CSC111 Project 2 Report: “Music for Your Ears to Hear” - A Music Recommendation System

Aakaash Rohra, Daniel Xie, Ethan Chiu, Jackie Chen

April 3, 2024

Introduction

In the age of digital streaming, while it is infinitely easier to find music to listen to, the vast landscape of music available to stream and the ease with which one can create and upload new music makes finding similar songs to those you enjoy a potentially daunting or overwhelming task. Thus, music recommendation systems have been on the rise in an effort to help guide the average listener, who would like to avoid spending hours finding music they might actually enjoy (with so much out there, it’s much more likely that one would stumble across a new song and discover it’s not their cup of tea). Thus, our goal of creating a music recommendation system based on this dataset and method of categorization is relevant to the real, modern-day music consumer landscape. **Our goal is to create a functional and justifiably accurate music recommendation software that ranks songs according to a variety of factors and uses those factors to generate a list of recommendations for a user, based on an inputted song that they already like. Alongside the recommendations will be links to a YouTube search for each song, making it convenient for the user to listen and see if they match their tastes, or add them to their playlists.**

Dataset Information

The dataset we used for our project is titled “Prediction of music genre”, collected by Vicsuperman and posted on Kaggle. It contains 41700 different songs and a variety of data on them that relates to their genres in some way (some more explicitly than others). It is in the public domain, and can be found in the references section below. The data included for each song, as well as a description for those that are not as self-explanatory, is the following:

1. Popularity
2. Acousticness: the extent to which a track involves acoustic instruments (any instrument that is not powered electronically in some way).
3. Danceability: the extent to which a track is “danceable” - that is, has a groove or beat that feels intuitively easy to dance to.
4. Duration
5. Energy: the level of energy a song has (more going on vs. more barren instrumentally).
6. Instrumentalness: whether the song highlights instrumentals more, or a vocal or other factor more.
7. Key: a music theory concept that will not be as relevant for our goal.
8. Liveness: the extent to which a recorded song sounds as though it could be performed live.
9. Loudness: how loud a song is overall, measured in decibels.
10. Mode: whether a song is in the major or minor mode; another concept that will not be very relevant for our goal.
11. Speechiness: the extent to which a song contains vocals.
12. Tempo: the beats per minute of a song.

13. Obtained Date
14. Valence: how 'happy' a song sounds.
15. Music Genre

Of these, we chose to use a select few that seemed the most relevant in determining the similarity of one song to another. We used acousticness, danceability, energy, instrumentalness, liveness, valence, and genre. The values for each category that we used (except genre) range between 0.0 and 1.0, and are all relative to each song, making them more valuable in comparing one song to another in similarity. Music genre was used to limit recommendations to those which belong to the same genre as the inputted song, as a) it would limit the loading of the graph, resulting in a more reasonable time (under 1 minute) to load, and b) songs within the same genre are the most likely to be effective recommendations.

Computational Overview

In this program, the user is first prompted to input a song to generate recommendations for. Using the Python pandas library, we wrangle the dataset to remove the irrelevant columns as mentioned in the previous section, and build a graph using only the portion of the dataset consisting of songs within the same genre as that of the user's inputted song. To build this graph, each song of the same genre is added as a vertex, with its instance ID being its item and its neighbours being the 15 most similar songs. We calculate which songs are most similar to a target song by taking another song in the same genre and computing the absolute difference of the ratings in each category. These differences are then added up, resulting in a "similarity score," where a lower score means a song is more similar. After this is done for all relevant songs, the 15 songs with the lowest similarity scores form weighted edges with the target song, with weights corresponding to their similarity score. Lastly, this process is repeated with all songs in the wrangled dataset, which completes the graph loading. This graph is then stored as a value in a dictionary with the genre as the key, so it can be reused for different input songs within the same genre. With this, it is important to note that some songs have more neighbours than others, as while one song may be in another song's top 15 most similar list, the opposite is not necessarily true. However, since the similarity score is contained within the weighted edges, this is a non-issue, as simply taking the songs with the lowest 15 similarity scores gives the intended recommendations. Following the graph building, a PDF file is created which contains the 15 most similar songs to the inputted song, found by navigating the graph, as well as black boxes next to each song. Upon clicking these boxes, the user is taken to a YouTube search results page with the name of the recommended song as the search.

The ReportLab library was used to generate PDF files. A canvas for a page in the PDF is first created, with elements then being drawn onto it. Each PDF has a unique name according to the inputted song, which prevents overwriting the same file for each new list of recommendations (unless the same song is inputted, in which case the recommendations would be identical anyways). This is done with a coordinate (x, y) system. The x and y positions are first decided on according to 1 inch margins on a letter size paper, and are adjusted after each text line or element is drawn in. An invisible clickable box is created above each black square that links to the YouTube search page for the song, and is aligned such that it looks like the same element on the PDF. Finally, the canvas is saved.

General Instructions

The dataset we used is a single CSV file, and is present in our MarkUs submission zip file. Running the main.py file will run our entire program. The user would see prompts for input in the console, followed by confirmation that a PDF has been created at the same file path as the main.py file, and other files associated with the project. The user can then navigate to this folder on their machine and access the generated PDF, which contains a numbered list of 15 recommendations with black squares that can be clicked to direct the user to a YouTube search page for the associated song.

Changes to Plan After Proposal

Overall, our project remained very similar to our proposal, aside from a few computational differences - the overall goal and final product are roughly the same. The YouTube links are a new addition that was not present in our proposal, which is roughly based on TA feedback to allow for playlist functionality in some way - while we did not directly deal with playlists, we set up links in the generated PDF for each song, to allow for easy access to recommendations, at which point the user can easily add songs they enjoy to their own playlists. Aside from this, as previously explained, the graph is loaded after taking input so that it can be restricted to a single music genre, which was not initially our aim - the reasoning for implementing the graph this way was also described earlier, in the "dataset information" section. While we were not entirely sure at the time of writing our proposal how we would weigh different factors from the dataset for each song, we ultimately decided on an even weightage system. This worked out well, as each category we used was measured relatively to other songs, and thus was already weighted to start, in a sense. We subjectively decided upon seeing the resulting recommendations that this system worked well. We also allowed for the user to keep asking for recommendations for new songs, which was not initially our plan. More on this can be found in the ”discussion” section.

Discussion

Our project does achieve the goal we set out to achieve. The following is a discussion on the alterations and limitations we encountered throughout the process.

Throughout the development of this project, we made many alterations to our implementations, small and big. One of the greatest changes we made relates to the efficacy of graphs in providing recommendations. Initially, we created the graph without limiting it to songs of the same genre. However, this came with two issues. The first and, at the time, the most important issue was that this would result in loading times of over 5 minutes from building the graph. Thus, we changed it so it only included songs of the same genre. The next problem was that in our first implementation, we did not allow users to get recommendations for multiple songs. As a result, using a graph was not efficient because a graph would create vertices and edges for every song of a genre, which would take more time than just computing the similarity scores for only the inputted song and would achieve the same result. So, to justify using a graph as our solution, we decided to allow users to get recommendations for multiple songs. This way, loading the graphs would be reasonable, as we could then store them for later use to quickly find recommendations for other songs. Furthermore, this would be faster than computing the similarity scores for only the inputted song because using the graph, we would only need to find the neighbours of the song in the graph.

Aside from the graph implementation limitation that we addressed as we worked, we encountered a number of limitations upon reflection of our project. An obvious one is that this dataset, while seemingly large, is finite. In the vast world of music, with all the new tracks released daily, any recommendation system based on a finite, static dataset is prone to inaccuracy as it can only offer recommendations from within the dataset. We do not have access to any sort of auto-updating, active dataset from some big streaming service such as Spotify or YouTube, which would be the only way to tackle this issue.

An interesting bug that we could not even begin to address is the lack of compatibility for foreign characters. There are some songs in this dataset (namely, those with the "anime" genre tag) that have foreign characters in their titles or artist names. When these appear on the list of recommendations on the PDF, the characters appear as black boxes. This is rooted in the non-existence of these characters in ASCII, existing instead as unicode characters, which was not a field we were able to tackle in our project. This is a fairly large limitation given the massive presence of eastern music in the overall world of music consumption and creation.

A possible next step for a more effective recommendation system would be the presence of more categories for a similarity score, as more metrics by which to compare songs to each other would inevitably result in better recommendations. Alongside this, while we subjectively believe our algorithm for calculating a similarity score to be effective enough for our recommendation system, some more exploration towards different weights based on the category might prove fruitful.

Works Cited

Python Assets. “Create PDF Documents in Python with ReportLab.” Python Assets, 12 Feb. 2022, pythonassets.com/posts/create-pdf-documents-in-python-with-reportlab/.

Stratton, Jerry. “Adding Links to PDF in Python.” Mimsy Were the Borogoves, 15 June 2007, www.hoboes.com/Mimsy/hacks/adding-links-to-pdf/.

Vicsuperman. “Prediction of Music Genre.” Kaggle, 2 Nov. 2021, kaggle.com/datasets/vicsuperman/prediction-of-music-genre/data.