SongSeeker

Interactive Music Discovery through Clustering and Data Visualization

Georgia Institute of Technology

Amanda Reedy, Brad Hill, Jake Yang, Min Qin, Samuel Lyon

Summary

The rise of streaming has introduced music recommendation systems that are opaque black boxes that give no further context to the user. To fill this gap, we are introducing **SongSeeker**, which uses clustering, network analysis, and interactive visualization to help users discover new music by identifying songs that are similar to the song they had in mind.

The ways that **SongSeeker** differs from current recommendation engines are through **transparency**, **user interaction**, and **more control over the recommendation system**.

- Interaction with millions of songs
- Force graph visualization providing multiple paths to explore
- Control over song features that are important to them
- Changing the strength of the relationships between songs

SongSeeker solves the black-box problem by allowing the user to specify the song features that are important to them, put more weight on those features, and adjust dynamically. This gives the user more control over the recommendation system.

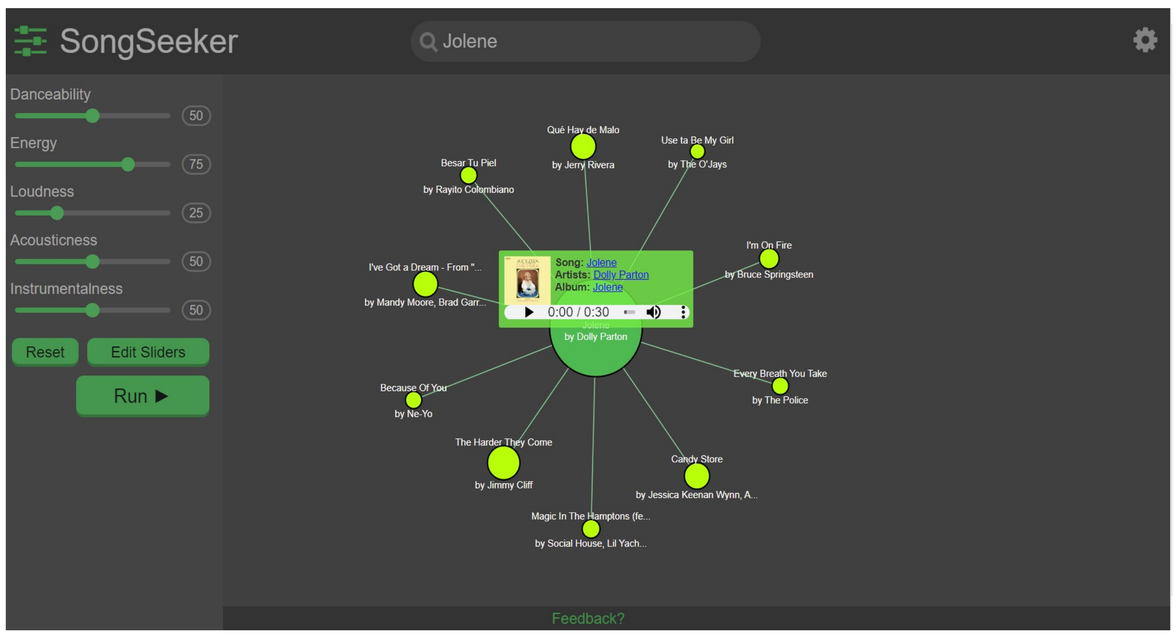


Figure 1. SongSeeker User Interface (UI)

Figure 2. Tools Used

Data Integration & Storage

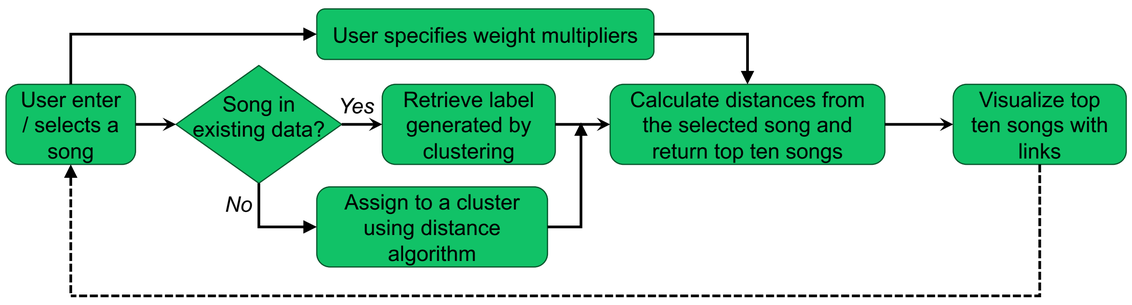
Our training dataset contains **1.8 million songs from Spotify**. Three datasets were originally **downloaded** from Kaggle, each containing a portion of the available fields we needed. We then **combined the datasets** and **filled in missing features by scraping from Spotify directly**. Cleaned training data is stored in Google BigQuery, allowing for easy integration with our application.

Searching a song hits Spotify API in real-time. Data handling and server / client interaction are handled in JavaScript. To stay current and grow our database, new songs information will be added to the BigQuery table, allowing us to constantly add new data to our already large dataset so that our recommendations get better the more our application is used.

Clustering & Song Matching Algorithms

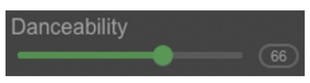
Approaches: To avoid overwhelming users with too many self-defined slicers, we used exploratory data analysis, **clustering, and PCA** to identify top audio features and divide the training dataset into clusters for song matching. SongSeeker determines which cluster the selected song belongs to and then computes the distance from the selected song to every other song in the cluster and select the top ten closest songs. The top ten closest songs are then visualized in the D3 force graph with links to explore more information about the recommended song and artist. The initial clustering was done through Python, but cluster assignment and recommendations are determined through SQL scripts.

This flowchart summarizes **how our approaches work**:



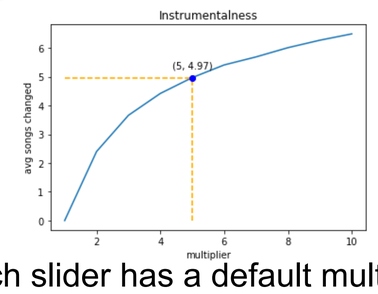
One of SongSeeker's primary innovations is the interactivity: after a user has selected a song, the user can specify the extent to which the various song features take precedence using interactive sliders.

Figure 3. Example Slider



SongSeeker has a slider for each top features, and the user can assign each slider a weight multiplier that feeds the song matching algorithm. This will bias the minimization of Euclidean distance towards the more heavily weighted features, making them more important.

Figure 4. Default Multiplier Value



Each slider has a default multiplier value (preset on the back end) so that it gives roughly 5 changed songs from the baseline.

Visualization & UI Design

The application is built with a front-end UI interface using **HTML, CSS, and JavaScript**, integrated with **D3 force graph visualization**. The song that the user searched for is the center node and the recommended songs are the surrounding nodes. Song nodes that are a stronger recommendation have both a larger radius and are shorter distance from the source song.

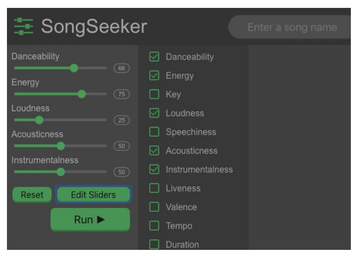


Figure 5. Sidebar

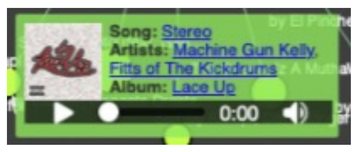


Figure 6. Tooltip

The UI consists of a **collapsible sidebar** with sliders, a menu for user to select additional sliders, a search bar, a 'feedback' button, which leads to a feedback survey popup, and the visualization area with the force graph.

The visualization also includes **tooltips** on node hover with a hyperlink to the Spotify page for the song, artist, and album and a 30 second audio sample so that the user may explore additional information and preview the song if they desire.

Experiments & Results

Experiments were conducted on the range of feature multiplier values. We tried out different multiplier values for each feature. We looked for a range that adequately allows a user to bias for a particular feature while still providing songs that are relevant in the aggregate. The second phase of multiplier determination was using **initial consultation and tests among future users** of the app to get a sense of whether the users found that the sliders changed the songs to a suitable degree.

User feedback was also crucial to **evaluating the success of SongSeeker recommendations** and helped us **identify areas of improvement**. At the time of this writing, we have collected 17 user surveys. On a scale of 1-5, the average rating of the UI/visualization is 3.4 and the average rating of satisfaction with the recommendations is 3.93. The most useful sliders are Acousticness, Danceability, and Energy, while the least useful ones are Loudness, Speechiness, Valence and Duration. Free-form comments acknowledged usefulness of the app and provided recommendations for improvements.

Innovations Compared to Other Methods

1. Song recommendations that do not depend on past listening history
2. Clear, visual feedback on what factors influenced the recommendations

Limitations and Areas of Improvement

1. Improving the run time required to produce recommendations. This can be done by using a local MySQL server on the site to speed up the recommendation process and allow for more open development moving forward, as using BigQuery requires stricter permission to develop.
2. Providing additional guidance regarding the sliders and feature definitions.