CSC111 Project Report: Algorithmic Music Recommender

Mohamed Abdelfattah, Johnson Qin, Olivier D'Aragon Flores & Hongyu Zhu

Monday, March 31, 2025

Problem Description and Research Question

We have chosen to create a program that suggests songs based on a user's interests because when listening to music, we often find ourselves listening to the same songs over and over again, without leaving our comfort zone and exploring new songs. We want to develop a tool that will suggest songs similar to our interests to add variety in our playlist and ensure that new songs align with our preferences, all without having to take the time to find new songs ourselves. As such, we decided to make a recommendation algorithm to help us find similar music more efficiently, and spice up our playlists a little bit, all without wasting our time!

Project Goal: Recommend songs to users by taking in songs they like and finding similar songs. The algorithm will compare the characteristics of the song from a Spotify song database to generate predictions. The experience should be customizable, allowing users to tweak recommendation parameters in order to obtain different results.

The user should be aware of some different variables associated with each song (some examples below):

- `Danceability`: "describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable."
- `Energy`: "a measure from 0.0 to 1.0 [that] represents a perceptual measure of intensity and activity."
- `Speechiness`: "detects the presence of spoken words in a track. The more [...] speech-like the recording [...], the closer to 1.0 the attribute value."

Datasets Used

`Name:` 30,000 Spotify Songs
`Description:` This dataset is a `.csv` file containing around 30,000 songs scraped by a user on Kaggle. Each song has associated information: some of it is an intrinsic part of the song (such as the title of the song or the artist), whereas the rest is based Spotify's own calculations (such as valence or energy). By using these calculated characteristics, we can compare songs and figure out how similar they are to each other. All information is a string, which is why we manually convert them into their respective datatypes.
The information in the CSV (not in order):

1. `track_id`: ID of the song.
2. `track_name`: Name of the song.
3. `track_artist`: Name of the artist.
4. `track_album_id`: ID of the album.
5. `track_album_name`: Name of the album.
6. `track_album_release_date`: Date of album release.
7. `playlist_name`: Name of the playlist.

8. `playlist_genre`: Genre of the playlist this song was found in.
9. `playlist_id`: ID of the playlist.
10. `danceability`: How danceable the song is. (Spotify)
11. `energy`: How energetic the song is. (Spotify)
12. `key`: The key of the song. (Spotify)
13. `loudness`: How loud the song is. (Spotify)
14. `mode`: Is the song in a major or minor key. (Spotify)
15. `speechiness`: How prevalent spoken word in the song is. Ex: is the song more electronic or singer-songwriter. (Spotify)
16. `acousticness`: How "acoustic" the song is. (Spotify)
17. `instrumentalness`: How instrumental the song is. (Spotify)
18. `liveness`: How confident we are that the song is a live performance. (Spotify)
19. `valence`: How happy/sad the song is. (Spotify)
20. `tempo`: The tempo of the song. (Spotify)
21. `duration_ms`: How long the song is. (Spotify)

`Format:` csv file
`Source:` https://www.kaggle.com/datasets/joebeachcapital/30000-spotify-songs
`Columns used:` track_id, track_name, track_artist, track_popularity, playlist_genre, danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, duration_ms
`Columns omitted:` track_album_id, track_album_name, track_album_release_date, playlist_name, playlist_id, playlist_subgenre

Computational Overview

The program takes an input song and personalized search parameters, performs calculations, and returns recommended similar songs.
The calculations work by assigning different weights to each attribute of the input song. For example, a higher weight for energy means that our algorithm will consider energy more than other attributes when finding similar songs. The user has the option to either use default or customized weights.
Personalized search parameters include:

- What song attributes to prioritize
- How many songs to input (ex: 1 song only)
- How many songs to receive (ex: 5 similar songs)

Each song in our database is represented using a mapping (dictionary) from a string to a list. The string is the song's title, and each element of the list is an attribute of the song, such as the song's artist, its popularity, its tempo, etc.

Our program works interactively by asking for user input and customization of search parameters. For any song, the program generates a graph of similar songs, linking those who more closely resemble each other, grouping similar ones together in order to show the relationship they have to each other. A shorter distance between two songs means that they are more similar. In addition, the user can see a bar graph showing which elements of their selected song are prioritized when calculating its similarity to the other songs in the dataset.

The Numpy library allows us to manipulate `np.arrays`, which are considerably more computationally efficient than python lists. Numpy also has very fast built-in mathematical operations on these arrays, optimized in C/C++, perfect for large-scale data parsing, such as `diff()` or `cross()`.
We originally said we would use Pandas to debug code, but we ended up not needing to use it. Also, rather than

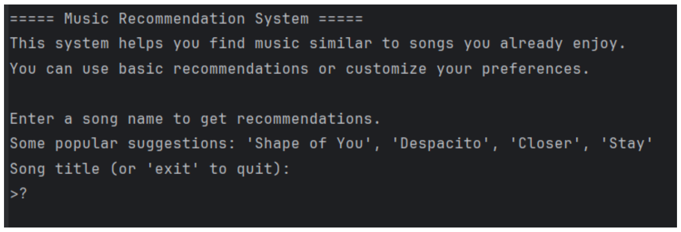
using matplotlib for the graph, we used plotly instead.
Our program utilizes Python *graphs* to connect a particular song with songs that are similar to it, according to our algorithm. When interacting with the program, the user is allowed to see the similarity graph of their selected song. This gives them the option of a visual representation of songs similar to their selected one.

Program Structure + Key Components
The user is prompted to enter a song and whether they want to customize how the songs are recommended. For example, they can restrict the recommended songs to be by the same artist, they can prioritize valence, and more. Afterwards, the program displays a list of recommended songs and gives the user the option to see a graph displaying the similarity of their selected song with the recommended ones, where a closer distance between two nodes indicates greater similarity.
Furthermore, the user can see a feature analysis of their song, displaying the numerical values of the attributes of their song (valence, tempo, etc.) as well as a bar chart showing which song attributes were weighted most heavily when finding similar songs.
Key program components include:

- Use of networkx and plotly modules to create the graph
- The algorithm to calculate song similarity using customizable weighted variables
- Asking for user input to allow them to customize their recommendations
- Function to convert the song data into a Python dictionary

Instructions for Obtaining Dataset and Running Program

The dataset can be downloaded in the attached zip file named **SpotifySongs_no_id.zip**. The user must ensure that the extracted dataset (in csv format) is in the same directory as `main.py`.
The following Python libraries must be downloaded: `numpy, networkx, plotly`
This is what the user should see when running `main.py`



```
===== Music Recommendation System =====
This system helps you find music similar to songs you already enjoy.
You can use basic recommendations or customize your preferences.

Enter a song name to get recommendations.
Some popular suggestions: 'Shape of You', 'Despacito', 'Closer', 'Stay'
Song title (or 'exit' to quit):
>?
```

Figure 1: After running main.py

Changes To Project Plan

Instead of having the user input a playlist of songs, which was our initial idea, only one is asked. Furthermore, the graphs in our program are used as a visual representation of a song's similarity to other songs and cannot be manipulated. We scrapped the use of the **pandas** module, and instead of **matplotlib**, we used **plotly** for the graph. Lastly, instead of having multiple datasets varying in size as originally planned, we only used the full 30,000 song dataset in our program.

We added a bar chart to display which attributes are taken into account the most when finding similar songs, something that was not in our original plan.
We also added an option for multi-song comparison, which allows the user to see a graph of their song and another one, and how these graphs compare to each other (how similar they are).

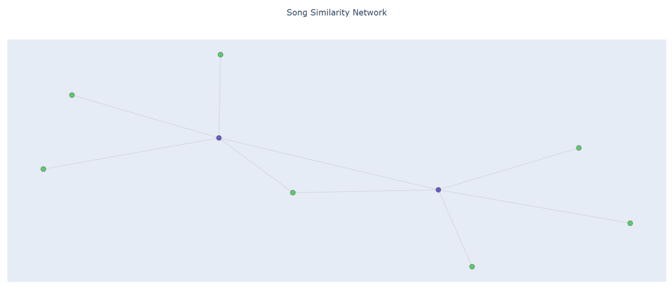


Figure 2: In this case, the two songs selected by the user (in purple) are extremely similar because their nodes are connected

Discussion

The results of our computations helped us achieve our project goal, as we successfully developed an accurate algorithm to calculate similar songs. We can now enter a song of our choice and interactively select settings for the algorithm to quickly get recommendations. Furthermore, we can choose how many recommendations we would like to have, allowing the user to as many similar songs to their playlist as they like.

Using weights to emphasize certain attributes is a simple and powerful way to calculate the similarity between different songs. Additionally, the combination of the Euclidean distance formula and the networkx and plotly Python modules create an effective visualization of how similar a song is to others, as the graph allows us to connect songs to similar ones, and the Euclidean distance formula allows us to directly correlate song similarity with distance on the graph (that is, the closer the song nodes, the more similar the songs are). Together, both of these produce an effective and straightforward visualization.

One of the limitations of our dataset is that it contains duplicate song titles. For example, there are multiple songs in the dataset titled *Say My Name*, and Ed Sheeran's famous *Shape of You* has several remixes and duplicates in the dataset. Consequently, for our program, entering a certain song may result in a remix or a different song with the same title inadvertently being used for computation.
Otherwise, the dataset is well suited for finding similar songs, given the fact that it contains numerous attributes of a song that listeners pay the most attention to. Specifically, valence, energy and danceability are some attributes included in the dataset that are highly relevant to completing our project goal.

For future study, exploration and improvements, our algorithm could be improved through extensive data analysis of the songs. To fix the issue of remixes and duplicates, we could manually edit the dataset to remove these. We could also consider expanding our dataset, as it consists mostly of modern songs, leaving out some classic songs of the past like The Beatles' *Yellow Submarine*. What's more, the dataset is limited to six main genres, including "EDM", "Rock" and "Pop" among others. As a result, when entering songs, the user is limited to entering mostly modern mainstream songs, a problem that also affects the songs returned by our program. By expanding our dataset to include more songs from previous eras (such as the Baby Boomer Generation and Generation X (1946-1980)) and more niche genres, older people, or more niche users could get more relevant and accurate recommendations.

Should any of us come back to this project a few years down the line, with our increased knowledge and experience of computer science programming, we could look into developing an artificial intelligence (AI) model that would be able to use more abstract information, such as date/time period, or event sentiment analysis of the song title. Just like processing language, the processing of abstract information is difficult to hard-code. However, AI techniques such as tokenizing or even fully using an LLM to assist us, could help us access this additional information.

To sum up, our program takes in a song, compares it to others by performing a customizeable similarity score calculation, and then returns the most similar songs visually with a graph. By taking in custom weights and filter preferences, we effectively offer a custom search experience that more advanced users benefit from. For less well-versed users, we also offer default parameters. However, the presence of duplicates and remixes in the dataset along with a bias towards modern songs can affect both accuracy and usability of our program. Future improvements include editing the dataset to remove unnecessary remixes and duplicates and expanding the dataset further to get a broader scope of music recommendations, as well as the possible use of AI for more precise recommendations.

References

Arvidsson, Joakim. “30000 Spotify Songs.” Kaggle, 1 Nov. 2023, www.kaggle.com/datasets/joebeachcapital/30000-spotify-songs.
https://numpy.org/doc/stable/index.html
https://plotly.com/python/
https://networkx.org/
Github: https://github.com/JoeZhu-CS/Algorithmic-Music-Recommender/tree/main