`set(list[])`: Analytics for Live Music Setlists

Will Diepholz, Ryan Woods, Steve Eckhart, Mahesh Imaduwa, Kyle Dalal, Tran Ton
Georgia Institute of Technology
CSE 6242
{wdiepholz3, rwoods32, seckhart3, pramindas, kyledalal, tton8}@gatech.edu

Motivation

One major challenge for live-performing musicians is choosing songs to perform while on stage, known as a setlist. Currently, musicians put together their setlist based on gut feel, which ignores the wealth of data collected by various services and music platforms. The motivation behind this project is to easily enable musicians to incorporate big data into live set decision-making by providing a tool that compares different artists and their setlists, creating impactful visualizations on similar types of songs using low-level audio features. Ultimately, our team strove to empower musicians to make the best decisions about their performance, eliminating the guesswork of creating a setlist.

Problem Definition

Our tool, `set(list[])`, strives to solve three fundamental problems in live setlist music:

1. Musicians can be difficult to directly compare, often transcending genres. **How can we cluster similar artists?**
2. Setlists are widely varied between artists and there is useful information by looking at similar artists' setlist. **How similar are all live-music setlists?**
3. Features of songs (loudness, tempo, danceability) are highly dimensional and difficult to classify. **How can we effectively visualize audio features?**

Survey

State of the Art algorithms regarding music analysis can be broken into one of three categories: recommendation, classification, and visualization.

Recommendation

With a majority of music consumption happening online, many machine learning models have developed to take advantage of user data and create recommendation systems based on user historical data. For example, a multinomial Hidden Markov Model (M-HMM) has shown success in personalizing of music recommendation, but requires historical user data that may not be directly accessible through Spotify's platform [1]. Additionally, a two-dimensional taxonomy has been used for mood detection in Western Classical music, which can be used for "mood-based" playlists [2] [3]. A survey of recommendation algorithms reveals a combination of low-level features and user historical data, can combine to make a successful recommendation system [4].

There exist many machine learning algorithms to create playlists such as LSTMs, RNN, and logarithmic regressions, which utilize bootstrapping aggregate to become more robust over time [5, 6, 7]. FeedRec, a tool that optimizes long-term user engagement, is an example of a reinforcement-learning approach to improve playlist recommendations [8]. Many music recommendation systems come from industry, where they can be directly utilized on massive datasets to recommend new music [9, 10].

Classification

There are an abundance of novel ways to take an input audio song and extract features; there are many algorithms operating on these features which seek to classify songs into different genres [11]. A wide range of quantitative metrics such as acousticness, danceability, energy, instrumentalness, liveness, loudness, speechiness, etc. - have been shown to be reliable classifiers of different types of music [12, 13].

Cover song recognition has been attempted by utilizing elementary rejectors of duration, tempo, and harmonic content of the song. While this technique is efficient in a large dataset, its rejectors may improve because the tempo, duration, and harmonic of even cover songs can vary drastically from the original song [14]. The Music Emotion Recognition (MER), a project aimed at classifying the emotional content of a song, showed moderate success by utilizing principal component analysis and support vector classifiers to map to an Arousal-Valence mapping [15].

The detection of repeating patterns in music data through a correlative matrix data structure, coined as an RP-tree data structure, shows promise on operating on MIDI data for song classification [16]. Additionally, rhythmic pattern similarity, pitch interval similarity, and timbral similarity have all shown success in clus-

tering songs together [17]. These are distance-based similarity functions that have been employed in developing clustering algorithms in music data analysis [18, 19]. Lastly, the algorithm *MaxLogHash*, is a successful and memory efficient way to classify songs at a high level [20].

Visualization

Visualizing music has always been a challenge since its inception; being able to translate an orchestra onto paper is a uniquely difficult and human task. Nonetheless, there have been novel ways of representing audio with visualizations. For example, a *representation learning framework* incorporates both node and edge graph properties which display similar song information in order to visualize the relationships between different artists and songs [21]. Paul Lamere's *Search Inside the Music*, offers techniques used to effectively visualize related songs and albums such as autocorrelation, spectograph analysis, and acoustic similarity [22].

Proposed Methods

Intuition

Currently, state-of-the-art algorithms all fall short of the problem of helping live musicians find similar musicians and setlist features for them. Recommendation systems all operate on a single user history and predicting one or more songs or artists that they would enjoy next, but are not generalizable enough to perform artist clustering. Classification systems seek to identify a genre of music but fail to link that information to other songs and artists. One key element that current algorithms fail to include that is key to building a setlist is the sequential element of song selection. Our method combines the sequential song component through setlist FM data with the audio features per track from Spotify to visualize artist setlists split into quarters.

Additionally, state-of-the-art algorithms provide breakthrough intelligence about music recommendations but ultimately fail if they are unable to deliver content to the user, the musician. One of the main purposes of `set(list[])` is to democratize the access to setlist information so that any musician will be able to make informed decisions on setlist choice regardless of technical background.

Approach

Our approach can broken down into three steps:

1. **How can we get data?** - setlist.fm, Spotify, and Python APIs for data gathering and cleaning
2. **How can we cluster the data?** - Dimensionality Reduction and Clustering for similar artist/setlist modeling
3. **How can we show the results?** - matplotlib, R, and R Shiny for data visualization and application hosting

Data Gathering/Cleaning

We began with a seed artist list of 1200 artists generated from influential artists of major genres such as pop, classic rock, and hip-hop and similar artists according to Spotify. For each artist, we pulled data from every setlist available from a general Setlist.FM search of the artist using their free API. We kept the following setlist features for a setlist with n songs: artist, date, venue, city, state, country, latitude, longitude, encore, song 0, ..., song n.

The major challenge with the Setlist.FM API is its restrictions on the number of requests. We were only allowed 2 requests per second, and a total of 1440 requests per day. Moreover, the API only allowed 1 page of results per request with a maximum number of results per page capped at 25. Major artists often had 30 or more pages of results. This proved to be a significant bottleneck and prevented us from pulling setlist data dynamically.

In addition, Spotify's API was queried in order to collect song information for the 1200 artists. For each artist, we pulled all the songs and their low-level audio features such as **danceability, energy, key, loudness, mode, speechiness, acoustic, instrumentalness, liveness, valence, tempo, duration, and time signature** from Spotify. Detailed descriptions for each of the audio features can be found in the README.txt, or at the Spotify API Reference website. Due to Spotify Web API rate limiting, it took about 12 hours to finish pulling all the songs and audio features for the 1200 artists.

The dataset obtained from Spotify was then used for two purposes. First, the data was aggregated to get summary statistics for each artist such as min, max, average, and median of each audio feature. The summary statistics would then be used for Similar Artist modeling. Second, the artists' song and audio feature data was joined with the setlist data for setlist analytics. Most of the data transforming and integration was implemented using Python pandas library because of its rich features and flexibility. However, due to the size of the data, there was an issue with insufficient memory usage. This issued was later mitigated by using different methods of reducing memory usage such as processing data in chunk, filtering out unimportant columns, and selecting appropriate data types for columns.

Having both an audio feature dataset by artist and a setlist dataset, it was time to join the two datasets. For a setlist datapoint with songs 0-n, we took each song and added a column for each audio feature. Because our audio feature dataset provided 13 audio features, we expanded each setlist datapoint by 13 columns for each song in the setlist. For example, a datapoint with song 0 was expanded to song 0 acousticness, song 0 valence, and so on added to its columns. This ultimately ended in a dataset with roughly 260,000 rows and 1400 columns. In order to build a clustering model out of this joined dataset, the setlist lengths had to be normalized - how else could we compare a 26 song Guns N' Roses

setlist with an 8 song Miley Cyrus setlist? We chose to normalize setlists by quarters as this gave enough granularity to see variation between quarters but also a large enough split for realistic computing times. To better understand the splits, imagine a setlist with 16 songs and 13 audio features. The first setlist quarter Q_0 would then be calculated like the following for each audio feature f_i:

$$Q_{0f_i} = mean(Song_{0f_i}, Song_{1f_i}, Song_{2f_i}, Song_{3f_i})$$

For time signature, mode (major/minor), and key we used the mode instead of the mean. This resulted in 52 columns for the 13 audio features in each of the 4 setlist quarters.

We ended up keeping only setlists with a minimum of 6 songs having no more than 2 songs with missing audio feature data. Under this filtering we ended up losing roughly half of our setlist data, but felt it was necessary to avoid excessive imputation and inaccurate depictions of artists' setlists. Given more time, we could explore better methods for audio feature retrieval and filtering to retain more setlist data.

Similar Artist/Setlist Modeling

The first part of the analysis involved exploring clustering of related artists based on the data from Spotify's API. In order to select artists similar to the user's selected artist, we first considered using Spotify's related artists feature. However, Spotify selects related artists by the number of users who listen to one artist and also listen to a second artist. Our model was built to take statistical information about the songs the artist had recorded on albums and compare it to the 1209 artists we sampled initially. We started with a list of 100 artists who had the most entries on any of the music charts. We pulled Spotify's related artists list and pulled those artists' songs, too. Using sci-kit learn, we first used principle component analysis to reduce the data space from 52 parameters to 4.

We then built a K-means model. We found that a K-means model with k=29 gave consistently large enough clusters to provide the set list model with 10-20 similar artists. Because of the limitations in generating setlist.fm data, we currently limit the model to only return artists for which we have setlist.fm data. When the K-means model predicts a cluster with more than 20 entries, we selected the 20 artists closest to the artist of interest in the PCA space.

The code for pulling the artist music data is in the Spotipyer module. The code to create the PCA and K-means models are in the Band Model band.ipynb file.

The second part of the analysis focused on the differences between all setlists as a whole. In order to segment the data, K-Means clustering was applied to the aggregated low-level acoustic features from setlist.fm and Spotify. The purpose was to see if there were clear and separable differences in setlists. Due to the high dimensionality and length of the data, `sci-kit learn`'s `MiniBatch K-means` was utilized, which operates on batches of data sequentially.

In order to select optimal cluster number, a silhouette and elbow plot was generated for different number of means to visualize the separation between clusters. The plot is shown in the results section, below. Ultimately, setlists were segmented into eleven different clusters.

The last part of the analysis focused on encore prediction. One of the responses from setlist.fm is a flag whether the artist performed an encore at the end of the performance. In order to explore the topic of encores as a response variable, we created a Decision Tree classifier based on the setlist quartile data. A decision tree classifier was selected over a random forest due to the higher level of introspectability on the splits between the data, which enables a "recommendation" system based on training data.

In order to maximize performance from the classifier, the acoustic parameters were normalized along each axis and randomly shuffled to create a test/train split of 70/30, respectively. Utilizing `sci-kit learn`, a Grid Search was carried out using various hyperparameters to optimize the model for accuracy on the training set.

Once the classifier was trained properly on the training set, it could be used to predict the probability that any novel setlist or a user-created setlist would receive an encore. Unfortunately, due to time restrictions, this feature was not fully implemented in the R Shiny application. The model, however, is available in `encore_predictor.py`.

For visualization purposes, the classifier was trained on PCA and t-SNE reduced data, in order to reduce the number of dimensions to 2. Both PCA and t-SNE are both dimensionality reduction techniques, with slightly different techniques to arrive at a desired number of components - or dimensions. PCA seeks to maximize variance between each principle component, whereas t-SNE seeks to preserve relationships between data points while probabilistically distributing the data among a desired number of dimensions. Results are shown for both PCA and t-SNE in the experiment section.

Data Visualization and Application Hosting

R Shiny proved to be a useful way to deliver our results as it is accessible and we can create interactive visualizations without having to manage any hosting, HTML, or CSS ourselves. The user inputs a given artist, and our similar artist model generates a list of 20 similar artists in the back end that exist within our setlist dataset. This similar artist list provides the vehicle for visualization through the rest of the interface.

SetList Builder Shiny Application (minimal app) is made of server.R to handle back-end business logic, ui.R for user interface, and dataprocessor.R for data loading and prepossessing. The application is available at shinyapps.io server and can be accessed via https://mimad.shinyapps.io/setlist/. The Git repository for the minimal application is located at https://github.gatech.edu/pmwi3/setlist_shiny_app

Due to the limitation of shinyapps.io server, the current version of the Shiny Builder Application is unable to include the clustering algorithm running in the server. It may be able to run in RStudio IDE. Git repository: https://github.gatech.edu/pmwi3/setlist_analyst_app.

Ridge plots are shown for the similar artists based on the user input. They show the four setlist quarters with the user selected similar artist setlist audio feature distribution data for each quarter. It provides a succinct method of comparing different setlist quarters. For instance, the first quarter might have a tight energy distribution around 0.8 indicating the artists frequently start their live sets really explosive and high energy. Analyzing the rest of the quarter distributions can show how the energy evolves over the setlist.

Experiments

Testbed

Primarily, our experiments aimed to answer questions around the usefulness and validity of the clustering model that was developed, as well as the ease-of-use of our user interface. Specifically:

1. Are clustering artists with audio features representative of user listening trends?
2. Are all setlists varied enough between non-similar artists?
3. Can we effectively predict reduce dimensionality of the setlist? If so, can we use that to predict encore likelihood?
4. Does visualizing the attributes of similar artists help chose type of song to play?

Results

1: In order to explore the idea of setlist variety, clustering between audio features of setlists *without* was performed. We wanted to see whether setlists from totally different genres and artists would cluster together based on the audio features of their performances; for example, perhaps country and death metal have very similar live music setlists based on danceability; but drastically different based on tempo.

In order to perform clustering on the setlist data, we aggregated the song data on each setlist into quarters. This way we could visually see changes in setlist periods, as well as control for setlist length. With this data, we tried using three different types of modeling approaches: Density-based Spatial Clustering of Applications with Noise (DBSCAN), hierarchical clustering (Ward) and K-Means Clustering.

Our first choice was using DBSCAN, which is a widely used clustering algorithm that segments points together based on density; leaving outliers that are far away from other neighbors as noise. This has advantages over other clustering methods as it enables outliers to not affect cluster centers. In our experimentation, `sci-kit learn`'s DBSCAN implementation was utilized. We used a K-Nearest Neighbor fit for an elbow plot that helped us figure out the proper eps for our DBSCAN model, which turned out to be 17000. If we moved the eps higher, the outcome would result in a large number points without a cluster. Min sample was another parameter used, to make sure we had a large enough cluster size, and to limit the number of clusters. Overall, due to the density of the features, DBSCAN resulted in either a vast majority of setlists being linked together or more than 30% being designated as outliers. Another issue we ran into is only a few clusters would hold a majority of the setlists.

Our second choice for a model was to use a hierarchical clustering model. These types of models start with cluster counts of n data points, then the algorithm looks for the two closest points and clusters them. The process continues until all points are clustered. This methodology would have allowed us to use a dendrogram as a part of the visualization process. For this method, we used the AgglomerativeClustering model from sklearn with the ward linkage. While the ward method doesn't create less optimal clusters than other linkage, it is far less computationally intensive. However, the dataset we use was too big for our computers. Even with reduced components, resulting from PCA analysis, Agglomerative Clustering took 58 GB of memory to complete the task. In the future, we would take multiple steps to deal with the computational challenges this model presents. We could either build an expensive computer that would have more than enough memory for future use, build methodology to process each data in chunks, or even down casting the datatype.

Our final and least desirable model is a K-means clusterer was applied to the aggregated setlist data. It was the least desirable for us because it's a method that has been used on audio features many times before. Before running our model, we had to find the correct number of clusters. After iterating through different number of means, the optimal was found to be 11 clusters, shown in the elbow plot below. We used the sklearn Mini Batch K Means model with the 11 cluster parameter to create our final cluster. (Fig. 1).

2: To explore effectiveness of our similar artist clustering algorithm, we directly compared with Spotify's "similar artists" functionality. Spotify utilizes a combination of user recommendations, genre trends, and many secret recipes; our clustering technique operates only on audio features from songs the artist had. Testing the original seed bands and comparing our model's output artists with Spotify's related artists, the largest overlap was eight of 20 artists. We had established that we would accept the new model if at least 10% of artists had different related artists than Spotify. In our actual testing all of the artists had different artists, usually the full list our model selected.

3: To create an encore-predictor, we tested both a Decision Tree and Random Forest model using a 70/30 train/test split. Upon testing, the Decision tree reached an average accuracy of 85% and the Random Forest

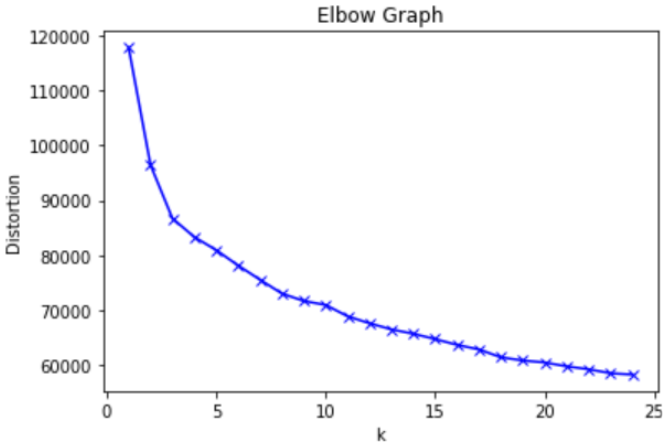

Figure 1: Cluster Count Elbow Chart `1`

reached an average accuracy of 89% over 10 trials using random seeds.

In order to reduce the computation required for the application, dimensionality reduction was performed on the setlist data to reduce it to just two dimensions. This enabled both easier load on the application for processing a user-inputted setlist as well as an opportunity to visualize the decision surface for each classifier in a 2-dimensional way. Both PCA as well as t-SNE were utilized to reduce 52 columns of features into just 2, which would be used for classification.

Principle Component Analysis was carried out on a normalized dataset, with 2 specified components. The first 2 principle components had variance ratios of 0.21 and 0.09, respectively. Training a Decision tree classifier on *just two components* resulted in an accuracy of 76%, with the decision boundary shown in Fig. 2

t-SNE (t-Distributed Stochastic Neighbor Embedding) is a probabilistic technique for reducing dimensionality that was tested alongside PCA in order to compare results. The best decision tree trained on two components of t-SNE achieved an accuracy of 84%, with the decision tree surface shown in Fig. 3.

Comparing and contrasting the two methods, PCA is significantly faster to compute compared to t-SNE; t-SNE requires exponentially more time to calculate pairwise connections between data points. Additionally, the results of PCA can be interpreted more easily (percentage of variability per component), compared to the cryptic t-SNE components. However, the performance gain on the decision tree cannot be denied. Practically speaking, t-SNE would take drastic compute resources on the server to calculate and is subject to finely tunable hyperparameters (perplexity). For these reasons, we determined PCA is a better dimensionality reduction tool.

4: In order to explore the effectiveness of `set(list[])`, the UI was built with the non-technical person in mind. Specifically, each of the ridgeline plots

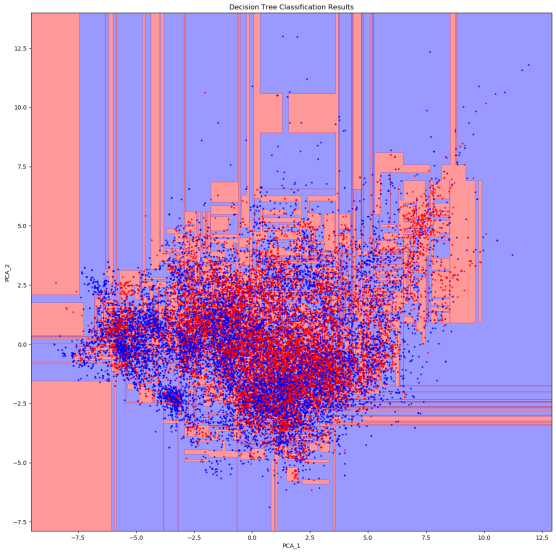

Figure 2: Decision Tree Surface Plot - PCA

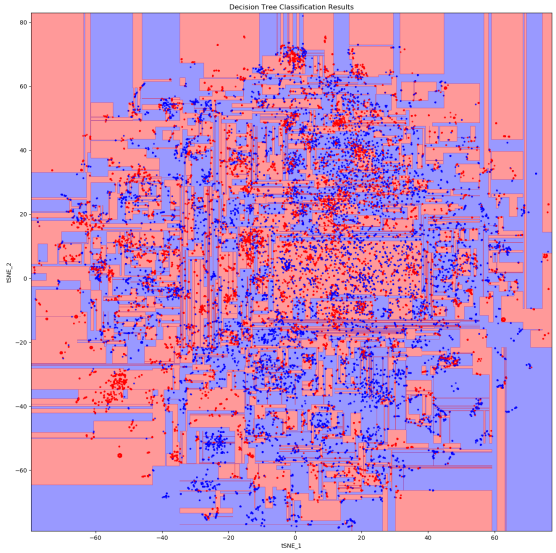

Figure 3: Decision Tree Surface Plot - t-SNE

have the description from Spotify to explain the difference between each low-level feature.

A screenshot of the R Shiny interface is shown below, in Fig. 4

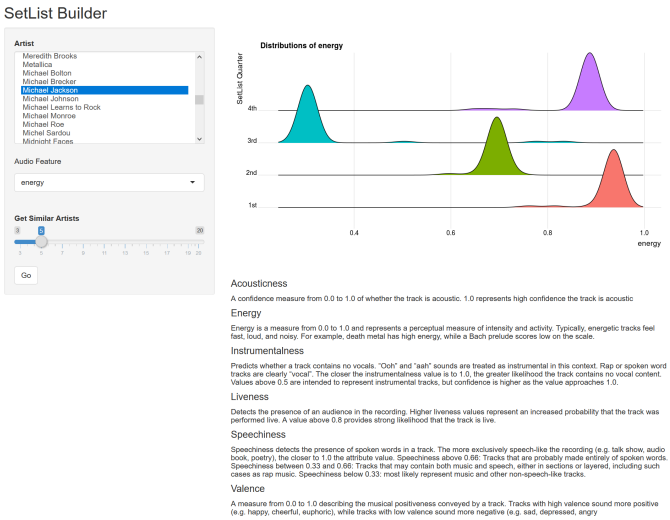

Figure 4: Final User Interface for `set(list[])`

Conclusion

`set(list[])` aimed at using low-level audio features in order to aide musicians in searching for similar setlists. By scraping setlist.fm and merging feature data from Spotify, we successfully clustered artists in a non-semantic way.

By utilizing only audio features, genre and user bias were removed, illustrated by the non-overlapping artist recommendations with Spotify (which are user-driven). Setlists and artists were successfully clustered using K-Means Clustering. In addition, both PCA and t-SNE dimensionality reduction applied to the setlist provided an accurate encore predictor.

Lastly, a web interface enables a musician to select artists and view cluster data and ridge-line plots of audio features of similar setlists.

Distribution of team member effort

All team members contributed a similar amount of effort.

References

[1] Carson K. Leung and Abhishek Kajal. "Big Data Analytics for Personalized Systems". In: *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing* (2019).

[2] Dan Liu, Lie Lu, and HongJiang Zhang. "Automatic Mood Detection from Acoustic Music Data". In: *Proc. ISMIR 2003; 4th Int. Symp. Music Information Retrieval* (Jan. 2003).

[3] Douglas Eck et al. "Automatic Generation of Social Tags for Music Recommendation". In: NIPS'07 (2007), pp. 385–392.

[4] D. Torres R. Taylor P. Boulanger. "Real-Time Music Visualization Using Responsive Imagery". In: (Jan. 2006).

[5] Geoffray Bonnin and Dietmar Jannach. "Automated Generation of Music Playlists: Survey and Experiments". eng. In: *ACM Computing Surveys (CSUR)* 47.2 (2015), pp. 1–35. ISSN: 03600300.

[6] Andreu Vall et al. "Feature-combination hybrid recommender systems for automated music playlist continuation". eng. In: *User Modeling and User-Adapted Interaction* 29.2 (), pp. 527–572. ISSN: 0924-1868.

[7] Markus Schedl. "Deep Learning in Music Recommendation Systems.(Report)(Abstract)". eng. In: *Frontiers in Applied Mathematics and Statistics* 5 (2019). ISSN: 2297-4687.

[8] Lixin Zou and Long Xia. "Reinforcement Learning to Optimize Long-term User Engagement in Recommender Systems". In: *KDD '19: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (July 2019). DOI: 10.1145/3292500. URL: https://di.acm.org/doi/10.1145/3292500.3330668.

[9] Beth Logan. "Music Recommendation from Song Sets". In: *ISMIR* (Oct. 2004).

[10] K. Kim D. Kim. "A music recommendation system with a dynamic k-means clustering algorithm". In: *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)* (2007).

[11] Thierry Bertin-Mahieux et al. "The Million Song Dataset". In: Jan. 2011, pp. 591–596.

[12] K. M. Ting Z. Fu G. Lu and D. Zhang. "A Survey of Audio-Based Music Classification and Annotation". In: *IEEE Transactions on Multimedia* 13.2 (Apr. 2011).

[13] Martin F. McKinney and Jeroen Breebaart. "Features for audio and music classification". In: *ISMIR* (2003).

[14] Julien Osmalskyj et al. "Efficient database pruning for large-scale cover song recognition". In: Jan. 2013, pp. 714–718.

[15] Yang, Dong, and Li. "Review of data features-based music emotion recognition methods". eng. In: *Multimedia Systems* 24.4 (2018), pp. 365–389. ISSN: 0942-4962.

[16] Chih-Chin Liu Jia-Lien Hsu and Arbee L. P. Chen. "Discovering Nontrivial Repeating Patterns in Music Data". In: 3.3 (Sept. 2001).

[17] Markus Schedl Peter Knees. *Music Similarity and Retrieval*. 1st ed. Springer-Verlag Berlin Heidelberg., 2016.

[18] Tanner O'Rourke. "Temporal Trends in Music Popularity – A Quantitative analysis of Spotify API data". In: *Temporal Trends in Musical Characteristics* (2018). DOI: 10.13140/RG.2.2.11551.71843.

[19] A Salomon B Logan. "A Music Similarity Function Based on Signal Analysis". In: *ICME* (2001).

[20] Pinghui Wang et al. “A Memory-Efficient Sketch Method for Estimating High Similarities in Streaming Sets”. In: *KDD ‘19: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (July 2019). DOI: 10.1145/3292500. URL: https://di.acm.org/doi/10.1145/3292500.3330825.

[21] Changii Li Yifan Hou Hongzhi Chen. “A Representation Learning Framework for Property Graphs”. In: *KDD ‘19: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (July 2019). DOI: 10.1145/3292500. URL: https://di.acm.org/doi/10.1145/3292500.3330948.

[22] Paul Lamere and Douglas Eck. “Using 3D Visualizations to Explore and Discover Music.” In: (Jan. 2007), pp. 173–174.