SongSeeker: Interactive Music Discovery through Clustering and Data Visualization

Amanda Reedy
Georgia Institute of Technology

Brad Hill
Georgia Institute of Technology

Jake Yang
Georgia Institute of Technology

Min Qin
Georgia Institute of Technology

Samuel Lyon
Georgia Institute of Technology

Abstract

Music consumption has fundamentally changed through time, and with it the way people discover new music. Music recommendations used to require a more personal touch, whether it was from radio disc jockeys knowing their audiences to record store owners talking to their clientele. The rise of streaming has replaced this with a less personal recommendation system that relies largely on what the model *perceives* as the user's taste rather than taking specific cues from the user. These systems are opaque, black boxes that give no further context or information to the user. To fill this gap, we are introducing **SongSeeker**. **SongSeeker** uses clustering and network analysis to identify songs that are similar to the user-provided song. The ways that **SongSeeker** differ from current recommendation engines are through transparency and user interaction. **SongSeeker** allows the user to specify the aspects of the provided song that are important to them, changing the strength of the relationships between songs and potentially even introducing new clusters. This gives the user more choice and more control over the recommendation system.

1 Introduction

With SongSeeker, we want to improve the average music-listener's experience when it comes to discovering new music by allowing the user to have more control over the process and see the impact their decisions make on the recommendation. Current recommendation systems often rely on listening history of both the user and people the model classifies as similar listeners rather than explicit input from the user [13]. When explicit input is used, it is usually a simple like or dislike flag rather than more detailed preferences. Papers [7], [15], [10], and [20] discuss the impact of user-controlled recommendation engines on user experience. [15] and [4] go so far as to have created a similar system to our proposed system with [15] focusing on recommending conference speeches and [4] recommending music using data scraped from social media and other web resources. Both papers show promising results regarding user interaction. These papers show that not only is this a worthwhile undertaking, but that the audience for this project is essentially anyone that uses a music recommendation system. These studies also show that the impact we would have, if successful, is hard to quantify on a macro level. The basis of this idea is that current recommendation systems ignore user satisfaction for "accuracy." Using a combination of user surveys and studies on user interaction is necessary to determine whether we are successful or not. Both of these papers, however, mention that the added complexity from introducing human interaction to the recommendation process can be a deterrent. The risk here, of course, is alienating potential users. We hope to avoid this in SongSeeker by limiting the displayed recommendations at some relationship threshold. Bearing this in mind, paper [18] details the willingness of users to put more effort into using the system and to provide more information if they receive better recommendations. If we are successful, the payoff is significantly increased user interaction and satisfaction and a potential change in the state-of-the-art for music recommendation.

2 Current State

Papers [22], [3], [8], and [1] discuss various methods of extrapolating information from song features which may prove useful in developing our recommendation algorithm and visualizations. [3] and [1] focus on the use of song lyrics, which limits applicability to only songs with vocalizations. [22] and [8] make use of musical features such as tempo, pitch, and melody to identify similarities in music. Additionally, [22] evaluates the results of various clustering algorithms on

the data extracted from the music. These methods or a similar method may be useful in our project in determining similar music however, the complexities of the algorithms and the availability of needed data may limit the applicability of these methods to our project.

Paper [12] describes phases of music discovery over the last 20 years and includes several visualization examples from each phase. Potential shortcomings of these visualizations are that they may be too simple to be satisfactory for the project, they are too inflexible to create interaction with, or they do not incorporate user interaction to the extent we wish. However, they may be useful to incorporate into our project in some regard or to at least serve as inspiration.

[14], [2], and [5] discuss some of the problems associated with algorithm generated content recommendations. Specifically, [14] discusses the cold start problem i.e. how to handle recommendations for new items and new users to the system and [2] and [5] discuss the negative effect that recommendation algorithms can have on the diversity of music consumption and the potential negative impacts of that reduced diversity to user satisfaction and user retention.

An understanding of the limitations of current research helps to focus attention on the problems of music discovery interfaces still needing to be addressed.

3 Methodology

With the preceding sections in mind, SongSeeker endeavors to create a music discovery interface that builds upon prior work in the field, making music recommendations more granular and customizable. Rather than simply suggesting new songs based on methods stated in [11], SongSeeker gives the user control over the recommendations.

3.1 Data

Spotify's Application Programming Interface (API)[1] is rather robust, offering both audio features for tracks and metadata like track number, disc number, release date, etc. Our original three datasets from Kaggle[2,3,4] each contained a portion of the available fields we needed, but not all of them. We used R[17] to combine the three datasets, and then looped through the dataset to fill in missing features using the spotifyr[19] package in R. This created a roughly 1.8 million song dataset to build our clustering and song matching algorithm and provide a source for song recommendations.

We are storing the data in a table in Google BigQuery[5], as BigQuery's API allows for easy integration with our application. Moving forward, all songs that are added to the database will be directly added to the BigQuery table.

3.2 K-means Clustering of Spotify Songs

Motivation of the Clustering Step: Our 1.8 million song data set contains a long list of audio features for each track. To avoid overwhelming users with too many self-defined slicers, we used this clustering step to explore the data and understand features' correlations with each other. The goal is to develop a clustering algorithm that would serve as the basis for our similarity score and divide the large data set into manageable clusters for visualization.

Tools Used: pandas and NumPy [6] were used for importing data and pre-processing; matplotlib[9], seaborn [21], and plotly were used for plotting; and sklearn [16] was used to build the clustering model.

Exploratory Data Analysis: We first loaded the 1.8 million song dataset and checked for unique values. Interestingly, even having individual IDs for each song, there are repeated track names, which could be due to different versions from the same artist, covers, or even repeated song names. Each track has a series of features. As a starting point, we used the following 15 features provided by Spotify: explicit, danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, duration_ms, time_signature, and year.

Pre-processing: We standardized the values of all feature so that features are placed on the same scale. This allows all features to be treated equally by the model. We accomplished this by using sklearn's MinMaxScaler.

Clustering Algorithm 1 - Clustering Using 15 Original Feature: For KMeans clustering, we have to determine the number of clusters to divide the data into. One of the most popular methods to do so is the Elbow Method. In the Elbow method, we iterated over

[1] https://developer.spotify.com/documentation/

[2] https://www.kaggle.com/rodolfofigueroa/spotify-12m-songs

[3] https://www.kaggle.com/luckey01/test-data-set

[4] https://www.kaggle.com/ektanegi/spotifydata-19212020

[5] https://cloud.google.com/bigquery

some clusters (K) from 1 – 10. For each value of K, we calculated the Within-Cluster Sum of Square (WCSS). As the number of clusters increases, the WCSS value will start to decrease. WCSS value is largest when K = 1. When we analyze the plot, we can see that the graph will rapidly change at a point, thus creating an elbow shape. From this point, the plot starts to move almost parallel to the X-axis. The K value corresponding to this point is the optimal K value or an optimal number of clusters. Figure 1 displays the elbow plot based on our data set. The point at which the elbow shape is created is 4. That is, our optimal number of clusters is 4. Next, we trained the model on the data set with 4 clusters. We used the KMeans function from the sklearn package to accomplish this. Below are the number of songs in each of our four clusters: Cluster 0: 179443 songs, Cluster 1: 456054 songs, Cluster 2: 462436 songs, and Cluster 3: 661590 songs.

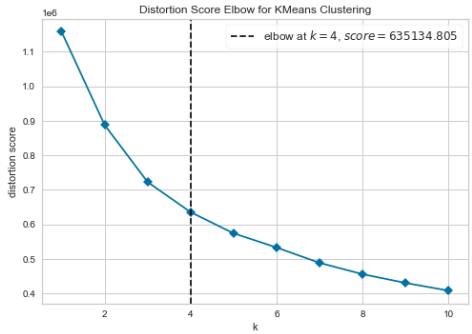

Figure 1: K-Means Clustering Elbow Plot

Clustering Algorithm 2 - Clustering Using Top Components from Principal Component Analysis (PCA): In Principal Component Analysis (PCA), the goal is to find the best possible subspace that explains the most variance. Figure 2 illustrates how much variance is explained by each of the components. Using 0.8 as the threshold explained variance, our PCA analysis picked five components. We then looked at the elbow point of the WCSS curve using those top 5 components from PCA. Running the elbow method has again determined that 4 is the ideal number of clusters or distinct groupings to separate the tracks in our data set. Lastly, we implemented k-means clustering using the top 5 components and an optimal number of 4 clusters. Below are the number of songs in each of our four clusters: Cluster 0: 179172 songs, Cluster 1: 534694 songs, Cluster 2: 606984 songs, and Cluster 3: 438673 songs. We also visualized the clusters on 2D planes to

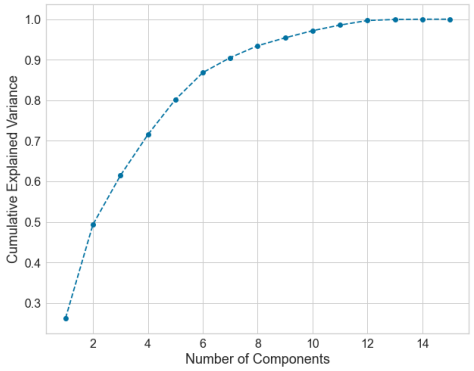

Figure 2: PCA Explained Variance

ensure that the clusters separated signals based on the sources of the signals encapsulated by the top PCA components. Figure 3 displays the 2D cluster visualization.

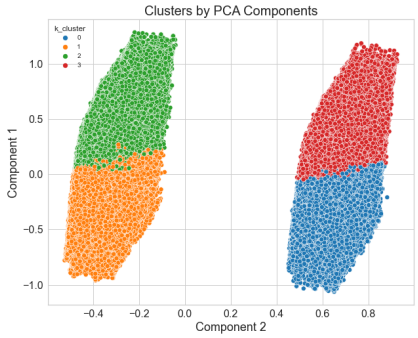

Figure 3: 2D Principal Component Clusters

Visualizing Features of Tracks in Each Cluster: Although there is some noise in the clusters, the plot indicates four distinct groups of tracks. To infer meaning from the transformed components, we examined the raw values of the audio features of the tracks after they have been clustered. Radar charts let us observe each cluster rapidly. A radar trace has been plotted in figure 4 for the average audio feature values in each cluster after normalizing the entire data frame.

Conclusions of Clustering Analysis: Each cluster's trace shapes reveal what type of tracks we can expect. In summary, Clusters 2 and 3 sound more acoustic and feature more instrumental tracks, whereas Clusters 0 and 1 are more energetic with higher danceability. The

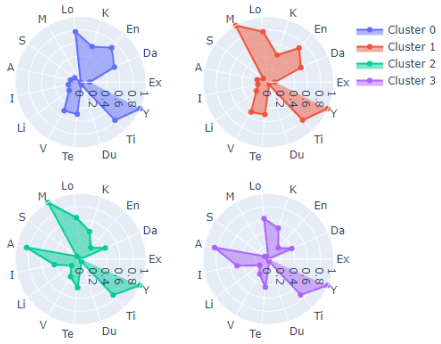

Figure 4: Cluster Radar Charts

main differences between Clusters 2 and 3 or Cluster 0 and 1 are the mode and key.

Based on the clustering analysis, danceability (DA), energy (En), loudness (Lo), acousticness (A), and instrumentalness (I) are candidate features for our similarity score and visualization sliders. Those are the most meaningful features returned by our PCA and clustering analysis. The other features in our data set did not show strong enough sources of signal on which to split the data into distinct groupings.

3.3 Song Matching Algorithm

Now that we have assigned cluster labels to the approximately 1.8 million songs in our standardized training set, SongSeeker will predict which cluster the user-selected song belongs to. By doing this, rather than querying the entire dataset for each song, we'll reduce processing time, providing a more enjoyable and streamlined process for the end user.

Once the song is assigned to its cluster, SongSeeker computes the Euclidean distance from the selected song to every other song in the cluster and selects the top 10 closest songs. We use the Euclidean distance as opposed to a metric like the cosine-similarity score, because the magnitudes of the song vectors do matter in this case; the cosine similarity score is better in instances where the magnitude is not important. The top ten closest songs are then visualized using a force graph, including links to explore more information about the recommended song, album, and artist.

One of SongSeeker's primary innovations is the interactivity: after a user has selected a song, he can also determine the extent to which the various song features take precedence through the use of interactive sliders. SongSeeker initially provides a slider for each of the top 5 features (danceability, energy, loudness, acousticness, and instrumentalness), and allows the user to select additional features to adjust if they feel so inclined.

The user will then assign each slider a value from 0-100. This 0-100 range is largely designed around user experience and intuition and not around the exact calculated weights of each feature. Table 1 below details the exact maximum multipliers, while each feature's minimum multiplier is 1. This means that a level of 100 for a given slider will equate to that feature's maximum multiplier value, while a level of 0 will set that feature's multiplier value to 1.

This does result in inconsistent ramp ups to a feature's maximum multiplier value, but as we detail in the Experiments and Evaluation below, the main consideration in these sliders is number of songs changed, so this movement actually helps in achieving that goal. The formula for converting slider value to feature multiplier value is:

$$FMV = 1 + (SV * (1 - MMV^{-1})(0.01MMV))$$

$$FMV = FeatureMultiplierValue$$

$$MMV = MaximumMultiplierValue$$

$$SV = SliderValue$$

This equation allows us to scale the slider value to the appropriate feature multiplier value. A slider value of 0 will always return 1, whereas a slider value of 100 will always return the maximum multiplier value. Due to the different scaling dependent on the maximum multiplier value, the halfway point is different based on the MMV as well. For instance, the halfway point for an MMV of 10 is a slider value of 44, while the halfway point for an MMV of 4 is 33.

These values will determine that feature's weight when computing the distances, with larger weights being assigned to features that the user has deemed more valuable. The larger the weight for a particular feature, the greater the average difference between the selected song's feature value and the feature values of the other songs in the cluster. This will bias the minimization of Euclidean distance towards the more heavily weighted features, making them more important.

A user can change as many of the sliders as he wishes, and the final output will not be computed until he presses the "Run" button. This will generate the aforementioned graph, complete with top ten song recommendations based on the sliders' values.

3.4 Data Integration and Visualization

Initially, our plan included using R as the main framework underlying our application, using R's shiny package to handle the client/server interactions, the spotifyr package to interact with Spotify's API, and the r2d3 package to integrate the D3.js[6] visualization. After building out this framework, we found that keeping all of the data handling and server/client interaction in Javascript would allow for a more seamless user experience and a more maintainable code base. Client/server interaction is handled with node.js[7], while the Spotify API interaction uses the spotify-web-api-node wrapper[8]. The search feature uses selectize.js [9] and jQuery[10] to allow the use of a remote source (Spotify's API) to dynamically change the search options provided.

Due to the amount of data involved, we use Google BigQuery as our database and pass all of our processing to BigQuery as well. The initial clustering step was done through Python, but cluster assignment and recommendations are determined through SQL scripts. All songs searched in the application are added to the database (if not already included) and assigned a cluster, which allows them to be recommended in the future. A consideration for further improvement is 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 permissioning to develop.

To stay current and grow our database, each song selected will be added to our song database. This means that we are constantly adding new data to our already large dataset, allowing our recommendations to get better the more our application is used.

The visualization portion of SongSeeker is a graph built using D3 force layout. The song that the user searched for is the center node and the recommended songs are the surrounding nodes. The visualization displays the strength of the recommendation in the node radii and length of the edges. Song nodes that are a stronger recommendation have a larger radius and are shorter distance from the song that the user selected.

[6]https://d3js.org/

[7]https://nodejs.org/en/

[8]https://github.com/thelinmichael/spotify-web-api-node

[9]https://github.com/selectize/selectize.js/

[10]https://jquery.com/

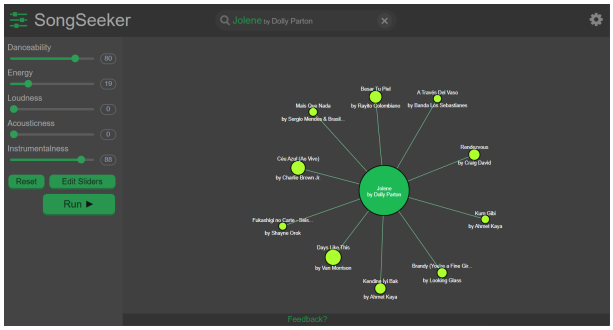

Figure 5: SongSeeker User Interface

The visualization also includes tooltips when hovering over the nodes that provide a hyperlink to the Spotify page for the song, artist, and album for the song and a 30 second audio sample, if one is available from Spotify, so that the user may explore additional information about the recommendation and listen to an audio sample if they desire. Figure 5 displays the UI and visualization of SongSeeker.

We have also enabled a feedback form to capture user feedback and send results to a database in BigQuery for analysis.

4 Experiments and Evaluation

What constitutes a "good" song recommendation is inherently subjective, and as such it is quite difficult to choose appropriate multiplier ranges for each of the sliders. Some relevant questions one should think about include: how big should the multipliers be? Should the different features have different multipliers? How should the multipliers be determined? To find the answers to these questions, we decided on a combination of two approaches.

In order to determine the best multiplier values, we endeavored to find out, on average, how many recommended songs change from the baseline (multiplier value of 1), for different multiplier values (called "MVs" hereafter), over a large number of trials. However, before we could do this, we needed to determine an adequate range of MVs. With this in mind, we tried out different MVs (as little as 1 and as big as 100), for each feature, over 10 randomly chosen sample songs. This provided an idea of the MVs necessary to change a specified number of songs for each feature. However, this information is not very useful without user input on what constitutes an adequate number of songs changed (called "NoSC" hereafter) for each feature's maximum multiplier value.

Initial consultation among potential future users of the app provided the agreed upon opinion that 5 songs should change, on average, for a single slider moved to the maximum value. Among our trials, we noticed that, for each feature, it generally took a multiplier of 10 or less to change 5 songs on average, with most of the multipliers being closer to 5. After a multiplier of 10, the NoSC generally leveled off, requiring larger and larger multipliers for incremental improvements in average NoSC. With this initial threshold in mind, we randomly selected 100 songs and ran the trial for each song, for each feature, with ten different MVs (1-10). We then found the rough multiplier, per each feature, that would result in 5 songs changed on average. Figure 6 illustrates this in graph format for the "Instrumentalness" feature, and Table 1 displays a table of the maximum multiplier values by feature.

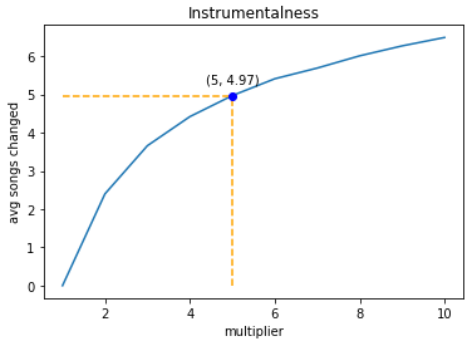

Figure 6: Instrumentalness Multiplier Evaluation

Feature	Maximum Multiplier Value
Danceability	4
Energy	4
Key	4
Loudness	4
Speechiness	5
Acousticness	10
Instrumentalness	5
Liveness	10
Valence	5
Tempo	4
Duration	4
Popularity	10
Time Signature	5
Year	4

Table 1: Feature Multiplier Values

User feedback is crucial to evaluating the success of the recommendations provided by SongSeeker and to identify areas of improvement. To that end, we have implemented a User Feedback form that asks the users to rate from 1-5 the User Interface (UI) and visualization (Is the tool easy to use/understand?) and their satisfaction with the recommendations, identify the features that were most useful for finding good recommendations, and identify the features least useful for finding good recommendations. We've also included an area for the user to provide free-form feedback. At the time of this writing, we have collected **17** user surveys. Overall, 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** and the least useful are **Loudness, Speechiness, Valence, and Duration**. Free-form comments acknwoledge the usefulness of the SongSeeker app and provide recommendations for improving the application such as improving the run time required to produce recommendations and providing additional user guidance regarding the sliders and definitions of features.

5 Conclusions and Discussion

The preliminary evaluation of the average rating of the recommendations provided by the SongSeeker tool is 3.93 out of 5 indicating that the users are generally satisfied with the recommendations that SongSeeker has provided. Areas of improvement already identified by the developers of SongSeeker this project are implementing a local MySQL database to improve the run time of the app, providing definition of the features, and providing a short user guide for with the application.

With additional time and a larger sample of user surveys, it's likely that additional areas of improvement in both the UI and visualization will be identified. Using the data regarding the most and least useful sliders could result in refining the recommendation algorithm to adjust the weighting of those sliders and subsequent A/B testing to evaluate the results of the algorithm adjustments. Additionally, as our song database grows, our clustering model will need to be periodically evaluated and refreshed to incorporate the new song data.

Our team has contributed a similar amount of effort towards this project.

References

[1] Akella, R., and Moh, T.-S. Mood classification with lyrics and convnets. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)* (2019), IEEE, pp. 511–514.

[2] Anderson, A., Maystre, L., Anderson, I., Mehrotra, R., and Lalmas, M. Algorithmic effects on the diversity of consumption on spotify. In *Proceedings of The Web Conference 2020* (2020), pp. 2155–2165.

[3] Barkwell, K. E., Cuzzocrea, A., Leung, C. K., Ocran, A. A., Sanderson, J. M., Stewart, J. A., and Wodi, B. H. Big data visualisation and visual analytics for music data mining. In *2018 22nd International Conference Information Visualisation (IV)* (2018), IEEE, pp. 235–240.

[4] Bostandjiev, S., O'Donovan, J., and Höllerer, T. Tasteweights: a visual interactive hybrid recommender system. In *Proceedings of the sixth ACM conference on Recommender systems* (2012), pp. 35–42.

[5] Celma, Ò. The long tail in recommender systems. In *Music Recommendation and Discovery*. Springer, 2010, pp. 87–107.

[6] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature 585*, 7825 (Sept. 2020), 357–362.

[7] He, C., Parra, D., and Verbert, K. Interactive recommender systems: A survey of the state of the art and future research challenges and opportunities. *Expert Systems with Applications 56* (2016), 9–27.

[8] Hsu, J.-L., Liu, C.-C., and Chen, A. L. Discovering nontrivial repeating patterns in music data. *IEEE Transactions on multimedia 3*, 3 (2001), 311–325.

[9] Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering 9*, 3 (2007), 90–95.

[10] Jin, Y., Htun, N. N., Tintarev, N., and Verbert, K. Contextplay: Evaluating user control for context-aware music recommendation. In *Proceedings of the 27th ACM Conference on User Modeling, Adaptation and Personalization* (2019), pp. 294–302.

[11] Knees, P., and Schedl, M. A survey of music similarity and recommendation from music context data. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 10* (2013), 1 – 21.

[12] Knees, P., Schedl, M., and Goto, M. Intelligent user interfaces for music discovery. *Transactions of the International Society for Music Information Retrieval 3*, 1 (2020), 165–179.

[13] Lex, E., Kowald, D., and Schedl, M. Modeling popularity and temporal drift of music genre preferences. *Transactions of the International Society for Music Information Retrieval 3*, 1 (2020), 17–30.

[14] Lika, B., Kolomvatsos, K., and Hadjiefthymiades, S. Facing the cold start problem in recommender systems. *Expert Systems with Applications 41*, 4 (2014), 2065–2073.

[15] Parra, D., Brusilovsky, P., and Trattner, C. See what you want to see: visual user-driven approach for hybrid recommendation. In *Proceedings of the 19th international conference on Intelligent User Interfaces* (2014), pp. 235–240.

[16] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[17] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017.

[18] Swearingen, K., and Sinha, R. Beyond algorithms: An hci perspective on recommender systems. In *ACM SIGIR 2001 workshop on recommender systems* (2001), vol. 13, Citeseer, pp. 1–11.

[19] Thompson, C., Antal, D., Parry, J., Phipps, D., and Wolff, T. *spotifyr: R Wrapper for the 'Spotify' Web API*, 2021. R package version 2.2.1.

[20] Verbert, K., Parra, D., Brusilovsky, P., and Duval, E. Visualizing recommendations to support exploration, transparency and controllability. In *Proceedings of the 2013 international conference on Intelligent user interfaces* (2013), pp. 351–362.

[21] Waskom, M. L. seaborn: statistical data visualization. *Journal of Open Source Software 6*, 60 (2021), 3021.

[22] Xu, Y., and Xu, S. A clustering analysis method for massive music data. *Modern Electronic Technology 5*, 1 (2021), 24–30.