Playlist Generation from Single Song Selection, A Generic Approach

Cole Perschon, Cindy Liao

April 15, 2019

1 The Problem with Making Playlists

Say you're planning a Game of Thrones themed party, or are tasked with playing music at the neighborhood barbeque. In either case, the music you choose will dictate the mood of your event. For example, the change of setting between playing "In Da Club" by 50 Cent vs. "Happy Birthday to You" by Patty Hill at your son's 5th Birthday Party is unarguable. And up until now, getting it right has come at a cost. In the past the best options have been to either make a playlist beforehand, or to just put on an internet radio station such as Pandora or Spotify if you're in a pinch for time. There are pros and cons to both approaches, but neither seem to get it quite right. Making playlists ensures you won't hear an unwanted song, but requires choosing songs manually from your potentially massive library, which limits your total song allotment to the amount of time you have to work on the playlist. In contrast, playing an internet radio station saves you time, but runs the risk of playing unwanted songs. Our solution to this was to allow you to generate an unending playlist of your songs, no matter how obscure, based on just a single song selection!

When we set out to accomplish this goal, we discovered a variety of third party music rating platforms such as The Echo Nest, which already offered great APIs to connect to and discover your songs similarities based on things such as genre and community ratings. However, the problem with services such as these is that they can be limiting in that they only have data on so many songs and artists. Meaning that many of the more unique and/or niche songs in your library would never get played. To encapsulate all of your music, it was clear that a more general solution to assessing song similarity was needed. Our approach was to dissect songs into their key components, using music theory, then compare these to determine song similarity. This way, even the upmost original of recordings in your library could get played.

2 Data

The data set used was a random 22,000 song sampling of the Million Song Subset [1]. Who's reduction, on the basis of both time and complexity, will be discussed more in depth later in this report. Further, although a total of 10 other fields were used in the testing of our proposal, our accuracy analysis included only a selection of the main audio fields from each song's field list: *loudness*, *key*, *mode*, *tempo*, *time signature*, and *artist mbid*. Described as "overall loudness in dB", "key the song is in", "major or minor", "estimated tempo in BPM", "estimate of number of beats per bar", and "ID from musicbrainz.org" respectively. Also included in the accuracy analysis was a complementary community dataset, Last.fm dataset [1]. Used as a sanity check, we compared

clustered songs with their 1-40 respective (genre) terms given on musicbrainz.org, which were rated by the community.

To store the massive (493 GB) dataset [1], along with the community dataset [1], we spun up an Amazon Web Services (AWS) EC2 instance running an Ubuntu Server 18.04 volume at (ec2-52-91-85-148.compute-1.amazonaws.com), then mounted the Amazon Public Dataset Snapshot to it. To process the data locally, we created a Python Script utilizing Paramiko, "a Python implementation of the SSHv2 protocol", to create both the SSH and SCP clients needed to access and download each of the HDF5 formatted song files. From there we were able to obtain our selection of fields from the files using the hdf5_getters.py module and appended each to our *library_csv.csv* document, as we sequentially iterated through and deleted each HDF5 file. This allowed us to get the data into both a dramatically more condensed (10 MB), and easier to process state, without ever needing to store the full extent of it locally. For 22,000 songs, or (2.2%) of the dataset, this took approximately 3 hours to generate because of the file transfer speed restrictions from SCP. Once generated, our *library_csv.csv* became the sole referencing document in all subsequent similarity procedures.

3 Key Idea

The key idea in this project was to discern whether or not it was feasibly possible to accurately determine song similarity based on just the core components of songs, without the use of accessory audio fields such as assigned genre, etc. This required the extension of our combined knowledge of music theory and much experimentation. Specifically, we were interested in learning what song components (included in our song fields) create a greater emotional experience for the listener [2]. For example, songs with higher *time signature*(s), *loudness*, and with a *mode* of major, convey feelings of excitement and happiness.

We also wanted to see if it was better to choose songs only in the same clusters. In order to compare the two, we compared genres of the songs in the playlist to the starting song. The genres of each song is also included in the larger dataset; however, we didnt use that category for our project. We predicted that cluster-based playlists will create a more cohesive emotion/mood because the arguments of each song will be more similar to each other.

4 Results

Web Server, the Sluggish Space-Saver

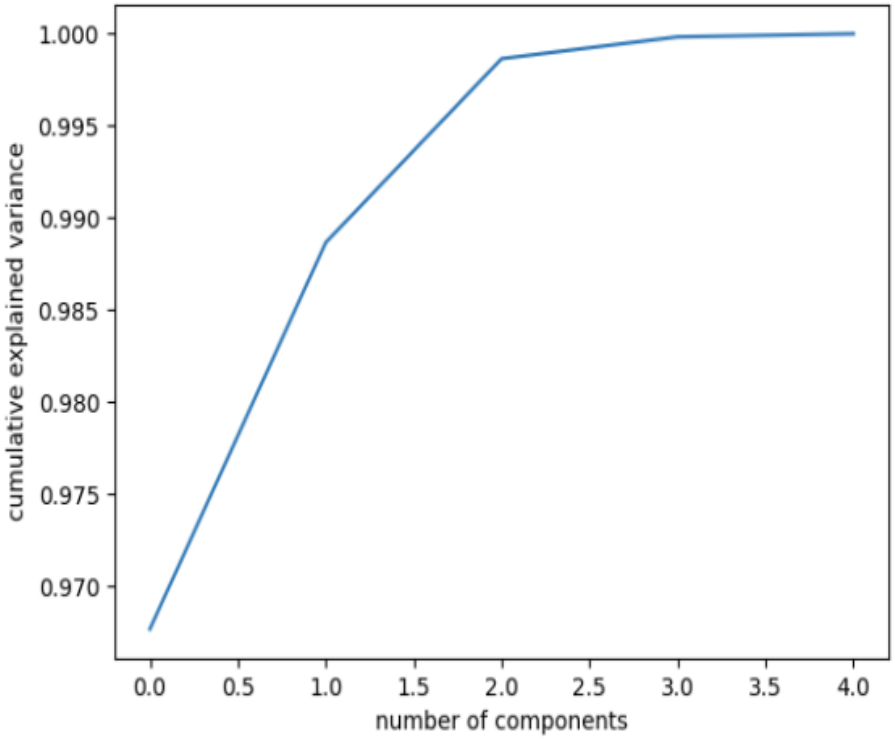
In summary, the web server described under **Data**, was a great success. It allowed us the freedom to store the brunt of our data remotely, of which, neither of our computers even had the space to hold anyways. Since we spent a majority of our time iterating through all possible main audio fields, including those not previously listed: *tone*, *energy*, *segments_pitches*, and *segments_timbre*.

Song Suggestions

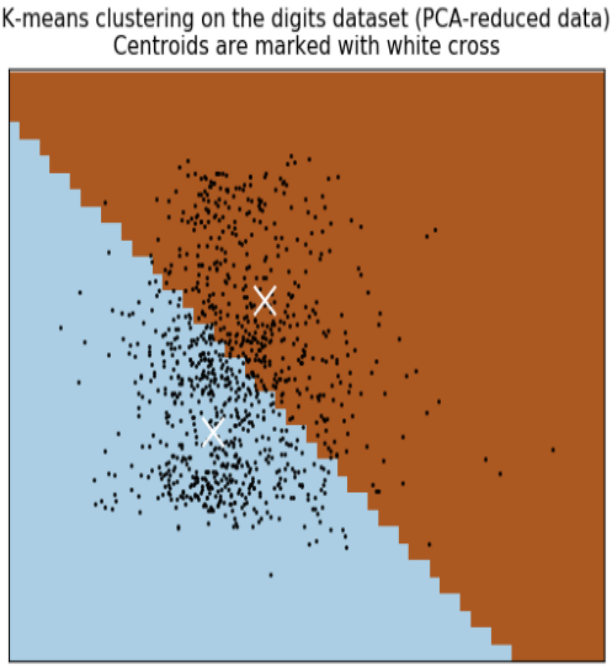
Our first step toward achieving generic song similarity suggestions was the creation of two classes: *Library* and *Song*. *Song* held the main audio fields for a given song whereas *Library* held these fields for all songs. *Library*, not to be confused with *Library_csv.csv*, also contained our required

functions for generating song suggestion. Populated by selecting n songs randomly, *Library*'s size is in the range $[1 \leq n \leq 22{,}000]$.

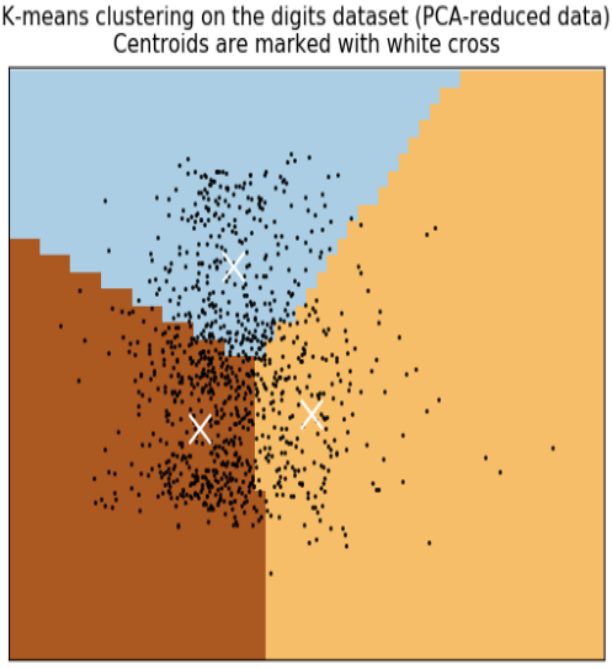
After *Library* is populated, we first call upon its K-Cluster function. In order to determine how many clusters are appropriate, we used the elbow technique after running Principal Component Analysis (PCA) on the standardized data. We can then see approximately how many clusters are needed to explain the percentage of the variance of the data [3] by producing a cumulative distribution function (CDF):



From here we produced k-means clustering graphs using a sample size of $n = 1{,}000$ for a better depiction of the data points. Our initial 2-means variant can be seen on the following page below:



Despite the plotted data in 2-Dimension not showing any visible clustering, we continued with a 3-means variant here:



Accounting for the blatant lack of visible clustering in the 2-Dimension graphing, we decided to generate two playlists, one where the songs added to a playlist are constrained to the clusters they

are in, and another where the songs were chosen solely in accordance their closest neighbor, to determine the difference clustering made on the data. In order to do so, we made a dictionary that mapped each song to their clusters based on their cluster center after running K-means, which was done in 5-dimensions, including all fields.

We are just about ready to start building a suggested playlist. Now, we needed to start with a single song from the *Library* to build upon. For our purposes, the starting song was selected by chosen at random from indexing said matrix. We find where the song is, then add its nearest neighbor to the playlist that hadn't yet been added. We then recalculate the center of the "playlist" to adjust for newly added songs. And, after the playlist had filled up to the desired size, we recorded the results in a .txt file, which includes the title of the starting song and the titles of the subsequent songs in the playlist.

There were several trails of cluster-based playlists and non-cluster-based playlists made. We determine the accuracy of each playlist and average the percentage.

Past Experiments

In the beginning of the project we went back and forth on deciding whether to suggest songs based off of a pre-existing playlist, to suggesting an entire playlist from one song. We built a working implementation for the former, and after building our library, we created a random playlist of size m. The created playlist pulled random songs from the library and added it to a new set. Afterward, we would calculate the playlist center and determine which cluster it fit into best. We would choose the closest song in that cluster that isnt already in the playlist, then add the new song to the playlist and recalculate the playlist center. We would repeat the process for x number of songs.

We decided to switch our method because there are already similar functions for aforementioned song streaming services. We decided itt would be more interesting to try to determine a mood off of one song. Also, the first implementation defeated one of our goals which was to minimize time in creating a playlist.

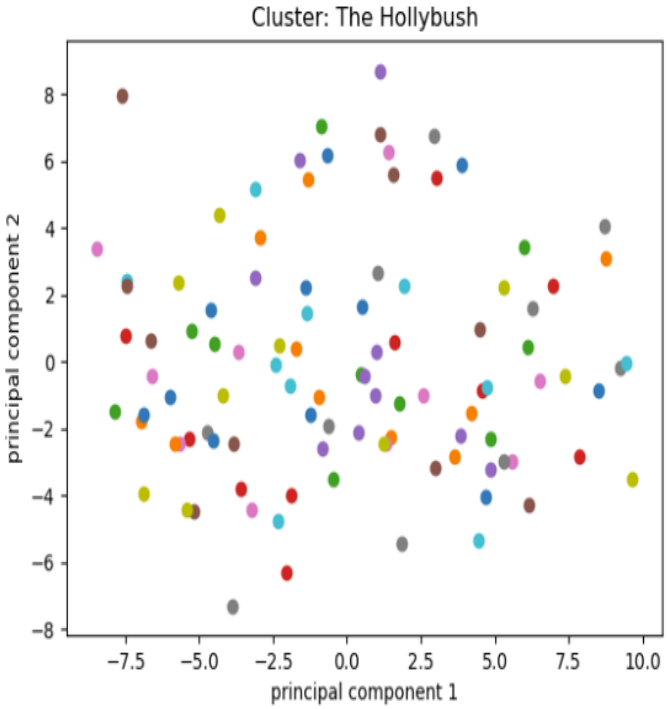
Accuracy Analysis

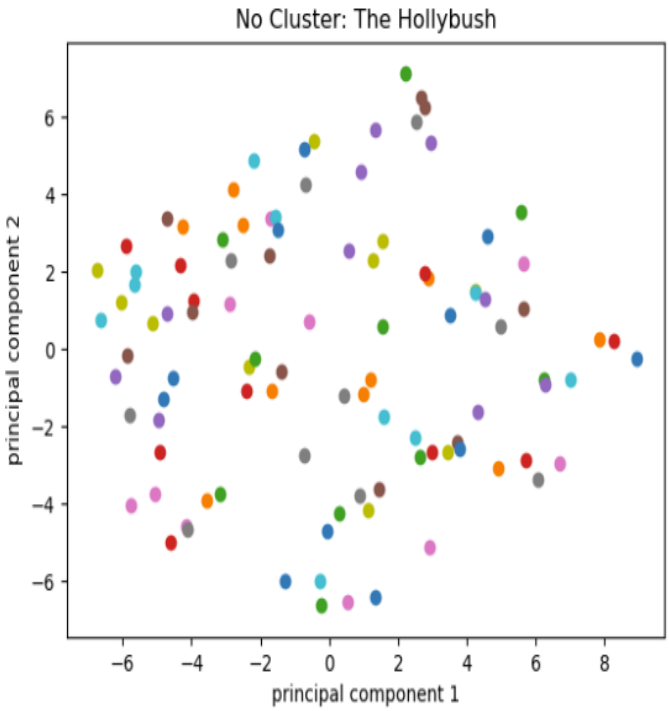
Determining the accuracy of the generated playlists proved to be a challenge since genre terms in the Last.fm database [1] were only given to artists, not their songs. However, that did not stop us from attempting some form of analysis. Now, since the Last.fm database [1] has such a massive list of terms (1,109,367), we condensed each artist's terms into one or more of 22 different sets: (alternative, blues, classical, country, dance, disco, electronic, folk, funk, gospel, hip hop, house, jazz, latin, metal, pop, punk, rap, reggae, rock, soul, soundtrack). Following this, we utilized the Jaccard-Similarity: $JS(a,b) = \frac{|a \cap b|}{|a \cup b|}$, between an initial chosen song, and 30 of its most similar song suggestions from both the variant based on clustering, and that based on distance (control), for three songs selected at random: "Don't Let Me Explode", "On doit rien a personne", and "Mars on Mars (Trance Mix)". What we found by taking the empirical means of each of these tests was that in each case, the clustered variant edged ahead, but only slightly, with marginal amounts of (0.0123, 0.0245, 0.0082) respectively. Mostly due to the fact that approximately 80% of the songs in each pairing were the same.

Since this analysis could only be done on artist id's and not based on the actual songs, and with such a marginal difference, we find little reason to believe that creating genres via clustering aids in finding similar songs based on the fields we had tested. With little quantitative reasoning to prove otherwise, the best methodology test may be to just listen to the suggested songs!

5 HOW Similar are The Suggested Songs?

In an effort to better demonstrate what we learned, here we used PCA again to plot our generated playlists. This is an example for the song "The Hollybush" with a library size of $n = 1{,}000$ songs and a playlist of 100 songs: Of these 100 songs, there were 78 that were the same. Yielding us a 78% in similarity between the two playlists. This indicates that most of the songs suggested are still in the cluster. This can be explained by how much of an outlier the starting song is - the higher the percentage of similarity between playlists, the more of an outlier. This is because there are more songs towards the center of the data. If the song is closer to the center, then it has more chances to choose songs that arent in its same cluster. When a new song is added and the playlist center is recalculated, it can be pulled further from its cluster center.





If the starting song is closer to the center, from our analysis we predict that no cluster might be more accurate. This is do to the fact that cluster-playlists are restricted by which songs they can choose. There might be a closer song that isn't in the cluster, but it wouldn't be picked. In the future it might be a good idea to compare playlists from songs that are outliers and songs that are close to the center. In summary, we learned that judging song similarity based on best music theory practices is hard, and analyzing the accuracy of said similarity is even harder! Our limited knowledge of these practices only increased the difficulty, as we determined that none of the methods in which we tested yielded tangible 2-Dimension clusterings.

References

1. Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere, 2011, 'The Million Song Dataset', *In Proceedings of the 12th International Soceity for Music Information Retrieval Conference (ISMIR 2011)*

2. Michael Nuzzolo, 'Music Mood Classification', 2015, *Electrical and Computer Engineering Design Handbook: An Introduction to Electrical and Computer Engineering and Product Design by Tufts ECE Students*

3. An Approach to Choosing the Number of Components in a Principal Component Analysis (PCA)