Music Genre Recommender System - Detailed Documentation

1. Project Overview

The **Music Genre Recommender System** is designed to suggest music genres based on user input. Users can describe their music taste in text, and the system will match their preference using **cosine similarity** with pre-encoded genre embeddings. Additionally, it integrates with **Spotify API** to fetch song recommendations and album covers. The system also incorporates **Retrieval-Augmented Generation (RAG)** to enhance genre discovery beyond the dataset.

2. Features

- Accepts **user input** in natural language (e.g., "I like Rock").
- Uses **SentenceTransformer embeddings** to encode and compare genres.
- Implements **cosine similarity** for genre matching.
- Retrieves **top songs** from a dataset based on the matched genre.
- Uses **Spotify API** to fetch song links and album covers.
- "Surprise Me" feature for random genre recommendations.
- Handles **negation detection** (e.g., "I don't like jazz").
- Uses **RAG (Retrieval-Augmented Generation)** to find genre-related songs even if the genre is not present in the dataset.

3. Technical Stack

- **Programming Language:** Python
- **Libraries Used:**
- `numpy` : For numerical operations.
- `pandas` : To handle the dataset.
- `spotipy` : To interact with the Spotify API.
- `sentence-transformers` : For embedding generation and similarity comparison.
- `IPython.display` : To show images in Colab.
- `random` : For the surprise feature.

4. Logic & Flow

Step 1: User Input Processing

- The user enters a genre or describes their music taste.
- Common phrases like "I like" or "I love" are removed.
- Input is converted to lowercase for consistency.

Step 2: Embedding Generation

- A pre-trained SentenceTransformer model (`all-MiniLM-L6-v2`) is used to **convert genre names into embeddings**.
- The **genre list** from the dataset is encoded once and stored.

Step 3: Similarity Calculation

- The system encodes the user's input into an embedding.
- **Cosine similarity** is computed between the user embedding and all genre embeddings.
- The **most similar genre** is selected based on the highest similarity score.

Step 4: Retrieval-Augmented Generation (RAG) with Spotify API

- If a genre is found in the dataset, top songs are recommended.
- If no exact match is found, **Spotify API** is queried using RAG principles:
- The system retrieves similar genres using embeddings.
- If the genre is missing, it queries **Spotify API** for tracks related to the input.
- Spotify's results are ranked based on relevance and displayed.

Step 5: Song Recommendation

- If a matching genre is found:
- The top 5 songs from the dataset in that genre are displayed.
- **Spotify API** is used to get a song link and album cover.
- If no match is found:
- The system searches **Spotify** directly for related tracks.

Step 6: Handling Special Cases

- **Negation Handling:** If the user states dislikes (e.g., "I hate jazz"), the system finds an alternative genre.
- **Surprise Me Feature:** If selected, a random favorite genre is chosen for recommendation.

5. Cosine Similarity Explanation

Cosine Similarity is used to measure how similar two text embeddings are. The formula is:

$$\text{dot}(A, B) / (||A|| * ||B||)$$

where:

- `A` is the user's input embedding.
- `B` is a genre embedding.
- `dot(A, B)` is the dot product of the two vectors.
- `||A||` and `||B||` are the magnitudes of the vectors.
- The result is a similarity score between **-1 and 1** (higher means more similar).

6. Example Workflow

Example 1: User enters "I like Rock"

- The phrase "I like" is removed, leaving "Rock".
- Rock is converted into an embedding.
- Cosine similarity finds the closest genre in the dataset (e.g., "Rock").
- The system recommends the **top 5 Rock songs** and provides **Spotify links**.

Example 2: User enters "I don't like Jazz"

- The system detects negation.

- It finds the closest genre to Jazz.
- Instead of Jazz, it suggests a genre with lower similarity (e.g., "Blues").

Example 3: User enters an unknown genre like "Phonk"

- The genre is not found in the dataset.
- The system queries **Spotify API** for tracks labeled "Phonk".
- RAG retrieves related results and ranks them based on similarity.
- The most relevant tracks are recommended to the user.

7. Conclusion

This project leverages **machine learning**, **text embeddings**, **retrieval-augmented generation (RAG)**, and **music APIs** to deliver a dynamic genre recommendation system. Future improvements can include **personalized playlists, user profiles, and deeper NLP analysis**.

This document provides a structured understanding of the **Music Genre Recommender** system, including logic, implementation, and cosine similarity concepts.

Music Genre Recommender System - Documentation

1. Project Overview

The **Music Genre Recommender System** suggests music genres based on user input. Users can describe their music taste in text, and the system matches their preference using **cosine similarity** with pre-encoded genre embeddings. It also integrates with **Spotify API** to fetch song recommendations and album covers. Additionally, the system uses **Retrieval-Augmented Generation (RAG)** to discover genres beyond the dataset.

2. Features

- Accepts **natural language input** (e.g., "I like Rock").
- Uses **SentenceTransformer embeddings** for genre encoding.
- Computes **cosine similarity** for genre matching.
- Retrieves **top songs** from a dataset based on genre.
- Fetches **Spotify song links & album covers**.
- **"Surprise Me"** feature for random genre recommendations.
- Detects **negation handling** (e.g., "I don't like jazz").
- Uses **RAG** to find genre-related songs even if the genre is missing in the dataset.

3. Technical Stack

- **Programming Language**: Python
- **Libraries Used**:
- `numpy` : For numerical operations.
- `pandas` : For handling the dataset.
- `spotipy` : For interacting with Spotify API.
- `sentence-transformers` : For embeddings & similarity calculations.
- `IPython.display` : To display images in Colab.
- `random` : For the "Surprise Me" feature.

4. Logic & Flow

Step 1: User Input Processing

- User enters a genre or describes their music taste.
- Common phrases ("I like", "I love") are removed.
- Input is converted to **lowercase** for consistency.

Step 2: Embedding Generation

- Uses **SentenceTransformer ('all-MiniLM-L6-v2')** to generate embeddings.
- The genre list from the dataset is **pre-encoded and stored**.

Step 3: Similarity Calculation

- Converts user input into an embedding.
- Computes **cosine similarity** between input and genre embeddings.
- Selects the **genre with the highest similarity score**.

Step 4: RAG & Spotify API Integration

- If a genre is found in the dataset, **top songs are recommended**.
- If no match is found, the system queries **Spotify API using RAG**:
- Retrieves **similar genres** using embeddings.
- If missing, it **queries Spotify API** for related tracks.
- **Ranks results** based on relevance and displays them.

Step 5: Song Recommendation

- **If a matching genre is found**:
- Displays **top 5 songs** from the dataset.
- Uses **Spotify API** to get **song links & album covers**.
- **If no match is found**:
- Searches **Spotify directly** for related tracks.

Step 6: Handling Special Cases

- **Negation Handling**: Detects dislikes (e.g., "I hate jazz") and finds an alternative genre.
- **Surprise Me Feature**: Recommends a **random genre**.

5. Cosine Similarity Explanation

- Cosine Similarity measures the similarity between two text embeddings.
- Formula: **cosine_sim(A, B) = dot(A, B) / (||A|| * ||B||) Where:**
- `A` = User's input embedding
- `B` = Genre embedding
- `dot(A, B)` = Dot product of the vectors
- `||A||` and `||B||` = Magnitudes of the vectors
- **Result is between -1 and 1** (higher means more similarity)

6. Example Workflow

Example 1: User enters "I like Rock"

- "I like" is removed → "Rock" remains.
- Rock is converted into an embedding.
- Cosine similarity finds the closest genre (e.g., **"Rock"**).
- **Top 5 Rock songs are recommended** with **Spotify links**.

Example 2: User enters "I don't like Jazz"

- System detects **negation**.
- Finds the **closest genre to Jazz**.
- Instead of Jazz, it suggests an alternative (e.g., **"Blues"**).

Example 3: User enters an unknown genre like "Phonk"

- **Genre not found in dataset**.
- System **queries Spotify API** for "Phonk" tracks.
- **RAG retrieves related genres & ranks them**.
- **Most relevant tracks are recommended**.

7. Conclusion

The **Music Genre Recommender System** integrates **Machine Learning, Text Embeddings, Retrieval-Augmented Generation (RAG), and Music APIs** to provide a **dynamic genre recommendation experience**. Future enhancements can include:

- **Personalized Playlists** 🎵
- **User Profiles** 👤
- **Advanced NLP Analysis** 🧠

This documentation provides a structured understanding of the system's **logic, implementation, and key concepts**.