Milind Kumar

Milindkumar.choudhary@pace.edu

Abstract

To learn and implement a custom playlist recommendation engine based on Spotify application using Contant-based Filtering mechanism.

Music REcommendation Engine

(Spotify)

**Table of contents**

* Prerequisites
* Key takeaways
* Introduction
  + What is a Recommendation System?
  + Types of Recommendation Systems
  + Content-Based Filtering
  + Collaborative Filtering
  + How to use Spotify Web API to fetch data
* Implementation
  + Feature Engineering
  + Connecting to Spotify Web API
  + Creating Playlist Vector
  + Generating Recommendations
* Few Points to Ponder
* Conclusion

**Prerequisites**

The reader should have basic knowledge of Python libraries like pandas, numpy, scikit-learn, and the basics of vector algebra.

Readers are also requested to go through the documentation of the Spotipy library. *Spotipy* is a lightweight Python library that we will be using to access the Spotify Web API.

**Key takeaways**

By the end of this, you will know the following:

* How Recommendation Systems work.
* Types of Recommendation Systems.
* Mathematics behind Recommendation Systems.
* Implementing a Spotify playlist Recommender Engine from scratch using Python.

**Introduction**

**What is a Recommendation System?**

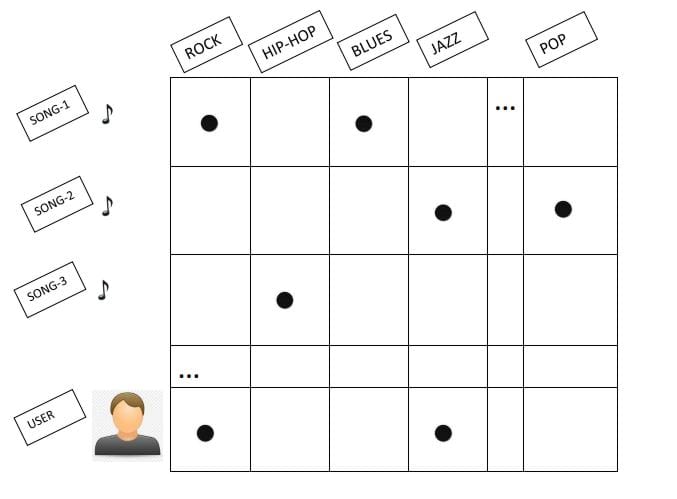
A **Recommendation System** is a tool that predicts the user’s preferences and recommends products or services that are likely to be of interest. These systems rely on user data, which can be broadly classified into two categories: **Characteristic information** and **User-item interactions**.

1. Characteristic information refers to data that defines the profile of a product (such as tags, categories, etc.) or a user (such as preferences, profile, etc.).
2. User-item interactions refer to data that defines the relationship between a user and an item (such as ratings, likes/dislikes, etc.).

Based on these two types of data, we can categorize the algorithms used in a Recommendation System.  
  
**Types of Recommendation Systems**

* Content-Based Filtering
* Collaborative Filtering

**Content-Based Filtering**

**Content-Based Filtering systems** use **characteristic information** to recommend new items or products to a user based on their past actions or explicit feedback. For example, Spotify uses a simple song recommender that relies on a feature matrix. In this matrix, each row represents a song, and each column represents a feature such as genre. The matrix is binary, meaning that a non-zero value represents that the song has that feature present. The user is also represented in the same feature space. Some user-related features are provided explicitly by the user. Figureno.

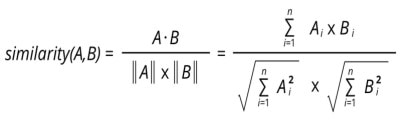
To recommend songs that users may find interesting, we can use a **similarity metric** like **cosine similarity**. The model scores each candidate according to this similarity metric and recommends songs based on the score. The higher the score, the more likely the user is to find that song of interest.

Cosine similarity is a mathematical concept that measures the similarity between two vectors. In the context of a recommendation system, the vectors represent the songs and the user’s preferences. The cosine similarity score ranges from -1 to 1, where 1 indicates that the two vectors are identical, 0 indicates that the two vectors are orthogonal, and -1 indicates that the two vectors are diametrically opposed.

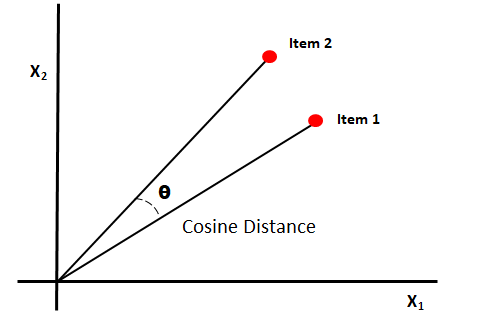
Spotify uses a **Content-Based Filtering system** to recommend new songs to users based on their past actions or explicit feedback. This system uses **characteristic information** to recommend new items or products to a user. For example, Spotify uses a simple song recommender that relies on a feature matrix. In this matrix, each row represents a song, and each column represents a feature such as genre.

**Cosine similarity**

Cosine similarity is the cosine of the angle between two n-dimensional vectors in an n-dimensional feature space. It is the dot product of the two vectors divided by the product of the magnitude of two vectors.



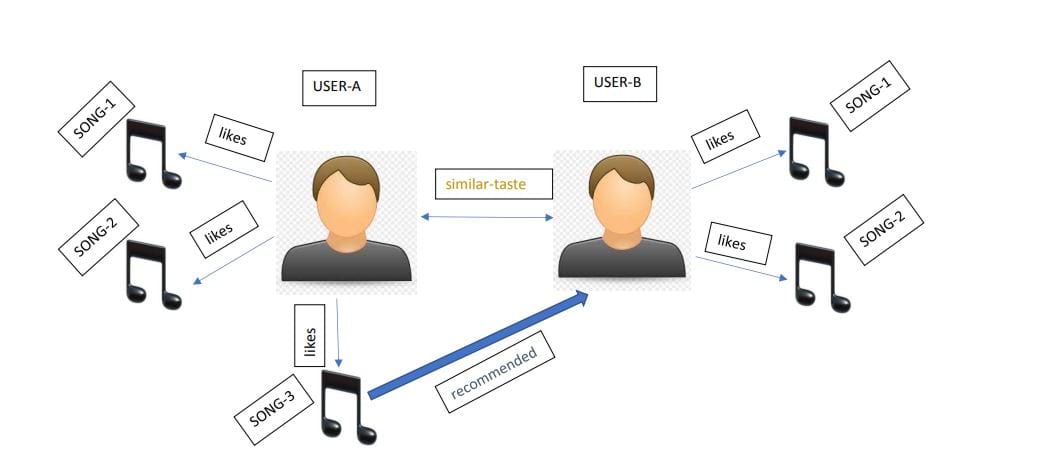
To understand it better, let us take an example of two items, item 1 and item 2, and two features x1 and x2, which define an item. The plot below represents item 1 and item 2 as vectors in a feature space.



The lesser the angle between vectors more the cosine similarity.

**Collaborative filtering**

**Collaborative filtering systems** use **user-item interactions** to generate recommendations. This means that collaborative filtering uses similarities between users and items simultaneously to provide recommendations.



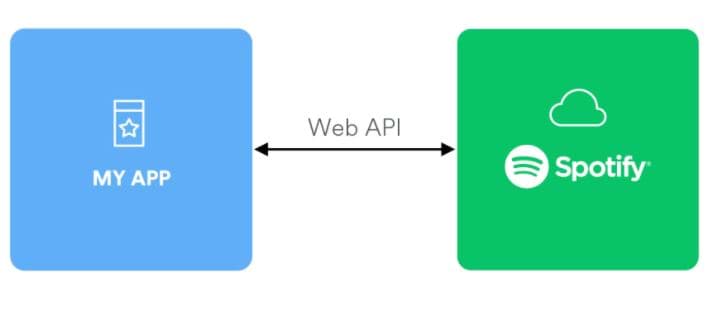
As we can see in the diagram, there are two users, user A, and user B. Both users have similar tastes in music as both of them liked song-1 and song-2, but there is a song-3 which user A likes, but user B never listened to it. The system will recommend song-3 to user A based on these user-item interactions.

As in this, the recommender system that we will implement is based on Content-Based Filtering. Therefore, we are going to limit our discussion of the Collaborative-Filtering system up to its definition.

**How to use Spotify Web API to fetch data**

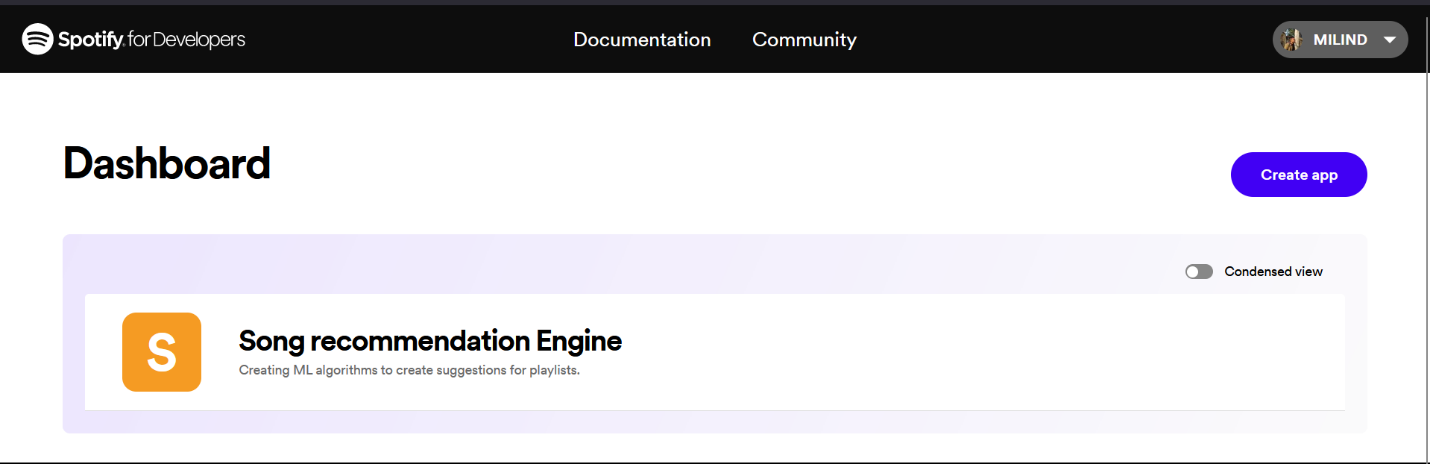
For this, we will implement a custom playlist recommender for which we will require our current Spotify playlist data, which we will generate further recommendations.

To create a custom application and fetch Spotify data, Spotify provides a **web API**. The Spotify Web API endpoints return **JSON metadata** about music artists, albums, and tracks directly from the Spotify Data Catalogue based on simple REST principles.

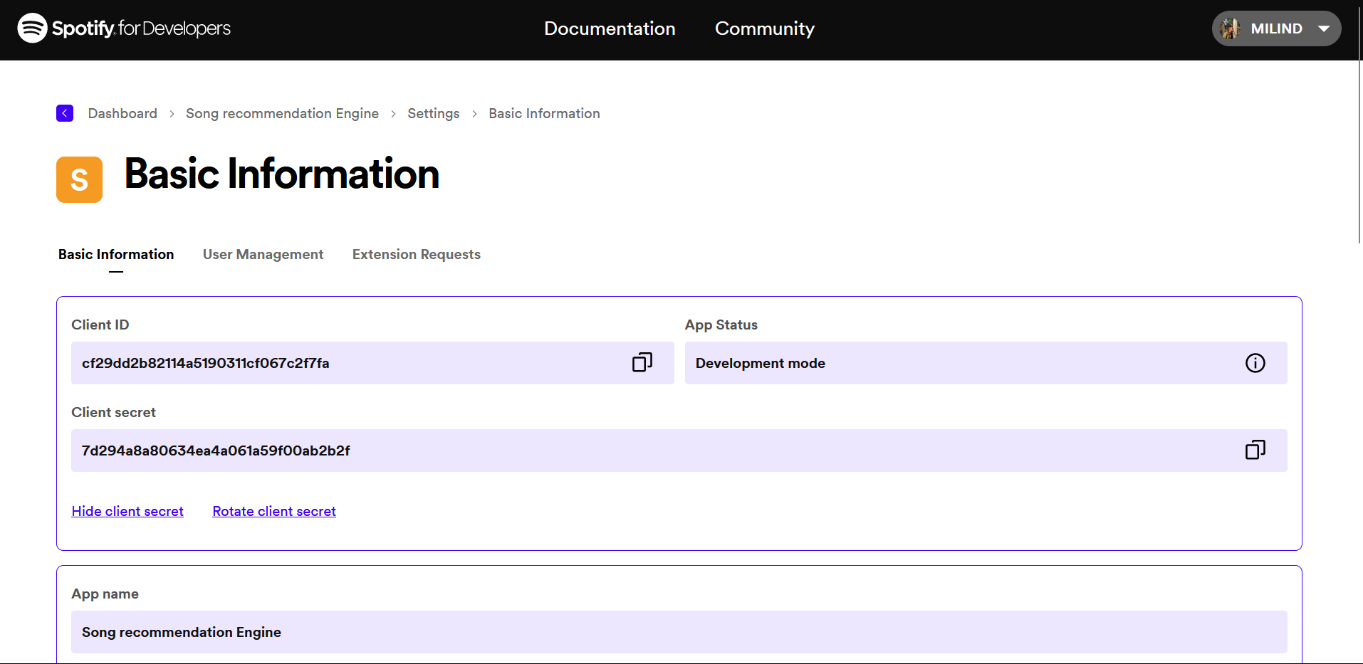


**Simple steps to use Spotify Web API**

Open the Spotify Web API dashboard. Log in using your Spotify account credentials. After logging in, your homepage will look something like this:



Now click on create an application. This will generate a unique client id and password, which will be used further. Let us open the application.



To authenticate the user, you need to add your local host URL in the edit settings option. Now you are all set to implement a Spotify-based playlist recommender system.

For more information on Spotify Web API and its usability, you can refer to this article by Steven Morse, which explains the above process in more detail.

**Implementation**

To access the Spotify Web API, we will use a python-based library known as Spotipy, so let us first install this library.



Now import the following libraries

A screen shot of a computer

Description automatically generated

We will require additional data related to the features of songs present in the Spotify application for this implementation. Using these features, we will determine the similarity between our playlist and the songs not in our playlist. Based on the similarity, we will get a new playlist recommended.

A screen shot of a computer

Description automatically generated

#### Feature engineering

In the dataset, we can observe that multiple columns represent the possible features for a song. Out of these, few features are categorical (columns having discrete values) like genre, key, popularity index, etc. Therefore, the first step would be to convert these categorical features into **one-hot encoding (OHD)** so that our songs can be represented as vectors in a feature space.

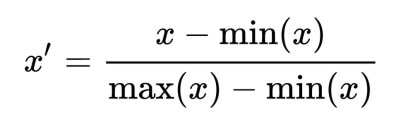
To keep it simple, for now, we are only taking two categorical features into consideration.

A screen shot of a computer

Description automatically generated

To standardize the values of numeric columns in the dataset, we will perform **max-min normalization**. This is the most common normalization approach where the minimum value in the feature column gets transformed to 0, and the maximum value in the feature column gets transformed to 1.

The equation for max-min normalization is given as follows:



A screen shot of a computer

Description automatically generated

We drop the features that are not considered to determine the similarity and the categorical features that are already converted into OHE vectors.

A screen shot of a computer code

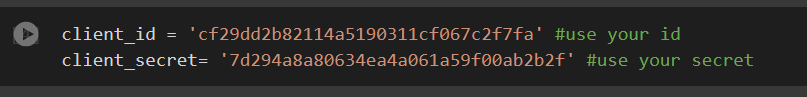
Description automatically generated

A screenshot of a computer

Description automatically generated

#### Connecting to Spotify Web API:

To fetch your Spotify playlist data, you will need to connect to the Spotify Web API using a unique client ID and a client secret key. I have already shown you how to generate these keys.



In the cell below, you can see that I have used my localhost URL. This URL is used to validate the client. I have talked about this in the Spotify Web API section. Here we are storing our playlist details in a python dictionary.

A screenshot of a computer

Description automatically generated

The cell below contains the method which creates a new dataframe for our playlist using the Spotify song features dataset that we have downloaded from Kaggle.

A screen shot of a computer

Description automatically generated

The cell below is used to visualize the cover-art of the song tracks with the help of the Matplotlib library.

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

#### Creating playlist vector:

To perform **cosine similarity** between our playlist and the songs not present in our playlist, we will summarize our playlist in one vector. This vector will represent our playlist in the feature space, and we will be able to find songs similar to the songs in our playlist.

The following code cell contains a method that returns our playlist as a single vector and all the songs not present in our playlist in a dataframe.

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

#### Generating recommendations:

As stated before, we are going to use **cosine similarity** as a similarity metric to determine the songs that are very much similar to our playlist. We will perform the cosine similarity between our playlist vector and songs not present in our playlist. Then we will perform cosine similarity using a python-based library, Scikit and store the cosine similarity values in a separate column. Next, we will reverse sort the dataframe based on the cosine similarity column. Finally, we will generate the top 15 recommendations of songs as our recommended playlist.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A black background with white text

Description automatically generated

A collage of music covers

Description automatically generated

**Points to be noted:**

Here are some points to keep in mind regarding the Content-Based Filtering mechanism we have implemented:

* The model will only be able to make recommendations based on that specific user’s interests. Therefore, it limits the ability of a user to expand their existing interests.
* The Spotify song features data that we have used to develop the recommendation system will not necessarily contain all the songs in your playlist. Therefore, sometimes we will not be able to harness the playlist fully to generate recommendations.
* To use this recommender, users should have at least one playlist on their Spotify account, which is a disadvantage.

**Conclusion:**

Using the concept of **cosine similarity**, we were able to generate recommendations of songs that are similar to the songs in our playlist. You can follow the steps above to develop your custom Spotify playlist recommender as well.

**References:**

1. <https://spotipy.readthedocs.io/en/2.16.1/>
2. <https://d3.harvard.edu/platform-rctom/submission/spotify-machine-learning-as-recommendation-engine-and-musical-composer/>
3. <https://www.kaggle.com/datasets/zaheenhamidani/ultimate-spotify-tracks-db>
4. <https://developers.google.com/machine-learning/recommendation>