playlist tracks and stores it in a list of dictionaries called music_data. For each track in the playlist, the function extracts data such as track name, artists (combined into a single string), album name, album ID, track ID, and popularity. The function uses the sp.audio_features method to fetch audio features for each track in the playlist. These audio features include attributes like danceability, energy, key, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo, etc. These audio features provide insights into the characteristics of each track.

The extracted information for all tracks is stored in the music_data list. The function then creates a DataFrame from the music_data list. The DataFrame organizes the music data in a tabular format, making it easier to analyze and work with the collected information.

Now, here's how we can use the function to collect music data from any playlist on Spotify:

[8]:
```
playlist_id = '37i9dQZF1DX76Wlfdnj7AP'

# Call the function to get the music data from the playlist and store it in a␣
↪DataFrame
```

```python
music_df = get_trending_playlist_data(playlist_id, access_token)

# Display the DataFrame
print(music_df)
```

```
                        Track Name                          Artists  \
0                            Prada  cassö, RAYE, D-Block Europe
1                         fukumean                            Gunna
2               IDGAF (feat. Yeat)                      Drake, Yeat
3                  I'm Good (Blue)         David Guetta, Bebe Rexha
4   Vois sur ton chemin - Techno Mix                          BENNETT
..                             …                               …
95                Little Girl Gone                       CHINCHILLA
96                     All By Myself  Alok, Sigala, Ellie Goulding
97       Kernkraft 400 (A Better Day)                       Topic, A7S
98              Red Ruby Da Sleeze                       Nicki Minaj
99            Rainfall (Praise You)                        Tom Santa

                        Album Name                Album ID  \
0                            Prada  5MU0RmBSpoSx0PYBfcobDc
1                  a Gift & a Curse  5qmZefgh78fN3jsyPPlvuw
2                  For All The Dogs  4czd0RdCWP9umpbhFXK2fW
3                  I'm Good (Blue)  7M842DMhYVALrXsw3ty7B3
4   Vois sur ton chemin (Techno Mix)  79Cyc8GRWnLyjdJSMyJ0dB
..                             …                     …
95                Little Girl Gone  7tzZQfNdN5rWCYFcM24byP
96                     All By Myself  31Amnw0gNntYuTltwETnSn
97       Kernkraft 400 (A Better Day)  2NIChqkijGw4r4Dqfmg0A3
98              Red Ruby Da Sleeze  0zCHOD0Z8y0rIP1fw7u1J6
99            Rainfall (Praise You)  4VanY5i4E59Mhz52qznJ95

                  Track ID  Popularity Release Date  Duration (ms)  Explicit  \
0   59NraMJsLaMCVtwXTSia8i          94   2023-08-11         132359      True
1   4rXLjWdF2ZZpXCVTfWcshS          93   2023-06-16         125040      True
2   2YSzYUF3jWqb9YP9VXmpjE          93   2023-10-06         260111      True
3   4uUG5RXr0k84mYEfFvj3cK          91   2022-08-26         175238      True
4   31nfdEooLEq7dn3UMcIeB5          90   2023-08-04         178156     False
..                     …         …          …            …       …
95  56rpEOCBATYItSa4yPksfe          76   2023-09-01         188596      True
96  5Hp4xFihd0E2dmDzxWcBFb          76   2022-10-07         171778     False
97  3kcKl0kQQEPVwxwljbGJ5p          76   2022-06-17         165800     False
98  4ZYAU4A2YBtlNdq0Utc7T2          76   2023-03-03         214445      True
99  1M8t1j3Kv2qp97bdq5q4Vl          76   2022-02-18         166570     False

                                   External URLs  …  Energy  Key  \
0   https://open.spotify.com/track/59NraMJsLaMCVtw…  …   0.717    8
1   https://open.spotify.com/track/4rXLjWdF2ZZpXCV…  …   0.622    1
2   https://open.spotify.com/track/2YSzYUF3jWqb9YP…  …   0.670    8
```

```
3   https://open.spotify.com/track/4uUG5RXrOk84mYE…  …   0.965    7
4   https://open.spotify.com/track/31nfdEooLEq7dn3…  …   0.824    2
..                                              …  …      …  …
95  https://open.spotify.com/track/56rpEOCBATYItSa…  …   0.683    1
96  https://open.spotify.com/track/5Hp4xFihdOE2dmD…  …   0.848    0
97  https://open.spotify.com/track/3kcKlOkQQEPVwxw…  …   0.727   11
98  https://open.spotify.com/track/4ZYAU4A2YBtlNdq…  …   0.733    1
99  https://open.spotify.com/track/1M8t1j3Kv2qp97b…  …   0.862    5

    Loudness  Mode  Speechiness  Acousticness  Instrumentalness  Liveness  \
0     -5.804     1       0.0375       0.00100          0.000002    0.1130
1     -6.747     0       0.0903       0.11900          0.000000    0.2850
2     -8.399     1       0.2710       0.04640          0.000089    0.2050
3     -3.673     0       0.0343       0.00383          0.000007    0.3710
4     -3.394     0       0.0470       0.09080          0.071100    0.1190
..       …    …          …            …                …         …
95    -6.342     0       0.2710       0.19000          0.000000    0.0819
96    -4.338     0       0.0346       0.09320          0.000008    0.2410
97    -5.570     0       0.0562       0.18400          0.000020    0.3090
98    -6.181     1       0.2560       0.11500          0.000000    0.1110
99    -5.464     0       0.0606       0.14000          0.009200    0.2520

    Valence    Tempo
0     0.422  141.904
1     0.220  130.001
2     0.138  136.952
3     0.304  128.040
4     0.371  137.959
..      …       …
95    0.534  159.998
96    0.773  123.041
97    0.400  125.975
98    0.292   98.355
99    0.509  128.039

[100 rows x 21 columns]
```

In this code snippet, we used a playlist ID: "37i9dQZF1DX76Wlfdnj7AP". The code then calls the get_trending_playlist_data function to extract music data from the specified playlist using the provided access_token. The collected music data is stored in a DataFrame named music_df. Finally, the code prints the DataFrame to display the extracted music data.

You can also add your playlist id here. If your playlist link is (https://open.spotify.com/playlist/37i9dQZF1DX76Wlfdnj7AP), the playlist ID is "37i9dQZF1DX76Wlfdnj7AP", which is what you would replace with my playlist id within the above code snippet.

Now let's check if the data has any null values or not:

[9]:
```python
print(music_df.isnull().sum())
```

```
Track Name          0
Artists             0
Album Name          0
Album ID            0
Track ID            0
Popularity          0
Release Date        0
Duration (ms)       0
Explicit            0
External URLs       0
Danceability        0
Energy              0
Key                 0
Loudness            0
Mode                0
Speechiness         0
Acousticness        0
Instrumentalness    0
Liveness            0
Valence             0
Tempo               0
dtype: int64
```

Now, let's move further to building a music recommendation system using Python. Let's import the necessary Python libraries now:

[10]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from datetime import datetime
from sklearn.metrics.pairwise import cosine_similarity

data = music_df
```

While providing music recommendations to users, it is important to recommend the latest releases. For this, we need to give more weight to the latest releases in the recommendations. Let's write a function to solve this problem:

[11]:
```python
# Function to calculate weighted popularity scores based on release date
def calculate_weighted_popularity(release_date):
    # Convert the release date to datetime object
    release_date = datetime.strptime(release_date, '%Y-%m-%d')

    # Calculate the time span between release date and today's date
    time_span = datetime.now() - release_date
```

```
    # Calculate the weighted popularity score based on time span (e.g., more␣
↪recent releases have higher weight)
    weight = 1 / (time_span.days + 1)
    return weight
```

The above function takes the release date of a music track as input, which is provided in the format 'YYYY-MM-DD'. It then uses the datetime.strptime function from the Python datetime module to convert the release date string to a datetime object. This conversion allows us to perform arithmetic operations with dates. The function then calculates the time span between the release date of the track and the current date (today's date) using datetime.now() – release_date. This results in a timedelta object representing the time difference between the two dates.

The weighted popularity score is computed based on the time span. The formula to calculate the weight is 1 / (time_span.days + 1). The time_span.days attribute of the timedelta object gives the number of days in the time span between the release date and today. Adding 1 to the number of days ensures that the weight is never zero, even for very recent releases, as this would lead to a division by zero error.

The idea behind this formula is that the weight decreases as the time span between the release date and today increases. More recent releases will have a higher weight, while older releases will have a lower weight. As a result, when combining this weighted popularity score with other factors in a recommendation system, recent tracks will have a more significant impact on the final recommendations, reflecting users' potential interest in newer music.

Now let's normalize the music features before moving forward:

[12]:
```
# Normalize the music features using Min-Max scaling
scaler = MinMaxScaler()
music_features = music_df[['Danceability', 'Energy', 'Key',
                           'Loudness', 'Mode', 'Speechiness', 'Acousticness',
                           'Instrumentalness', 'Liveness', 'Valence', 'Tempo']].
↪values
music_features_scaled = scaler.fit_transform(music_features)
```

We will create a hybrid recommendation system for music recommendations. The first approach will be based on recommending music based on music audio features, and the second approach will be based on recommending music based on weighted popularity.

Here's how to generate music recommendations based on the music audio features:

[13]:
```
# a function to get content-based recommendations based on music features
def content_based_recommendations(input_song_name, num_recommendations=5):
    if input_song_name not in music_df['Track Name'].values:
        print(f"'{input_song_name}' not found in the dataset. Please enter a␣
↪valid song name.")
        return

    # Get the index of the input song in the music DataFrame
    input_song_index = music_df[music_df['Track Name'] == input_song_name].
↪index[0]
```

```
    # Calculate the similarity scores based on music features (cosine␣
↪similarity)
    similarity_scores =␣
↪cosine_similarity([music_features_scaled[input_song_index]],␣
↪music_features_scaled)

    # Get the indices of the most similar songs
    similar_song_indices = similarity_scores.argsort()[0][::-1][1:
↪num_recommendations + 1]

    # Get the names of the most similar songs based on content-based filtering
    content_based_recommendations = music_df.iloc[similar_song_indices][['Track␣
↪Name', 'Artists', 'Album Name', 'Release Date', 'Popularity']]

    return content_based_recommendations
```

The above function takes input_song_name as the input, which represents the name of the song for which recommendations are to be generated. The function checks if the input_song_name exists in the music_df DataFrame, which presumably contains the music data with features like 'Track Name', 'Artists', 'Album Name', 'Release Date', and 'Popularity'. If the input song name is found in the music_df DataFrame, the function retrieves the index of the input song in the DataFrame. This index will be used to compare the audio features of the input song with other songs in the dataset.

The function calculates the similarity scores between the audio features of the input song and all other songs in the dataset. It uses cosine similarity, a common measure used in content-based filtering. The cosine_similarity function from scikit-learn is employed to compute these similarity scores.

The function identifies the num_recommendations most similar songs to the input song based on their audio features. It does this by sorting the similarity scores in descending order and selecting the top num_recommendations songs. The input song itself is excluded from the recommendations (hence the [1:num_recommendations + 1] slicing). The function then extracts the details (such as track name, artists, album name, release date, and popularity) of the most similar songs from the music_df DataFrame using the indices of the most similar songs.

Now here's the function to generate music recommendations based on weighted popularity and combine it with the recommendations of the content-based filtering method using the hybrid approach:

[14]:
```
# a function to get hybrid recommendations based on weighted popularity
def hybrid_recommendations(input_song_name, num_recommendations=5, alpha=0.5):
    if input_song_name not in music_df['Track Name'].values:
        print(f"'{input_song_name}' not found in the dataset. Please enter a␣
↪valid song name.")
        return

    # Get content-based recommendations
```

```
    content_based_rec = content_based_recommendations(input_song_name,␣
↪num_recommendations)

    # Get the popularity score of the input song
    popularity_score = music_df.loc[music_df['Track Name'] == input_song_name,␣
↪'Popularity'].values[0]

    # Calculate the weighted popularity score
    weighted_popularity_score = popularity_score *␣
↪calculate_weighted_popularity(music_df.loc[music_df['Track Name'] ==␣
↪input_song_name, 'Release Date'].values[0])

    # Combine content-based and popularity-based recommendations based on␣
↪weighted popularity
    hybrid_recommendations = content_based_rec
    hybrid_recommendations = hybrid_recommendations.append({
        'Track Name': input_song_name,
        'Artists': music_df.loc[music_df['Track Name'] == input_song_name,␣
↪'Artists'].values[0],
        'Album Name': music_df.loc[music_df['Track Name'] == input_song_name,␣
↪'Album Name'].values[0],
        'Release Date': music_df.loc[music_df['Track Name'] == input_song_name,␣
↪'Release Date'].values[0],
        'Popularity': weighted_popularity_score
    }, ignore_index=True)

    # Sort the hybrid recommendations based on weighted popularity score
    hybrid_recommendations = hybrid_recommendations.
↪sort_values(by='Popularity', ascending=False)

    # Remove the input song from the recommendations
    hybrid_recommendations =␣
↪hybrid_recommendations[hybrid_recommendations['Track Name'] !=␣
↪input_song_name]


    return hybrid_recommendations
```

The hybrid approach aims to provide more personalized and relevant recommendations by considering both the content similarity of songs and their weighted popularity. The function takes input_song_name as the input, representing the name of the song for which recommendations are to be generated. The function first calls the content_based_recommendations function to get content-based recommendations for the input song. The num_recommendations parameter determines the number of content-based recommendations to be retrieved.

The function calculates the popularity score of the input song by retrieving the popularity value from the music_df DataFrame. It also calculates the weighted popularity score using the calculate_weighted_popularity function (previously defined) based on the release date of the input song. The alpha parameter controls the relative importance of content-based and popularity-based recommendations.

The content-based recommendations obtained earlier are stored in the content_based_rec DataFrame. The function combines the content-based recommendations with the input song's information (track name, artists, album name, release date, and popularity) and its weighted popularity score. This step creates a DataFrame named hybrid_recommendations that includes both the content-based recommendations and the input song's data.

The hybrid_recommendations DataFrame is then sorted in descending order based on the weighted popularity score. This step ensures that the most popular and relevant songs appear at the top of the recommendations. The input song is then removed from the recommendations to avoid suggesting the same song as part of the recommendations.

Now here's how we can test the final function to generate music recommendations:

[15]:
```
input_song_name = "I'm Good (Blue)"
recommendations = hybrid_recommendations(input_song_name, num_recommendations=5)
print(f"Hybrid recommended songs for '{input_song_name}':")
print(recommendations)
```

```
Hybrid recommended songs for 'I'm Good (Blue)':
                   Track Name                                     Artists  \
3  FE!N (feat. Playboi Carti)                  Travis Scott, Playboi Carti
4                Call It Love                      Felix Jaehn, Ray Dalton
1                       REACT  Switch Disco, Ella Henderson, Robert Miles
0                        BOTH                       Tiësto, 21 Savage, BIA
2              Where You Are                          John Summit, Hayla

      Album Name Release Date  Popularity
3         UTOPIA   2023-07-28        89.0
4   Call It Love   2022-09-16        80.0
1          REACT   2023-01-13        79.0
0           BOTH   2023-08-29        78.0
2  Where You Are   2023-03-03        77.0

C:\Users\KEERTHAN\AppData\Local\Temp\ipykernel_10568\209674984.py:18:
FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  hybrid_recommendations = hybrid_recommendations.append({
```

So this is how you can create a Music Recommendation System using Spotify API and Python.

3 Summary

So, I hope you liked this article on building a Music Recommendation System using the Spotify API and Python. A Music Recommendation System is an application of Data Science that aims to assist users in discovering new and relevant musical content based on their preferences and listening behaviour.