# Project Conclusion:
# Song Recommendation System

Santa Clara University
Applied Machine Learning
Spring 2019

## Team

Olivia Hess, Meghan McGinnis, Liam Van Leynseele

## Subject

The motivation behind exploring a song recommendation system stems from our general interest in music and our general frustration with current song recommendation systems. These particular systems found in common sites such as Youtube, Spotify, and Pandora, tend to recommend predictable songs time and time again. The reason these systems are more or less deterministic is that their algorithms are fueled by user history. They keep track of the artists, playlists, and albums that you listen to, and then they utilize those surface level externalities of artist, genre, playlist, etc to recommend similar songs that you might like. We wanted to explore a system that went past those external features of the song and really dug into it. We decided to use the internal structure of a song to classify it, and find songs with similar classifications to recommend to users. The end all project goal is for a user to be able to input a song of choice and receive a list of “recommended” songs as output.

We experimented with the internal logic of 10,000 songs and developed a better sense for what would and what would not work for approaching accurate predictions. We will discuss the various routes we took with our experimentation in the Experiments Section. Ultimately we found that there is no single solution to this problem and every approach has its corresponding pros and cons. Comparing these advantages and drawbacks was the most interesting part of the work for us. At this point in time, the system is still in the works and is not yet capable of delivering the outputted list of recommended songs that we expect it to at some point in the future. Much of our time went into understanding how to deal with the massive dataset and properly cluster the songs so the following sections will focus on detailing this process.

## Data

We utilized the 10,000 song subset provided within Amazon Web Service’s “Million Song Dataset”. This decision to use the subset rather than the 1 million song set, which is a 280GB download, came down to a decision of maintaining system functionality and efficiency on our personal laptops.  This system could of course handle a larger dataset moving forward, but this would require a more robust configuration which was beyond our project's scope at this point in time, This dataset's most noteworthy features are listed below:

```
def get_artist_location
def get_artist_familiarity
def get_artist_hotttnesss
def get_song_hotttnesss
def get_title
def get_duration
def get_key
def get_loudness
def get_mode
def get_tempo
def get_time_signature
def get_segments_start
def get_segments_confidence
def get_segments_pitches
def get_segments_timbre
def get_segments_loudness_max
def get_segments_loudness_start
def get_sections_start
 def get_sections_confidence
def get_beats_start
def get_beats_confidence
def get_bars_start
def get_bars_confidence
```

Some of the above features are categorical while the majority of them are numerical.  We actually had no plausible cause for converting any of the categorical features into a numerical scheme because they did not seem to further our cause in terms of looking into the internal structure of the music.  One example of this would be the the artist location which we found to be an exceedingly unclean column of data where one entry would be "Miami, Florida" and the next would be "FRANCE", with absolutely no consistency in convention. Furthermore, about a quarter of the song entries did not have an artist location at all.  Had this particular feature been of more importance to our ultimate goal we could have found ways to go through and clean the data, but because we believed it would not contribute to a more accurate prediction, we decided to drop this column completely.  This is just one example of the preprocessing decisions we had to make in order to clean our dataset.

One main obstacle we encountered with our dataset was figuring out how to deal with features that were not singularly numerical - they were lists of numerical values. Interestingly enough, we found that the main features we were interested in had this structure of being lists of data points. These features were the ones that pertained the most to the actual composition of the music: timbre, segment starts, segment pitches, etc. Because these lsts actually were the same size for every song, we were naturally inclined to just compare each point within a list to those of the other lists. However, with a list of 100 points, this approach would be doing little more than adding 100 more dimensions to the already large feature set. Our goal had to be to reduce dimensions, not grow them. Our solution to this major difficulty will be discussed later on in this report.

## Features

As mentioned in the above Data Section, the majority of our time was spent understanding how to deal with the dataset's features. The features that we put the most time into learning how to utilize were the ones that provided information about the internal composition of the song rather than the ones that provided externalities about the music. Although we did not necessarily "derive" any features, we did put a substantial amount of time into performing linear dimensionality reduction on features that were not a single numerical point, but rather a list of numerical values describing a certain attribute of the song, such as Timbre. To do this we had to make use of PCA, the Principal Component Analysis, provided by sklearn. This tool allowed us to decompose the high dimensionality of a data attribute in order to project it to a lower dimensional space. By making conversions from many dimensions to two dimensions, we were then able to see the features in a 2D Cartesian Coordinate system and proceed to cluster them effectively. In total we make use of 12 features.

## Experiments

After an extensive amount of data preprocessing, we wanted to plot different features next to each other to see correlations between data. We also did research into the intricacies that make a song like other songs, and one of the largest indicators of similar songs happens to be the way that timbre changes throughout time. With the assumption that specific genres will exhibit these timbre changes, we really began playing with the get_timbre_segments feature provided to us by MSD. As we said in the above section, this first required some major dimensionality reduction. Each track is split into a variable number of segments, and each segment has 12 Mel-Frequency Cepstrum values, which are representations of the short-term power spectrum of a song. This value is based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. We first reduced the 12-dimension segments of each song into 2 dimensions. This allowed us to have a visual representation of a song on a scatter plot. To represent one song, we took the centroid of the latter described representation. We then pursued the following clustering methods.

## K-Means

K-Means seemed like an obvious choice for our first clustering method, because of its simplicity and relatively short training time. It was easy to implement, and it is advantageous that an instance can change clusters when the centroids are re-computed. The division of objects occurs on the objects which are “similar” between them and “dissimilar” to the other cluster’s objects. This was beneficial in our case because we were trying to find a similarity between the principal components of timbre’s in a song. However, because of our ultimate way of representing the songs (by the centroids of their own respective scatter plot), it was not as accurate as we would have liked. However, it was a great algorithm to start with, having a decent complexity of O(n^2). But because this complexity means it is not so effective in large applications, when we extend the algorithm to cluster a million songs, K-Means will certainly not be the best way to go.

## Agglomerative Clustering

As we will show in the results section, agglomerative and K-Means clustering had nearly identical clustering results. Agglomerative clustering recursively merges the pair of clusters that minimally increase a given linkage distance. In scikit-learn, Agglomerative Clustering uses the linkage parameter to determine the merging strategy to minimize the average distance between observations and minimize the variance of merged clusters. Because the results were so similar to K-Means, we concluded that there are certainly more parameters we need to take in for our ultimate prediction algorithms to create an accurate prediction of genre clusters.

## DBSCAN

We had very high hopes for DBSCAN’s performance on our data, but the results were suboptimal which actually taught us quite a bit about our data and how we needed to proceed with it. We thought the density-based clustering method would work very well on our data as we did not expect it to cluster in perfectly round clusters, and a density-based clustering method is well suited for clusters of arbitrary shapes. The first iterations of running DBSCAN resulted in the generation of a single cluster. Even as we varied epsilon, the neighborhood radius, and the number minimum points, we continued to only get one cluster. After some analysis, we realized that the issue was arising from those data attributes that were lists. Because every value within those array lists are so close together, they ultimately cause the density based calculations to clump them all together, consequently amount to the single cluster that we were seeing.

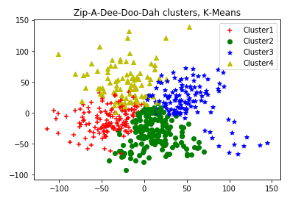
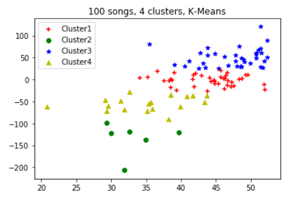
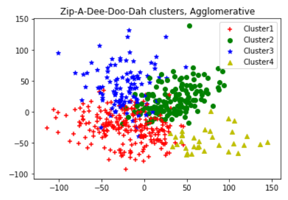
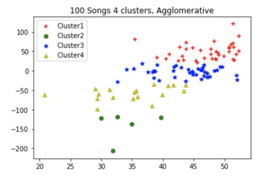
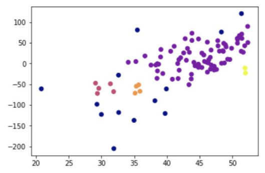
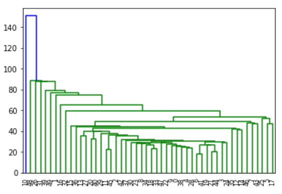
Once the PCA dimensionality reduction had been performed on these data attributes, we tried running DBSCAN again. At this point we had more success with finding more clusters, but another problem ensued. We realized that its performance was suboptimal because the clusters had varying densities. There was not a single epsilon value that could effectively accommodate all of these densities. If we were to continue on with utilizing DBSCAN, we would have to normalize our data attributes. Each attribute was vastly different in that their ranges of values took on a very large spectrum. Another solution could be to only use the features that did have similar ranges, but the risk here would be the loss of song information. If we only selected the features that worked well with DBSCAN, then we would ultimately sacrifice the complexity of what we are trying to analyze. However, we could take the limited DBSCAN results and couple them with another approach in order to make more broadly informed decisions.

## Hierarchical Clustering

We found that the hierarchical clustering worked particularly well with our dataset because of its underlying nature. Its iterative grouping nature showcased the complex underlying relationships in music that simultaneous clustering methods cannot necessarily represent. While this algorithm makes the most logical sense for predicting song recommendations from our dataset, we discussed the obvious drawback of its higher runtime complexity. If this system was to be run on the full million song dataset, the $O(n^3)$ runtime, and $O(n^2)$ space complexity would prove to be far too taxing.

## Results

.

| ***Model*** | ***Graphs*** | ***Training Error (100)*** | ***Testing(25)*** |
|---|---|---|---|
| ***K-Means*** |   | ***18.3%*** | ***27%*** |
| ***Agglomerative*** |   | ***17%*** | ***24.4%*** |
| ***DBSCAN*** |  | ***33.4%*** | ***64%*** |
| ***Hierarchical*** |  | ***15%*** | ***22%*** |

## Interpretations and Discussion

We will now discuss the biggest interpretations that we found while working on our song recommendation system. In all honesty, as soon as we saw the complexity of the dataset, we knew that we had our work cut out for us. However, we still anticipated that we would be able to apply some clustering algorithm that would cluster the songs that had some semblance of similar composition. What we did not anticipate was exactly how much work would be needed for preprocessing the enormous dataset. Furthermore, we did not anticipate how much insufficient preprocessing would adversely affect the clustering algorithms' performance.

Our first takeaway is the critical importance of really understanding the form of the data you are working with before proceeding. Because there was very minimal documentation on the AWS Million Song Dataset, we had to spend a lot of time simply understanding what each attribute was based off of the feature name. After understanding what each attribute was, we had to explore how each of these attribute were stored within the dataset. Was it categorical, numerical, or what it a list of numerical values? The biggest unforseen problem here was that the attributes that were lists of numerical values were very difficult to learn how to deal with. We discussed comparing each index within the lists, but ultimately decided against this approach as it would just end up increasing the amount of dimensions that we already were struggling to deal with. Discovering how to reduce the dimensions of these lists to a 2D cartesian plane was no simple feat, and ultimately showed us that successful clustering performance would be impossible without it.

Our next interpretation was that DBSCAN would perform the best on the data, but clearly DBSCAN presented some hurdles as mentioned in the Experiments section. We found that the most effective clustering algorithm for this particular system is Hierarchical Clustering. It took trial and error to come to this conclusion, but in hindsight we all acknowledge that hierarchical clustering makes logical sense with regards to our dataset. Because music is intrinsically composed of variation, a method that works agglomerative to group together similar genres is ultimately what we need. Of course the runtime is not ideal so that is something that we would continue to explore if we chose to use it for a full blown song recommendation system.

Overall, our results were good but incomplete. In other words, our work at this point is more of a theoretical exploration than a practical application. We spent so much time on properly preprocessing the data that we did not have sufficient time left to generate results on the entire million song dataset. However, our results are successful enough to proceed in growing our theoretical exploration into a successful song recommendation application.

## Conclusion

If we were to continue on this project, the first thing that we would want to do is perform dimensionality reduction on more song relevant features, like we did for timbre. Ultimately these list features are the ones that tell us the most about the composition of a song, and will be needed in order to make the prediction the most accurate. Once all desired features are properly reduced, we would again want to explore both DBSCAN and Hierarchical Clustering. Once our clustering has been streamlined, we would add the human component of testing the accuracy of our cluster's predictions. Because we are attempting to have machine perform the function of the human ear, ultimately the best way of testing the system's accuracy is to listen to songs that have been placed in the same cluster.