CSC111 Project Proposal: Interactive Music Genre and Album Recommendation Tree

David Wu, Kevin Hu

Tuesday, March 16, 2021

Problem Description and Project Goal

To many of us, music is a large part of our lives, helping us get through the day. According to the International Federation of the Phonographic Industry (IFPI), the average person spent 18.4 hours a week listening to music in 2021[1]. With the sheer number of music available on modern streaming platforms such as Spotify or Apple Music, it can be extremely difficult and time-consuming for the casual listener to find new music that they enjoy, especially since listeners themselves may not know what they like. A listener may know that they typically enjoy Rock music but "Rock" is a broad term that encompasses many distinct subgenres, perhaps they are into the raw, distorted sound of Noise Rock or maybe they prefer the more technical, complicated time signatures of Math Rock. This is not to mention that different albums within a subgenre could also be further categorized by different descriptors such as warm, aggressive or depressing. The casual listener is simply unaware of these distinctions when trying to find something they like on the Rock section of Spotify. We believe that good music recommendations should be fast and easy. **Our goal is to provide users with the best possible song recommendations, given the current preferences of the user in either the form of a preferred genre or a favourite album,** letting users spend less time looking for music and more time listening. This would be done with a visual element in the form of a tree, with the best recommendations located towards the top, and the subsequent branches giving the user options to discover other musical pathways on their own. We would also provide an interactive element for users to further explore their recommendations and how they fit within the larger tree of music genres and albums, allowing users to potentially discover new songs and albums they can enjoy, without having to search through individual songs on music streaming sites.

Datasets Used

1. Rate Your Music: The Top 5,000 Most Popular Albums(rym_clean1.csv)[2]

Found on kaggle, this dataset contains 5,000 of the most popular albums found on the music database and review website rateyourmusic.com (RYM). The popularity of the albums was determined by the number of reviews each album had, showing the number of meaningful interactions. As a downloadable csv file with the name `rym_clean1.csv`, each line contains the information of a specific album, with the following attributes in order: `position`(popularity rank), `release_name`, `artist_name`, `release_date`, `release_type`, `primary_genres`, `secondary_genres`, `descriptors`, `avg_rating`, `rating_count`, and `review_count`. In our program, we will primarily use the `position`, `release_name`, `artist_name`, `primary_genres`, `release_date`, and `descriptors` attributes for computations. Our program also utilizes all 5001 lines in the csv file to generate the best possible recommendations(excluding line 1 with includes the header information).

2. Genres Dataset(genres_dataset.csv)

Using information from rateyourmusic.com (RYM)[3], we generated a data set containing all music genres and subsequent subgenres found on RYM. In the file `genres_dataset.csv`, each line contains the information of a music genre, with the attributes in the order `genre`, `parent genre`. `genre` is the name of the genre, while `parent genre` is the genre under which Genre may be listed. If `genre` is a main genre with no parent genre, `parent genre` is given the value `NA`. All attributes are used for computations in our program, as well as all 1817 lines in the csv while(excluding line 1 with includes the header information).

Computational Overview

As the main database for our recommendations(`rym_clean1.csv`), we will use the top 5,000 most popular albums pulled from the music review site rateyourmusic.com (RYM). As mentioned in the previous section, the main attributes we will be utilizing are `position`, `release_name`, `artist_name`, `primary_genres`, `release_date`, and `descriptors` to help with recommendation computations and visualizations. To help identify genres and their subgenres, we have created a genres dataset(`genres_dataset.csv`), which contains the name of all music genres listed on RYM, and their parent genre if there are any. Our projects will then primarily use trees to represent the genres and albums in our datasets in a hierarchical manner, with three main trees: the genre tree, the genre recommendation tree, and the album recommendation tree. Starting with the genre tree, this tree is separate from the other recommendation tree as it serves as more of an informational role. The starting node of the tree just has the title "Genre", which branches off into the main music genres(i.e. the genres with no parent genre), and then the tree branches off into the subgenres. As for the album recommendation tree, its root will be an album inputted by the user(the album will be from the top 5,000 albums dataset). From there, the tree will branch into the top recommendations, which will be calculated by finding other albums in the dataset with the most matching strings from the descriptors attribute. The tree will then recurse through the top recommendations as if they were the album inputted in by the user, but will also avoid using already recommended albums for further subtrees. The genre recommendation tree is similar to the album recommendation tree, except the root will instead be a genre inputted by the user. The first group of subtrees will then be the most popular albums of the inputted genres. From there, further subtrees will be added using the same recommendation algorithm used in the album recommendation tree.

To extract data from our datasets, we have created the `genres_data` and `albums_data` files, which contain the `Genre` class and `Album` class respectively. These classes serve to represent each individual genre and album, with the relevant attribute pulled from the top 5,000 most popular albums dataset, and genres dataset. Here is a look at the Album class:

```
class Album:
"""
a class to represent an album.
"""
name: str
artist: str
genres: list[str]
rank: int
release: str
descriptors: list[str]
```

The instance attributes include the name of the album, the name of the artist, a list of the album's associated genres, its position on the music database RateYourMusic's list of most popular albums, it's the release date and a list of descriptors for the album. Using an import from `csv`, the `albums_data` file will also read the data from the top 5,000 most popular albums dataset, converting each line in the dataset into an `Album` object, and storing everything from this dataset into a list called `albums`, which is a list of `Album` instances. This `albums` list will later be imported into other classes and files for computations. The `genre_data` file is similar to `albums_data`, except it only has the instance attributes `name`(a string), and `parent_genre`(which is defined as `Optional[str]`). `parent_genre` is `None` when the genre is a main genre. Using `csv` imports again, each line in the genres dataset is read and converted into a `Genre` instance, which is all saved into a list called `genres`.

The main tree structures and methods are provided in `tree_classes`. Within the `tree_classes` file contains the class `Tree`, with the typical implementation and instance attributes `_root`(which is defined as `Optional[Any]`) and `_subtrees`(which is defined as `list[Any]`). `Tree` is also implemented with some of the standard methods(i.e. `is_empty`). The child class `Album Tree` is also implemented here, which is initialized the same way as `Tree`.

The major computations of our program are done in the `plot_recommendation_tree` and `plot_genre_tree` files. Examining `plot_recommendation_tree`, the main attraction of our program, the recommendation algorithm is done through the function `get_albums_by_matches`, which outputs a list of `Album` instances. Taking in a given `Album` and a list of `Album` instances, this function compares the number of matching `descriptors` from the given `Album` and each album in the list of `Album` instances(as long as the given `Album` is not the same as the compared `Album`). This function also takes in a visited set, stopping comparisons from the given `Album` to any albums in `visited`, which is later used for tree generation. After each comparison between the given and compared to `Album`, the compared to `Album` is stored in a dictionary `album_to_matches`, mapping the `Album` name to the actual `Album` instance, and the number of matches. The final parameter of this function is `num_recommendations`, which is an `int` that decides how long the output list is. Based of `num_recommendations`, that number of albums is pulled out of `album_to_matches` with the highest number of matches for `descriptors` being pulled first. This also results in the output to `get_albums_by_matches` being a sorted list, with the earlier elements having more matches than the later elements. The next main function is `generate_album_recommendation_tree`, which is what actually creates the frame for the album recommendation tree. It takes in a `depth` parameter to control the size of the tree. Then, from a given `root_album`, the function uses the algorithm from `get_albums_by_matches` to find the best-recommended albums and creates a subtree for each recommendation. `generate_album_recommendation_tree` then recuses through each subtree to further develop the recommendation tree. A visited set is taken in by this function, which is fed into `get_albums_by_matches` to make sure further subtrees down the tree don't repeat the same recommendations. The actual visualization of the album recommendation tree is then done through the function `plot_album_recommendation_tree`, which further explores the `plotly` library and the `igraph` library. Using the implementation of tree plots found in the `plotly` library[5], `plot_album_recommendation_tree` generates a visualization of the tree generated from `generate_album_recommendation_tree`, using the roots and branches of the tree(i.e. vertices and edges). To get all the vertices and edges of a tree, two recursive helper functions, `get_all_branches` and `get_all_vertices` was created. Both have essentially the same implementation, recursing through a given tree and saving all vertices or edges(a tuple of vertices) into a list. Using the list of vertices and edges obtained from these helper functions, `plot_album_recommendation_tree` creates a visualization of the tree.

*************insert photo

As for the genre recommendation tree, when the user inputs a genre, the top albums of the inputted genre are outputted, and then the album recommendation tree takes over for further subtrees. Only the root and direct subtrees are needed for the genre recommendation tree, so there is no need to create a separate tree generation function. Instead, given the `name` attribute of a `Genre`, the function `get_albums_by_genre_and_popularity` takes in a list of `Album` instances and filters it, returning a list of `Album` instances with the same genre name as the inputted `Genre`. This function was also intended to sort the returned list of `Album` instances from most to least popular, however, due to the top 5,000 albums dataset already being sorted in order by popularity, the inputted `albums` list is already sorted, so `get_albums_by_genre_and_popularity` only acts as a filter. `get_albums_by_genre_and_popularity` is then used in `plot_genre_recommendation_tree` to find recommendations for a given genre. `plot_genre_recommendation_tree` has the same `plotly` implementation as `plot_album_recommendation_tree`, with the edges instead being between the inputted `Genre` and the outputs from `get_albums_by_genre_and_popularity`. `plot_genre_recommendation_tree` will then display a tree with at most 10 recommendations for the inputted.

In the file `plot_genre_tree`, this is used to create a visualization for the informational genre tree, but still uses the same `plotly` implementation as `plot_album_recommendation_tree` and `plot_genre_recommendation_tree` for tree plot visualizations. Using the list of genre names `genres` from `genres_data` the `name` attribute of each `Genre` in the list is made into a vertex, and an edge from each `Genre`'s `name` attribute to its `parent_genre` is created. These vertices and edges are used in both plotting functions in `plot_genre_tree`.

Instructions for Use

Changes to Project Plan

From our original project proposal, overall, we have not changed much. Our original project goal has stayed as same(as outlined in the Problem Description and Project Goal section), and we have created our three genre trees, genre recommendation tree, and album recommendation tree as intended. As cited in our original project proposal, we have also continued on using Plotly tree plots[5] and Plotly Dash[4] to create a visualization of our created trees, and an interactable interface for the user. After given some suggestions, we considered using the Tkinter library(notably Tkinter `treeview` and Tkinter `pytree`, however after some struggles to create a clean interface to display both the graph and interactable elements, we decided to continue with Plotly. As for the mention of using the Spotify API in our project proposal to give a playback of the recommended album, we decide to use Spotipy to implement this feature.

Analysis and Interpretation of Results

Here is the final product:

******************insert photo

Overall, we have achieved our goal of providing a convenient way to get song recommendations given a preferred genre or album. Through a couple of clicks, users can easily see the recommendations for their input in a hierarchical manner, with better recommendations intuitively found at the top near the input. Users can also explore new songs and genres by simply clicking on a recommendation they like. Due to the use of trees and their recursive implementation, users can potentially explore a continuous branch of subtrees with albums they enjoy. If they like a certain recommendation, they can go further down into the subtrees of that recommendation. We believe our use of album descriptors in our recommendation algorithm also is better than simply using genres, which is why the main feature of our program is the album recommendation tree. While people often have a preferred genre, only recommending albums of a specific genre may be too limiting, and doesn't allow the user to potentially explore new albums and genres they may actually like. From our vision developed during our project proposal, we are satisfied with the result

However, while we have provided a quick and convenient way to find song recommendations, whether these are the 'best' recommendations is up for debate. First is the limitation of our datasets, which only contains 5,000 albums, found as the most popular albums on rateyourmusic.com (RYM). There are likely millions of music albums in the world, but our recommendation tree only builds off the information from RYM. This dataset also is of the most popular albums, not individual songs. A user may only like one or two songs from a certain album, possibly wasting time for them. Our genre dataset, it is also limited to the genres on RYM, which may be labelled/described differently from other sources. Due to the limited scope of the datasets, our recommendation trees are building off of, it is hard to tell how well it produced good recommendations. Another limitation is in combination with our recommendation algorithm and datasets. While we believe album descriptors are an overall better measure for recommendations than genres, the algorithm is still limited to one element of the album. These descriptions are also once again, limited to the descriptors found on RYM. It may be difficult to produce good recommendations when only one attribute of the albums is considered.

For future exploration, updating the recommendation algorithm to use multiple parameters, such as adding the genre, release date, artist, and more may help produce better recommendations. Adding more datasets to allow the algorithm to use more songs and albums would also be beneficial, though the running time of the program may increase. We were also given the suggestion to implement the recommendation trees as binary search trees(BSTs), with albums that have more matches located on the right, and albums with fewer matches on the left. While this implementation isn't necessarily a better way to organize recommendations, it does provide a different viewpoint, which may be preferred to be some users. An option to switch between the current implementation and the BST implementation can be another future idea to improve the accuracy of recommendations.

To reiterate our goal, we wanted to provide users with the best possible song recommendations knowing their current preferences for a certain music genre or album. Using implementations with trees, we first provided an informational genre tree was also provided to help guide users through the different music genres, helping them discover any new genres they may like. The recommendation trees were then constructed with the most popular albums or albums with the most matching descriptors located toward the top, and the recursive nature of the tree allowed users to further explore a line of albums they liked. However, due to limitations in the size of our datasets and the linearity of our sorting algorithm, possible album recommendations may not be the best. For better recommendations, adding more data from a variety of sources, as well as incorporating more parameters to the sorting algorithm can help improve recommendations. While this program may not be perfect,

References

1. IFPI. (2021). Engaging with Music. https://www.ifpi.org/wp-content/uploads/2021/10/IFPI-Engaging-with-Music-report.pdf

2. Bryan O. (2022, March). Rate Your Music: The Top 5,000 Most Popular Albums, Version 1, Retrieved March 8, 2023 from https://www.kaggle.com/datasets/tobennao/rym-top-5000

3. Sonemic (n.d.). RateYourMusic All Music Genres. https://rateyourmusic.com/genres/

4. Plotly Dash. (n.d.). Introduction — Dash for Python Documentation. https://dash.plotly.com/introduction

5. Plotly (n.d.). Tree Plots in Python. https://plotly.com/python/tree-plots/

6. Richardson, L. (2007). Beautiful soup documentation. https://www.crummy.com/software/BeautifulSoup/bs4/doc/

7. Reitz, K. (n.d.). Requests: HTTP for Humans™. https://requests.readthedocs.io/en/latest/

8. Spotify (n.d.). Spotify Web API. https://developer.spotify.com/documentation/web-api/reference/#/