Milestone Report for Project 2

1. The Problem

Recommendation systems are usually built upon a social factor. “Other users purchased” or “others listened to” seem ubiquitous. A lot of music recommendation algorithms rely primarily on a social aspect or some sort of user interactivity or history to determine other songs that we might be interested in. This works to a degree as some of us are likely to check out albums that our friends recommend to us. Music tastes can change though, and a song that may have been a favorite a few months ago may no longer be. Since these tastes are both fluctuating and unique, I believe that it may be best to utilize a recommendation system based solely on audio features, allowing a playlist to be curated on just the audio similarities from one song to the next, allowing for a solo listening experience that can change as often as we would like.

1. The Client

The ultimate client would be any listener of music. Currently many people’s listening habits are based on others without even realizing, the most popular songs that are chart toppers are due to the fact that everyone is listening to them, leaving some quality music that may not be as well-known entirely un-listened to. By implementing this type of recommendation system, it benefits both the consumer and the music maker, by allowing listeners to experience music that they otherwise would not have, and the music maker reaches a wider audience. More immediately, music streaming and library services such as Spotify, Pandora, YouTube, and iTunes could implement this type of system in addition to existing algorithms to generate a unique listening experience. Though since these services already have a recommendation algorithm in place, it would be most beneficial for a new streaming service to be created that focused their recommendations solely on audio, allowing independent on smaller artists to increase their audience, purely on the music’s own merit.

1. Data Set and Cleaning/Wrangling

My initial intention was to work with all 1 million songs in the Million Song Dataset, but I quickly saw that my own hardware would be the limiting factor, and so for this project specifically, I decided to work with the random subset given by the compilers of the Million Song Dataset, which consisted of 10,000 songs randomly selected from the larger dataset as well as a small portion of the larger set, as broken up for distribution. The reason I included a small portion of the larger set will be detailed blow in the cleaning section.

Cleaning/Wrangling

One of the largest challenges I faced was getting the data into a useable format. The original dataset contained all the features I needed on the single song level. Each hdf5 file was a song containing song features. The random subset of 10000 of 10,000 of these tracks nested in folders which was then zipped up into one file. The rest of the 1 million song dataset was structured similarly to this, single hdf5 files inside dozens of folders then compressed into a tar.gz file. After unpacking the subset, I used a python program designed specifically to extract the features from the million song database set to read all the tracks into a single dataframe and did some exploratory data analysis. What I found was a ton of missing data where important song data features should have been. I knew that this alone would not be a good enough dataset for a project so I decided to use the subset as a list of songs that would be included in the dataset and turned to another source that I knew would have the data I needed, Spotify’s API.

The million-song dataset was scraped off of an API, which was later acquired by Spotify so I knew that most of the songs from that dataset would be there, excluding the fact that Spotify has a nearly immeasurably large database of songs on its own. Therefore, from the original hdf5 datset, I only extracted the artist and song name. Getting the song features from Spotify then became a two-fold process. The first was passing in the artist name and song title into the search portion of Spotify’s API in order to obtain the Spotify ID for it. The maximum number of queries able to be searched at a time in 50, so I grouped the artist/song pairs into groups of 50. After running through the set, I realized that the total was less than 10,000 meaning that a portion of the songs in the original sample set could not be found in Spotify’s library. Since I did not want to work with a small dataset, I used the first tar.gz file of the entire dataset to gather more artist name/song title pairings to fill out a more robust dataset. After gathering the Spotify IDs, I was then able to use Spotify’s web API to get audio features for several tracks by having arrays of 100 Spotify IDs at a time to return audio features. They were returned as a json file which was subsequently turned into a dataframe. At that point, I realized that there was duplicates as the original subset did contain songs that were also in the first file of the large dataset. After checking for and removing duplicates, the entire dataset was compiled into one large dataframe with approximately 14000 tracks and all its accompanying audio features. The only identifying portion in the dataframe of audio features was the Spotify ID which doesn’t bear any useful information to a person, so I created a dictionary with the Spotify IDs as well as the Artist and Name to join a bunch of audio features to a song title that people would actually recognize. Then I worked on cleaning out extra coluimns that had no relevance to the final product, leaving only audio features, one ID, and an artist name and song title.

Then I normalized all numerical variables so that all values are between 0 and 1 to make differences able to be computed within al the same scale, so one category where all variables are hundreds of times larger does not skew distance/inertia in clustering.

1. Other data sets

In a perfect world, I would have been able to use all 1 million songs in the million song dataset, but there is nothing stopping this from working on Spotify’s entire library. In addition, Spotify has not just audio features but also audio analysis but there does not seem to be a set format in which that data is returned, so I refrained from working on it for this project. The last time the million song dataset was nearly a decade ago, so if I was more creative I would have figured out how to incorporate more recent songs, however, Spotify’s search algorithm put a big weight on popularity when searching for songs, which I wanted to refrain from, made it difficult to get a well rounded set of both very popular but also very unpopular songs to create a balanced dataset. What I would have also liked that was not included in this dataset in additional audio features such as timbres, pitch, and other frequency type signals.

1. Initial Findings

The way to build a recommendation system with just this raw data set is naturally to find the songs that are closest to each other. Therefore each song then becomes an array of its audio features which can then be compared to each other to determine distance to other songs. This can be done without any machine learning by calculating pairwise distances between every pair of songs in the dataset. However, even with ~14000 entries in this dataset, it becomes an absolute nightmare to calculate and store and gets exponentially worse as the dataset grows, and the hope is for this to be relevant no matter how large the dataset, therefore it is necessary to implement some machine learning. To make this easier for larger datasets, I tried two different approaches. First was kprototypes clustering, which is similar to Kmeans but accepts both numerical continuous variables as well as categorical variables to cluster, then cluster them twice, dividing up the dataset into recommendation size cluster. In addition, I converted categorical variables to dummy cateogries to be able to complete a k-means clustering in which I was able to use a larger number of cluster, but which also gave me a new problem: songs in their own clusters. To combat this problem, I did pairwise for all songs with less than 4 in the clusters, which would then be stored on their own where as clustering would return recommendation for the others.