Music Recommender

A Web Application Using Spotify API

Introduction to the Problem

In today's rapidly evolving technology landscape, recommendation-based applications play a crucial role in personalizing user experiences. An increasing number of companies utilize recommendation algorithms to deliver personalized content, products, and services. One popular example is music recommendation systems like Spotify, which allow users to discover new music based on their previous preferences. This project's goal is to create a simple recommendation application that uses the Spotify API to generate personalized music playlists. This project aims to explore and understand the mechanisms of recommendation systems and provide a functional tool that can be used in various music application contexts.

Recommendation applications are becoming more advanced, using various techniques such as content filtering, collaborative filtering, and hybrid models that combine different approaches to increase the accuracy of recommendations. In the context of Spotify, recommendation algorithms analyze user data such as played tracks, favorite playlists, and social interactions to provide personalized suggestions.

This project will focus on the following aspects:

Integration with Spotify API: Utilizing key Spotify API functions such as retrieving track data, creating and modifying playlists, and managing user authorization.

Recommendation Algorithms: Implementing basic recommendation algorithms such as collaborative filtering to generate personalized playlists.

Data Analysis: Collecting and analyzing user data to better understand their music preferences and tailor recommendations.

Testing and Evaluation: Conducting functional tests of the application to assess its effectiveness and accuracy.

This project responds to the growing demand for tools that can automatically, efficiently, and accurately deliver valuable recommendations to users. This document will present the key aspects of the issues related to music recommendations, an elaboration of the problem, the applied methodology, the results of the application's operation, and discussion and future development possibilities.

Expanding the Problem

Music Recommendation Issues

Music recommendations are a key element for streaming platforms like Spotify, which aim to increase user engagement by delivering personalized content. The recommendation problem involves predicting which tracks a user might like based on the analysis of their

previous actions and preferences. This is challenging for several reasons:
Musical Diversity: Music is a very diverse form of art with many genres, styles, and artists. A recommendation system must account for this diversity to provide relevant and varied suggestions.
Scalability: Platforms like Spotify support millions of users and have huge databases of music tracks. Recommendation algorithms must be scalable to efficiently process and analyze this data in real-time.
Personalization: Each user has unique music preferences that can change over time. The recommendation system must dynamically adjust its suggestions to reflect the user's current tastes.
Project Specifics
In the context of this project, it is essential to note that the recommender does not rely on the user's listening history but on artists, tracks, or genres provided by the user. This is important as it introduces additional challenges and opportunities in designing the recommendation algorithm.

Recommendation Methods

There are several main approaches to the music recommendation problem:
Content-based Filtering: This approach involves analyzing the features of music tracks (such as genre, tempo, mood) and recommending similar tracks based on user preferences. In our project's context, content filtering will be crucial as the user provides the preferred features such as specific artists, tracks, or music genres.
Collaborative Filtering: In this approach, recommendations are generated based on the preferences and behaviors of other users with similar tastes. Although we focus mainly on content filtering in our project, we may consider elements of collaborative filtering as a complement.
Hybrid Models: These models combine different approaches such as content-based and collaborative filtering to increase the accuracy and diversity of recommendations. Hybrid models can leverage the strengths of both methods while minimizing their weaknesses.

Technical Challenges

Implementing an effective recommendation system faces several technical challenges:
Data Collection and Processing: The system must efficiently collect data about users and tracks and then process it to be useful for recommendation algorithms. In our case, the correct processing of input data provided by the user will be crucial.
Algorithm Optimization: Recommendation algorithms must be optimized for performance and accuracy, requiring advanced knowledge in machine learning and data analysis.
User Interface: The ultimate success of the recommendation system depends on how well the recommendations are presented to users. The interface must be intuitive and allow easy

access to recommended tracks.

Project Significance

This project is an interesting case for those interested in modern recommendation technologies and their practical applications. While it is not of great significance to a wide audience, it can be a valuable educational element showing how recommendation systems work in the context of music.
The goal of this project is not only to create a functional tool but also to gain knowledge and experience in working with recommendation algorithms and the Spotify API. Through this project, we will understand how to create music recommendations that are scalable, personalized, and relevant. For those interested, this project can be an inspiration for further exploration of the topic and development of their own recommendation solutions.

Methodology

Integration with Spotify API

The project was implemented using the Spotify API, which provides access to various music data and functions related to playback and music management. Key integration steps included:
Accessing the Spotify API: Configuring access to the Spotify API by creating an application in the Spotify Developer Dashboard and obtaining the necessary API keys (CLIENT_ID, CLIENT_SECRET).
User Authorization: Implementing the user authorization process using the OAuth protocol, allowing secure permissions to manage user playlists and read their data.
Retrieving Music Data: Using the available Spotify API endpoints to retrieve data about artists, tracks, and music genres to generate recommendations.

Spotipy Library

The application uses the Spotipy library, a simple and effective wrapper for the Spotify API. It allows easy API queries and authorization management. Key elements of using the Spotipy library include:
Configuration and Authorization: The script simple_recommendation_app.py contains the configuration of the connection to the Spotify API and the implementation of the user authorization process using OAuth.
```
import spotipy
from spotipy.oauth2 import SpotifyOAuth
```

```
sp = spotipy.Spotify(auth_manager=SpotifyOAuth(
   client_id='c97059e9f08e41159b29064147e82ff5'
   client_secret='bed718ec384e4b77bf1b69b62d4287cc'
   redirect_uri='http://localhost:8000/callback'
   scope='playlist-modify-public playlist-modify-private'
))
```

Retrieving User Preferences: The script allows the user to enter preferences, which are then processed to search for appropriate data in the Spotify API.

```
def get_recommendations(preferences):
   results = sp.search(q=preferences, type='track,artist')
   return results
```

Generating Recommendations and Creating Playlists: Based on search results, recommendations are generated, which are then saved as a new playlist on the user's account.

```
def create_playlist(user_id, playlist_name, track_ids):
   playlist = sp.user_playlist_create(user_id, playlist_name)
   sp.user_playlist_add_tracks(user_id, playlist['id'], track_ids)
```

Running the Application on Flask

The application was built using the Flask framework, which allows easy creation of web applications in Python. Flask is lightweight and flexible, making it an ideal choice for this project. Key elements of the application include:
Flask Configuration: The script simple_recommendation_app.py contains the configuration and launching of the Flask server.

```
from flask import Flask, request, redirect, url_for, session
app = Flask(__name__)
app.secret_key = 'some_random_secret_key'

@app.route('/')
def index():
   return 'Hello, this is the recommendation app!'
```

User Authorization: The user authorization process is integrated with Flask, allowing secure login and session management.

```
@app.route('/login')
```

```
def login():
    sp_oauth = SpotifyOAuth()
    auth_url = sp_oauth.get_authorize_url()
    return redirect(auth_url)

@app.route('/callback')
def callback():
    sp_oauth = SpotifyOAuth()
    session.clear()
    code = request.args.get('code')
    token_info = sp_oauth.get_access_token(code)
    session['token_info'] = token_info
    return redirect(url_for('index'))
```
User Interaction: Flask handles the user interface, allowing the input of music preferences and displaying recommendation results.
```
@app.route('/recommend', methods=['POST'])
def recommend():
    preferences = request.form['preferences']
    recommendations = get_recommendations(preferences)
    return render_template('recommendations.html', recommendations=recommendations)
```

Data Analysis

To ensure the relevance of recommendations, the project involves analyzing music data for their characteristics such as genre, tempo, mood, popularity, etc. This allows for more precise matching of recommendations to user preferences.

Testing and Evaluation

The project was tested for functionality and recommendation effectiveness. Tests included:
Verifying the correctness of authorization and connection to the Spotify API.
Checking the relevance of generated recommendations based on different sets of user preferences.
Assessing the intuitiveness and usability of the user interface.
This project is an interesting case for those interested in modern recommendation technologies and serves as a basis for further research and development in this field.

Results

Creating a Playlist
As a result of the recommendation application, a new playlist with a predefined title is created on the user's account. The playlist contains 10 tracks selected based on user preferences. The playlist creation process is as follows:
User Data Input: The user inputs music preferences by providing a link containing the ID of an artist, genre, or track.
Generating Recommendations: The application processes the provided preferences and uses the Spotify API to find tracks matching the specified criteria. Based on the search results, a list of recommended tracks is generated.
Creating the Playlist: A new playlist with a predefined title is created on the user's account, and the recommended tracks are added to it. Each playlist contains 10 tracks.
How the Application Works
The application is designed as a simple web tool that allows the user to generate personalized playlists based on provided preferences. The application uses the Flask framework and the Spotipy library, which allows easy integration with the Spotify API.

Instructions to Run the Application (How To)

Step 1: Environment Setup
Install the required libraries: Make sure you have the Flask and Spotipy libraries installed. You can install them using pip:
```
pip install Flask spotipy
```
Create a Spotify application: Log in to the Spotify Developer Dashboard and create a new application. Note your CLIENT_ID and CLIENT_SECRET.
Configure the configuration file: Create a config.yaml file with the following content, replacing the values with your data:
```
spotify:
  CLIENT_ID: 'your_client_id'
  CLIENT_SECRET: 'your_client_secret'
  USER: 'your_spotify_user_id'
  REDIRECT_URL: 'http://localhost:8000/callback'
  SCOPE: 'playlist-modify-public playlist-modify-private'
  PLAYLIST: 'My Recommended Playlist'
```

Step 2: Code Implementation
Application Script: Create a file simple_recommendation_app.py and paste the following code into it:

```
import spotipy
from spotipy.oauth2 import SpotifyOAuth
from flask import Flask, request, redirect, url_for, session, render_template

app = Flask(__name__)
app.secret_key = 'some_random_secret_key'

sp = spotipy.Spotify(auth_manager=SpotifyOAuth(
    client_id='your_client_id',
    client_secret='your_client_secret',
    redirect_uri='http://localhost:8000/callback',
    scope='playlist-modify-public playlist-modify-private'
))

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/login')
def login():
    sp_oauth = SpotifyOAuth()
    auth_url = sp_oauth.get_authorize_url()
    return redirect(auth_url)

@app.route('/callback')
def callback():
    sp_oauth = SpotifyOAuth()
    session.clear()
    code = request.args.get('code')
    token_info = sp_oauth.get_access_token(code)
    session['token_info'] = token_info
    return redirect(url_for('index'))

def get_recommendations(preferences):
    results = sp.search(q=preferences, type='track,artist')
    tracks = results['tracks']['items']
    track_ids = [track['id'] for track in tracks[:10]]
    return track_ids

def create_playlist(user_id, playlist_name, track_ids):
    playlist = sp.user_playlist_create(user_id, playlist_name)
    sp.user_playlist_add_tracks(user_id, playlist['id'], track_ids)
```

```
@app.route('/recommend/<preference>')
def recommend(preference):
    track_ids = get_recommendations(preference)
    user_id = session['token_info']['id']
    create_playlist(user_id, 'My Recommended Playlist', track_ids)
    return 'Playlist created successfully!'

if __name__ == '__main__':
    app.run(debug=True)
```

HTML Template: Create a folder named templates and within it a file index.html with the following content:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Recommendation App</title>
</head>
<body>
    <h1>Welcome to the Recommendation App</h1>
    <form action="/login" method="get">
        <button type="submit">Login to Spotify</button>
    </form>
</body>
</html>
```

Step 3: Running the Application

Run the Flask server: In the terminal, navigate to the directory with the simple_recommendation_app.py file and run the application.

Open the browser: Go to http://localhost:8000 in your web browser to access the application.

Log in to Spotify: Click the login link to authorize the application with your Spotify account.

Enter Preferences: After logging in, the user should enter a link in the format http://localhost:8000/recommend/<artist_id_genre_track_id>, where <artist_id_genre_track_id> is the ID of the artist, genre, or track.

Creating the Playlist: The application will process the provided ID, generate recommendations, and create a new playlist with 10 tracks on the user's Spotify account.