MusicMatch: Using Spotify API and Machine Learning for Audio Feature Visualizations and Song Recommendations

Andrew Nickerson, Catherine Huang

December 13, 2021

1 Introduction

1.1 Motivation

With over 380 million monthly active users, Spotify is the world's largest music and media streaming service provider. Many of us know Spotify as the go-to platform on which to unwind during study breaks, share playlists with friends, discover new music, and—for the ambitious among us—land among the top 0.1% of active listeners for our top artists. From an engineering and technology standpoint, however, how does Spotify provide the aforementioned features and experiences for its users? **Specifically, how does Spotify use technology to capture an impression of each user's music taste, and how does it generate song recommendations based on that impression?** Fortunately, Spotify's *open-source Web API* (Application Programming Interface) (`https://developer.spotify.com/`) helps us get one step closer towards exploring these questions. This API allows any programmer to perform functionalities similar to what Spotify's web and mobile applications perform every instant. One can explore metadata about songs, playlists, users, or artists (for example, a user's top tracks, an artist's top albums, or a song's audio features) with the API.

Given that most other companies do not make the technology behind their products open to the public, we were eager to explore Spotify's Web API to get a sense of what lay behind the scenes of a service whose underpinnings are at the intersection of music and technology. In this project, we set out to 1) visualize the **audio features** of any playlist by any user, as well as 2) generate **song recommendations** for that user given the playlist.

1.2 Our Approach

There are several ways to use the API, for example, either through a *node.js* web server or through Spotipy, a *Python wrapper* for the API. We chose the latter, since 1) it allowed us to focus directly on the API's features rather than setting up a complicated web environment, and 2) Python is generally a more succinct and versatile langauge to work with than JavaScript. To call Spotipy, we worked on Jupyter Notebooks hosted on DeepNote, a cloud-based notebook editing and execution environment.

In order to visualize audio features and generate song recommendations, we used the Web API to first fetch metadata about a user's playlists and the songs in each playlist (given a username), and then we used various machine learning and data visualization techniques to handle the rest. Regarding the latter, Spotify API already supports generating song recommendations given a set of audio features, but we intentionally wanted to explore the feature representation, clustering, and machine learning behind creating such a recommendation algorithms. Our algorithm combines information from web sources (as initial inspiration) with our existing programming, machine learning, and musical knowledge.

1.3 Summary

In this paper, we first discuss the Spotify Web API and its features in detail, as well as how we used the API. We then present our code, and this includes a walkthrough of all three Jupyter Notebooks we worked on, as well as a conceptual overview of our recommendation algorithm and of the clustering and dimensionality reduction techniques we use to generate two-dimensional visualizations of which songs related to each other and which are not. In the section after that, we discuss how our work both draws from and connects to topics covered in GENED 1080. Lastly, we will analyze and reflect upon this project and the process of working on it.

2 Overview of Spotify Web API

2.1 Overview

Spotify offers a wide range of possibilities for developers with their developer console, called "Spotify for Developers" (`https://developer.spotify.com/`). Its clear that Spotify cares about its community and the exploration of the data they have to offer, as the open-source Web API and other services under Spotify for Developers is quite extensive.

The API features 50+ endpoints, which one can think of as unique functions that will send or receive data from Spotify's music database with information about any particular track, artist, album, genre, or user found on Spotify. For example, if we know a particular artist's name (or rather, their unique alphanumeric Spotify artist ID), we can extract the information listed on Spotify's API documentation (`https://developer.spotify.com/documentation/web-api/reference/#/`) which includes data such as the artist's total followers, their related genres, an image of the artist, and an integer representing their popularity ranking amongst other artists.

2.1.1 API Features

As mentioned, the Web API contains many open-source endpoints that allow for the querying of music-related data embedded in the Spotify library. The endpoints are divided into sections by topic, such as Album, Artist, Tracks, Users, Playlists, Genres. Within each, there are unique endpoints that serve a specific function (such as the `GET ARTIST` endpoint mentioned above). The API can do things from adding and deleting songs from a user's playlist to making song recommendations (using the Spotify *algorithm*) based off seed artists or tracks. Many of the endpoints are available without user authentication, however, there are some endpoints—such as adding a song to a private playlist—that require a user to login using their Spotify credentials. Regardless, it is phenomenal that Spotify offers such an extensive range of functionality at no cost to the developer.

2.1.2 Spotipy (Spotify API for Python)

We used Spotipy as a bridge between our language-of-choice, being Python, and the Web API. We made this decision in avoidance of the inevitable large amount of time that would come with setting up and debugging a new *node.js* web environment. Although creating a web app would be something we'd be interested in working on in the future, it did not fall within the timeframe nor scope of this project. Rather, we chose to use Spotipy, which is a Python library that would allow us to focus on using the API and its functionality rather than setting up an environment unrelated to the learing goals of the project.

2.2 How We Used Spotipy and the API

Spotipy offers all of the functionality provided by Spotify's API, but makes querying Pythonic. This allowed us to do all computations in a Jupyter Notebook environment as mentioned in Section 1.2, making collaboration and exploration much easier than the use of a node app.

Within the API, we were particularly interested in harvesting information about a user's playlist and pulling audio features and metadata from each track on a given playlist, as well as making recommendations to a user in some way. For this, we primarily made use of the `GET USERS PLAYLISTS` and `GET SEVERAL TRACKS` endpoints, which served as the main tools necessary to achieve our playlist audio analysis and song recommendation algorithm.

Spotipy made it simple to make our API calls, as we were able to keep everything contained within three Jupyter Notebooks hosted on DeepNote, a cloud-based Python notebook editing and execution environment. We now dive into the contents of these notebooks.

3 Code Run-Through

The following DeepNote project houses all of our notebooks: `https://deepnote.com/project/MusicMatch-aXccbqduS3WAQ0CbCHtrcA/%2Frecommender.ipynb`.

3.1 Python, Deepnote, and Jupyter Notebooks

After settling on using Python as our programming language, we decided to utilize *Jupyter Notebooks*, which are executable coding notebooks (.ipynb). The term notebook is used because the code within is separated into cell blocks and can be executed one-at-a-time, unlike a Python script (.py). Additionally, we decided to host our notebooks on a web-based editor called DeepNote rather than locally, as the sharing capabilities of DeepNote are highly conducive to collaboration.

3.2 Audio Features

As mentioned earlier, Spotify's Web API allows us to extract metadata about any song; such metadata includes not only popularity, year released, and duration but also audio features, such as valence, acousticness, danceability, energy, instrumentalness, liveness, loudness, speechiness, and tempo. Many, if not all, of these features are numerical (e.g. a value between 0 and 1 for some of these features), which means songs with similar numerical values for many of these features will tend to be similar overall. We draw upon this helpful property in both our feature representations and our algorithms, and the sheer number of data points (i.e. songs) and features allows our data to be meaningful enough to provide for an accurate and helpful algorithm.

3.3 Notebook 1: Playlist Visualizer

The first notebook, *playlist_visualizations.ipynb*, performs the following: given a Spotify username and a playlist, we utilize Spotify's Web API and the features it collects on every song in its library (over 50 million songs) to create a polar graph visualization of that playlist's relative danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo, and duration. This aspect of our project came out of our initial exploration of Spotipy, a Python library that utilizes Spotify's Web API to hit API endpoints that extract data on tracks, artists, albums, and users—that is why the notebook begins with analyzing the audio features of a singular song given the song's unique Track ID (from the URL). The next exploratory task we did (that appears in the notebook) is that given an artist (in the form of Artist ID), we called the API to get information on similar artists. After those two exploratory tasks comes our main tool: the `getPlaylists(username)` function uses the `GET USERS PLAYLISTS` endpoint of the Spotify Web API to retrieve a user's public playlists on spotify, complete with all of their songs and information about each song. We use the output of this function to print for the user a list of their own playlists. The next function, `analyze_playlist(playlist_id)`, takes one playlist and extracts information about each song in the playlist, taking the mean of the data for each audio feature, and graphing it using Python's Matplotlib visualization package. These functions are executed in the last cell of the notebook, which generates input prompts for the user, taking their username and asking for them to select one of their playlists for the code to analyze. Once a playlist is selected, its mean audio features are visualized. Figure 1 is a sample visualization of the features of one of Catherine's playlists titled "...beethoven?":

3.4 Notebook 2: Song Data Set Visualization and Clustering

The second notebook, *song_dataset_visualizations.ipynb*, visualizes and performs K-means clustering on the data set we utilize in our recommendation algorithm (Section **??**). We first visualize features of the datasets as a whole (especially over time) and then visualize the outputs of K-means clustering algorithms on the songs and genres datasets. Although Spotify offers access to the data of what seems like every streamable song, genre, and time period, in order to perform visualizations and clustering, we used a smaller data set with over 170,000 track entries found at `https://github.com/AmolMavuduru/SpotifyRecommenderSystem/tree/master/data`.

Our visualizations on our data set were quite fascinating. For example, we generated a plot of various audio features over time, which can be seen in Figure 2. In the plot, acousticness drives down over time, and danceability drives up. Liveness, however, remains relatively constant throughout time, which is intuitive.

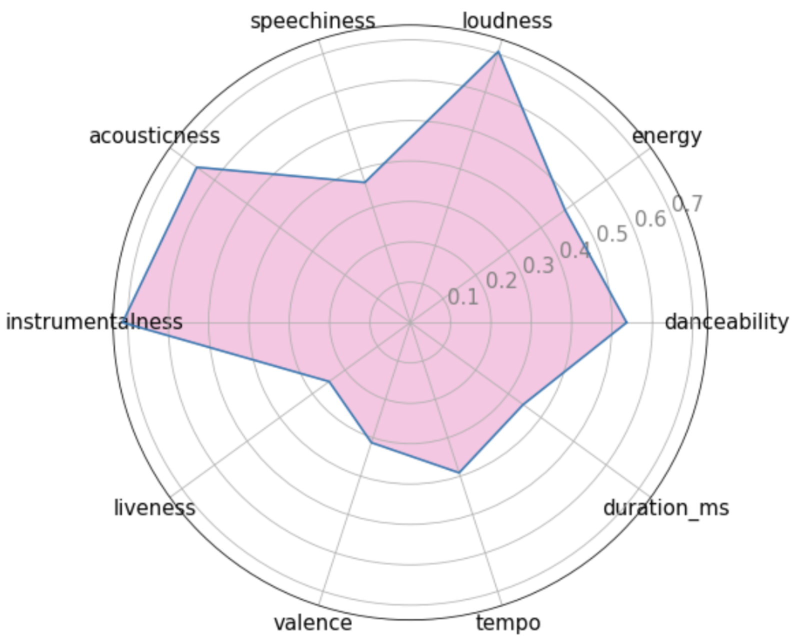

Figure 1: The audio features polygon graph for a playlist titled "...beethoven?". Note the high instrumentalness, acousticness, and loudness, as well as the low speechiness—this makes sense in the context of instrumental music.

Beyond data set visualizations, we clustered the 170,000+ songs into 19 clusters (based on similar features) using the unsupervised K-means algorithm. After clustering, we generated a plot, where each song's features were reduced to two dimensions using principal components analysis (PCA) and plotted on the xy-plane. Songs that are closer to one another (and have similar cluster color) are more similar to one another. This plot can be seen in Figure 3. Below, we introduce several key concepts used to generate our visualizations.

3.4.1 Feature Representation

As mentioned in Section 3.2, we draw upon a wide variety of audio features to inform our recommendations. The list includes valence, acousticness, danceability, energy, instrumentalness, liveness, loudness, speechiness, and tempo (along with other features such as popularity, year released, and duration). How do each of these features feed into the algorithm? Generally, in machine learning, features must be *encoded* (i.e. represented) in a certain way compatible to the parameters of the model, but encoding in our case is simple: most of our features take on continuously real values, e.g. any real value between 0 and 1. Based on the numerical nature of our feature values, we can simply represent each song i as a d-dimensional feature vector $\mathbf{s_i}$, where

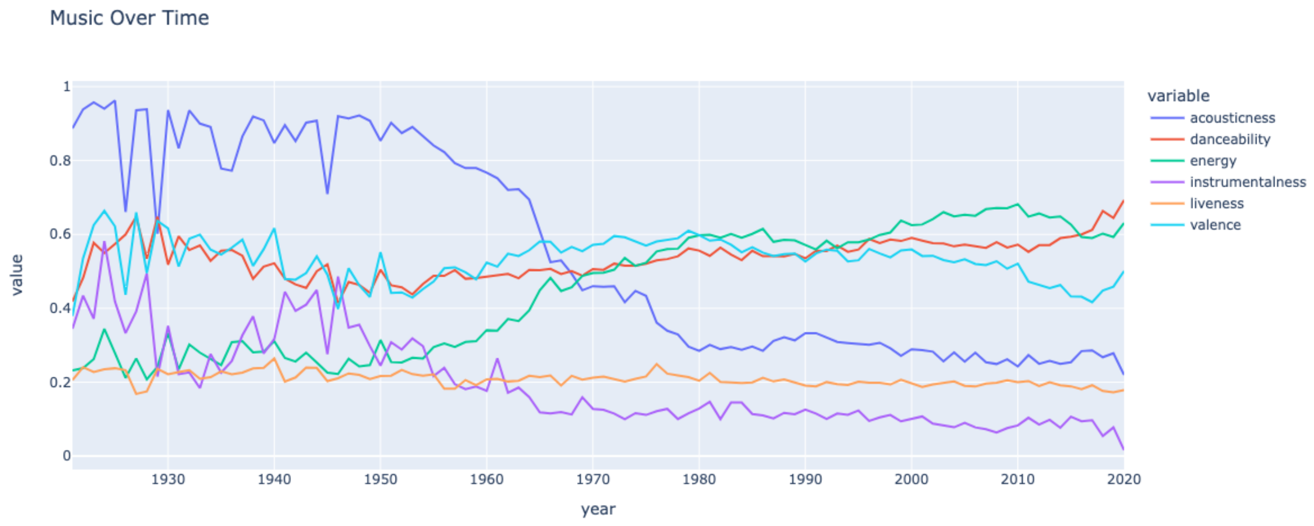

Figure 2: Music audio features in our data set plotted over time, from 1920 to 2020. Upon analysis, one can see acousticness tends down over time, and danceability tends up. Liveness, however, remains relatively constant throughout time, which is intuitive.

d is the number of features:

$$\mathbf{s_i} = (a_1, \dots, a_d).$$

It follows that the distance between two songs' feature vectors determine how similar they are: the closer the distance, the greater the similarity.

3.4.2 Clustering: K-Means

The K-means clustering algorithm is a form of *unsupervised learning*, which is when we work with unlabeled data, i.e. data without pre-labeled groups. This algorithm takes in k mean values and iteratively clusters datapoints according to those mean values. At each iteration, the algorithm computes the means of each cluster, calculates each cluster's distance to each of the data points, and updates the cluster means as necessary. At the end, data points are then identified as part of the cluster to whose mean they are closest. This process is repeated until a standard of convergence is met, for example, when we see no further changes in the cluster assignments. At a high level, the K-means algorithm finds groupings among messy data, and songs within a group are in a sense most "similar" to one another and most "dissimilar" to songs in other clusters.

3.4.3 Dimensionality Reduction: Principal Components Analysis (PCA)

Since the dataset of song features is of such large dimension, it is essential for algorithm efficiency that we narrow in on only the data that contribute to the uniqueness of each song, through which we can more effectively extract insights. One effective way to handle this high-dimensional data is using an unsupervised learning technique known as Principal Components Analysis (PCA). PCA, in short, is a dimensionality reduction technique that generates low-dimensional datasets from high-dimensional datasets (i.e. a smaller data array that is freer of noise and easier to work with) that still preserves the majority of information from the original dataset.

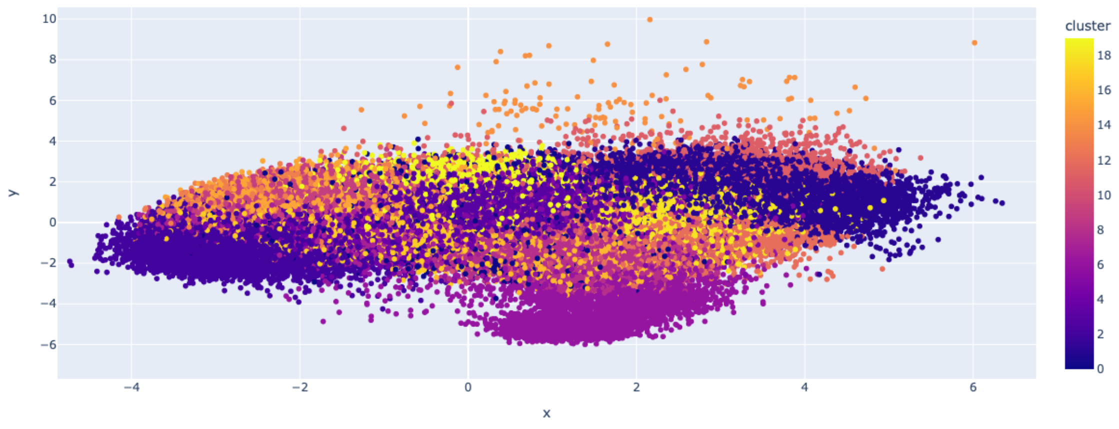

Figure 3: K-Means clustering on our song data set of size 170,000+. Songs that are closer to one another (and have similar cluster color) are more similar to one another. To interact with this visualization and see which songs belong to which clusters, run `https://deepnote.com/project/MusicMatch-aXccbqduS3WAQ0CbCHtrcA/%2Fsong_dataset_visualizations.ipynb/#00028-8fc29d2f-b0b2-47ff-9a59-528a53bcb31f`

3.5 Notebook 3: Song Recommendation Algorithm

The third notebook, *recommender.ipynb*, is a song recommender. Given a Spotify username and a playlist, we introduce an algorithm that outputs song recommendations based on and similar to songs seen in the playlist. We draw upon existing datasets of 170,000+ songs, 2,900+ genres, and information about songs over time to run our algorithm.

3.5.1 The Algorithm

Given our d-dimensional vector representation of each song, we compute the average vector v of all audio features for each song in the playlist. Next, we find the k-closest data points in our data set, being sure to ignore songs that are already in the user's playlist. Finally, we take these k points and recommend the corresponding songs associated with each. This follows a version of *k-nearest neighbors*, which is a popular machine learning technique that involves finding the k nearest data points to an input point.

However, the *distance* to our data point can be defined however we like. Since our vector is an average of all of our songs, we will find points of *similar configuration/ratio* of audio features rather than the standard Euclidean distance. To do this, we use the *cosine distance* (measuring distance based off the angle between vectors), which is defined as follows:

$$distance(u, v) = 1 - \frac{u \cdot v}{||u||||v||} = 1 - \cos\theta$$

We can see that if two songs are so similar such that $\theta = 0$, our distance will also be 0. We utilize the package Scipy's `cdist` function to compute this distance in the code implementation of our algorithm. In order to test the algorithm, we simply ran it on several of our own playlists to see if the results checked out (they did, on the most part!).

As an example, we go back to the "...beethoven?" playlist. Below is the list of songs generated based on that playlist:

1. String Quartet No. 8 in E Minor, Op. 59, No. 2 "Rasumovsky": III. Allegretto (Arranged by Barry Lieberman) by Ludwig van Beethoven, The American String Project (2011)

2. Andante spianato et Grande polonaise brillante, Op. 22: Grande Polonaise in E-flat Major. Molto allegro by Frédéric Chopin, Janusz Olejniczak, Tadeusz Strugala, The Warsaw Philharmonic National Orchestra of Poland (1995)

3. Sphärenklänge, Walzer, Op. 235 by Josef Strauss, Christian Thielemann, Wiener Philharmoniker (2019)

4. Variations On A Theme From Pachelbel's Canon In D Major by David Lanz (1991)

5. Concerto For Flute And Harp, K. 299; 2nd Movement by Wolfgang Amadeus Mozart, Sir Neville Marriner, Academy of St. Martin in the Fields, William Bennett, Osian Ellis (1984)

6. Flute Concerto No. 1 in G Major, K. 313: II. Adagio non troppo by Wolfgang Amadeus Mozart, Anton Gisler, Werner Tripp, Wiener Philharmoniker, Karl Böhm (1998)

7. Symphony No.6 In F, Op.68 -"Pastoral": 1. Erwachen heiterer Empfindungen bei der Ankunft auf dem Lande: Allegro ma non troppo by Ludwig van Beethoven, Berliner Philharmoniker, Herbert von Karajan (1984)

8. P by Labradford (1997)

9. Piano Concerto No. 5 in D Major, K. 175: II. Andante ma poco adagio by Wolfgang Amadeus Mozart, Jenő Jandó, Concentus Hungaricus, András Ligeti (1991)

10. Piano Concerto In E Flat, K. 482, 3rd Movement by Wolfgang Amadeus Mozart, Sir Neville Marriner, Academy of St. Martin in the Fields, Ivan Moravec (1984)

For reference, the original playlist is here: `https://open.spotify.com/playlist/5pHwuHhfHopWnOH9IhjueA?si=b002f0edf7724486`. This playlist mostly consists of Beethoven's Fourth Piano Concerto in G Major and Third Piano Sonata in C Major. These track recommendations are generally apt: most are instrumental and by Classical period composers such as Beethoven and Mozart, while at the same time, we receive tracks by more contemporary composers such as Ligeti that are outside the comfort zone of the playlist. It is also fascinating that many of the recommendations are concertos in major keys, just like our original playlist.

Note that due to the fact that we are referencing a limited set of songs from our data set, our recommendations are limited to the songs found in the data set.

3.5.2 Spotify's Algorithm and Ours

We mentioned earlier that Spotify's Web API already supports generating song recommendations for a user: `https://developer.spotify.com/documentation/web-api/reference/#/operations/get-recommendations`. The API's function takes in information about a user's top artists, genres, and tracks and outputs a list of track information objects. Of course, we could directly call the API's function, but we were curious about how such an algorithm could potentially work under the hood (that still draws upon existing user data, such as the audio features the user prefers in a particular playlist). Based on what we see in the example in the previous section, our recommender seems to output fairly relevant results that the user is hopefully nearly as likely to enjoy as the real Spotify algorithm's output.

4 Connections to GENED 1080

4.1 A New Approach Towards Audio Feature Analysis

During lecture, Professor Wood demonstrated a way of analyzing the similarity between two audio files, which can often be a grounds for copyright violations. Our method of finding similarity between songs uses more discrete approach—mapping a set number of already given audio features with certain values and comparing these features between songs. Both methods are more resistant to some conditions than others. If we analyze by values extracted directly from the .mp3, we can avoid any reliance on assigned audio feature values, which we assume to be accurate in our algorithm. However, this approach may fail to consider key differences between two distinct pieces of music, such as acousticness or instrumentalness. By using the assigned audio features, our algorithm takes advantage of these features and is able to explore qualities beyond comparing the bit-representation of the .mp3 files.

4.2 A New Application of Song Similarity Analysis

In class, we explored similarity from the perspective of copyright and building a copyright detection algorithm, where two songs may be classified as similar if their feature representations are similar to one another. In our project, on the other hand, if two songs have close enough feature representations, one song might be outputted as the recommendation based on another song, or vice versa. Both use cases for an algorithm like this are important in the context of music production and streaming.

5 Project Analysis

Overall, we found this project successful in that we 1) explored and directly worked with an API that we found intriguing, 2) built several tools that are meaningful and interesting to potential users of these tools, and 3) applied both concepts we learned in class and our own interests and backgrounds. what worked what didn't, etc.

With that being said, we wish we could have explored various stretch goals. Had we been given more time, we would have written a script generating visualizations, track recommendations, or even a moodboard given a user's *most recently played tracks*, but given the more complex nature of this function call in the API (it required user sign-in and authentication procedures that we found beyond the scope of this project), we decided to not pursue this avenue. Another stretch goal was to host our song recommendation and playlist feature visualization tools on a web server, such as node.js or Flask, for user experience's sake.

However, that would have taken considerable amounts of environment setup and full-stack web programming that would've steered our focus away from the Spotify API. Besides, the DeepNote environment already gives users a sufficiently comfortable environment to use our tools.

Another stretch goal of ours was to create a survey for the user that would ask questions unrelated to their music taste, mapping their responses to audio features and thus recommending music to them. However, we quickly realized the difficulty in creating this "mapping," as we would likely be forced to make large generalizations about people's music taste and our recommendations would likely be off. In addition, this method would require our bank of recommendable songs to be quite small, given that the survey wasn't very large if we wanted to avoid having virtually random recommendations. This would massively limit the power of our recommendation algorithm and would make our recommendations biased to songs we know. As a result, we decided to take a machine learning approach and are pleased with the result.

6 Individual Reflections and Takeaways

6.1 Catherine

I had an awesome time working on this project with Andrew! Since we both study computer science and are interested in working at the intersection of technology and music, this project was a great way to synthesize our interests. Working with the Spotify API was gratifying—it was driven mostly by intrinsic curiosity rather than, say, a looming problem set deadline. Of course, there are stretch goals I wish we could pursue, but that does not mean we cannot continue working on this project after this class finishes. We applied specific topics from this course (such as audio feature analysis and copyright detection), but beyond that, I felt like I applied the overarching mindset this course gave me of approaching creative problems from a technological standpoint, and vice versa—and for this, I am tremendously grateful. While I wish we covered more topics relating to software + music (in the form of lectures or labs, perhaps), I thoroughly enjoyed the interdisciplinary and nuanced topics already covered in the course.

In general, Andrew and I worked roughly equally on all parts of the project, both in terms of coding and writing. In terms of my specific contributions, I researched the machine learning ideas used in this project (K-means and PCA), introduced us to Spotipy (the Python wrapper for the Spotify Web API), and partially set up the playlist-to-recommendation and playlist-to-visualization pipelines.

6.2 Andrew

I truly enjoyed the experience of working on this project with Catherine! Throughout the entire process, I learned a great deal—including the possibilities provided by Spotify's Web API. I had no idea the extent to which Spotify allowed anyone to access the behind-the-scenes of the platform. As Cat and I researched the best methods for recommendation algorithms, I learned about different popular methods used by recommenders, one of which being k-nearest neighbors. In addition, I discovered qualities of songs that I never would have considered a critical piece of metadata until seeing it in Spotify's API. For example, I never thought about how speechiness would be a huge factor in distinguishing songs from one another. From the course itself, I used a great deal of intuition gained during the week of lectures on music copyright and the Spotify guest lecture. I found it fascinating comparing the algorithm that Spotify uses—which was described as using artists rather than genres and audio features to recommend songs—to our purely audio feature recommender. In addition, we heavily relied on a mindset prescribed by Prof. Wood, to utilize modern methods of solving problems and apply them to music! In retrospect, I do wish we spent a more time checking out music recommendation algorithms as I am personally very fascinated in this, but I still thought the guest lecture and brief discussion in lecture was amazing!

Between Cat and I, we worked fairly equally on this project with our energy concentrated in different parts. I spent a good deal of time setting up the Spotify Web API (getting access tokens, etc.) as well as our initial exploration of the Spotipy package in our Jupyter Notebook, as doing some sort of project involving the Spotify Web API had been a goal of mine for a long time. I also worked on many of the visualization components seen in our notebooks and this report. Due to Cat's higher level of experience with some of the paradigms we explored in our ML algorithm, she did a good amount of the research involving the development of our ML algorithm.

References

[1] `https://towardsdatascience.com/how-to-build-an-amazing-music-recommendation-syster`

[2] `https://spotipy.readthedocs.io/en/2.19.0/#examples`

[3] `https://stmorse.github.io/journal/spotify-api.html`

[4] `https://developer.spotify.com/documentation/web-api/reference/#/`

[5] `https://towardsdatascience.com/an-attempt-to-construct-spotify-algorithm-9ac21ae1e`

[6] `https://github.com/AmolMavuduru/SpotifyRecommenderSystem/tree/`
`master/data`