Team Name: AlgoRhythms

Team Members: Luke Gorman (LukeGorman321)
Kevin Concha (kconcha02)
Samuel Polo-Varona

Github Repository: https://github.com/LukeGorman321/AlgoRhythms

Video Demo:
https://youtu.be/l7dVZ30oyco

Vide Demo File:
RecordRecs Pitch for Users.mp4
(https://drive.google.com/file/d/17Ag83KB6QJdG0zIqsv-kbLqADT0Fdoq5/view?usp=share_link)

Proposal

What problem are we trying to solve?

In the modern world everyone tries their best to make the most of every moment they have and in an attention economy our seconds are precious. Therefore, people don't want to waste time searching for what they enjoy and instead want to spend the most time possible enjoying it.

So, in order to avoid spending their precious time searching for music they like, our users can use our website to find songs similar to the ones they already like.

Motivation: Why is this a problem?

Many people struggle with finding songs that might sound similar to their already favorite songs, but sometimes algorithms can miss the mark on their recommendations. We aimed to deliver a product that could tell the consumer what exactly is similar about the music, as well as metrics that they can relate to on the similarity of these songs.

Features implemented

We implemented features such as the ability to use your own CSVs, if you have them of your favorite songs, while also providing a dataset that we can use. Our dataset contains over more than 100,000 points of songs, all with metrics that can be used to compare other songs. You can also filter by common characteristics, apply explicit filtering, and compare the speed of the implementations as they run searches.

Description of data

We are using a public dataset calculated from the spotify web API which can be found here. The different columns include information such as the length of the song, artist names, release date, liveliness, tempo, mode, and popularity.

Tools/Languages/APIs/Libraries used

We used JavaScript, HTML, and CSS for our project. Any libraries of APIs that we did use, are those that are already included within JavaScript, HTML5, and CSS.

Algorithms implemented

We implemented two algorithms for finding the k-nearest-neighbors of a song. One of these used a k-d tree and one used an octree. Both algorithms were similar, they traversed down the tree to where the song would go if it were being added to the tree, then started adding nodes to the list of nearest neighbors as it traversed back up the tree. If there was a different branch of the tree that could have closer nodes than the currently discovered ones, the algorithm would go down that path as well.

Additional Data Structures/Algorithms used

The two data structures we used were k-d trees and octrees. Both of these enable us to effectively create a binary tree to search along multiple dimensions of our data at once.

K-d (k-dimensional) trees are like binary search trees except they alternate which dimension of the data is being compared at each level. For example, the first level may be split based on the vectors' x values, whereas the second is based on the y, the third the z, the fourth the x, and so on.

Octrees are similar to k-d trees, except each node has eight children. Each of the eight children correspond to one direction in a three dimensional space. For instance, child 0 may be (-x, -y, -z) and child 1 may be (+x, -y, -z). When a new node is inserted into the tree, it goes in the slot of the child corresponding to which direction it is from the parent node.

Distribution of Responsibility and Roles: Who did what?

Kevin Concha: Creation and implementation of visual interface with HTML and CSS, and plotting the data into the graph via JavaScript and innerHTML.

Luke Gorman: Import data from CSV, create datastructures, traverse datastructures to find k-nearest-neighbors.

Analysis [Suggested 1.5 Pages]

Any changes the group made after the proposal? The rationale behind the changes.

The biggest change we made was that we were originally planning on using graphs to represent the data, and comparing Dijkstra's algorithm with the Bellman-Ford algorithm to find nearby nodes. Unfortunately, we did not think that it made much sense to represent our data as a graph, so we decided to use search trees instead.

Big O worst case time complexity analysis of the major functions/features you implemented

The KD-Tree and Octrees used in our final product vary in Big-O notation. Our KD-Tree in the average case runs in O(k * log n) time to find the k nearest neighbors out of n nodes. In the worst case, our k-d tree runs the nearest neighbors search in O(k*n) time. This is because it may have to go through every node to find the closest ones in an unlucky situation, and finding the correct spot to input the current node's value into the array of nearest neighbors is O(k).

Our octree also ran in O(k * log n) average time for finding the k nearest neighbors given an octree with n items. But they run in O(log n) for inserting, deletion, and peaking. In the worst case, they run at the same time as the complexities mentioned before, O(k*n). The justification is the same as for the k-d tree.

To measure these differences, we implemented a timing mechanism when searching, and sometimes they achieve the same result or better, but they also flip on being the worst.

Reflection [Suggested 1-1.5 Page]

Overall experience for the project?

The overall experience for the project was good! As always there were some parts that definitely could've gone a little better, and some things that we would've liked to add, but ultimately things have to be either cut or reworked so that the end product can be the best that it can be. We feel like we did a good job developing something out of a need.

Any certain challenges?

Communication is definitely a tough one, especially when members of the team all have different schedules from one another, and its not always a guarantee when you'll be able to catch one another. Regardless, we feel that we did the best week could by utilizing Google Docs, GitHub, Slack, and Zoom to communicate on what we wanted to develop and work on as a team.

If you were to start once again as a group, any changes you would make to the project and/or workflow?

If we had to start over, we feel like we could've flushed out our ideas a little more and if we had all the time in the world, to make something that we feel that people would use on the daily to improve the efficiency of finding new things to enjoy, in this case, music. Even though our project was a little smaller than the big league apps you see today, we feel pretty proud about what we've been able to build, especially for some of us where it's the first time working with a different language than the normal.

Comment on what each of the members learned through this process.

Luke - I learned about importing and reading from a file in javascript as well as how the data structures we use actually work. I now have a much better understanding of the algorithms and what can go wrong when implementing them. Also, I learned that having the data be skewed in a certain way can mess with the results. For instance, if property x is [0, 100] and y is [0, 1] a difference in x is almost always going to overshadow a difference in y, so it is important to normalize our data before using it.

Kevin - I learned a lot about working in all of these languages, HTML, JavaScript, and CSS. Web design actually seemed like something that I could very well get into more, and during this project I felt really inclined to maybe start more of my own projects with these languages for my portfolio. Working in a group also taught me a lot about how important communication between developers is, and how important it is to be on the same page by keep an open line of communication. I also learned to be more careful when handing File IO data, it's very easy to lose it when you least expect it.

References

Minamedez, G., 2020, Spotify Data. GitHub; https://github.com/gabminamedez/spotify-data/blob/master/data.csv

GeeksforGeeks. "Ball Tree and KD Tree Algorithms." *GeeksforGeeks*, 9 Dec. 2023, www.geeksforgeeks.org/machine-learning/ball-tree-and-kd-tree-algorithms/.

---. "Octree | Insertion and Searching." *GeeksforGeeks*, 9 Jan. 2020, www.geeksforgeeks.org/dsa/octree-insertion-and-searching/.

Hristov, Hristo. "Introduction to K-D Trees." *Baeldung on Computer Science*, 4 Mar. 2023, www.baeldung.com/cs/k-d-trees.