MAT 167 Report: Final Draft

JP Maestas

September 2025

Introduction

In an effort to dampen the use of illicit web scraping and API queries to train AI models, many organizations have removed these legal protocols altogether. One such organization is Spotify, which, as of recently, has disabled many features that allow users to obtain audio analysis from songs they query. This feature is the backbone for Spotify's recommendation algorithms, and its depreciation has led to a vacuum in open-source software for audio analysis. In an effort to reverse engineer this software, I am creating a clustering algorithm to define a metric used to sort music based on its attributes. I will further use this to determine a recommendation within the set based on a new input's euclidean distance from a PCA vector.

Dataset

The data set I chose to create these clusters are by courtesy of the Free Music Archive. I chose the small (7.2 Gb) dataset which consists of 8000 individual 30 second MP3 tracks. When we open one of these files, it is read within the time domain. That means that we perceive the signal's amplitude as its intensity (loudness) over time. However, if we use the Fourier transformation, we can decompose this signal into the frequency domain. This allows us to obtain core information about the music such as its centroid, bandwidth, Rolloff, and flux.

The centroid, the perceived brightness of the music can be denoted as

$$C = \frac{\sum f_k X_k}{\sum X_k}$$

This is the weighted mean of the frequency with respect to amplitude over the k recorded data points. The Rolloff is generally interpreted as the frequency which is beow the cutoff point $P \in (0,1)$. This measure depicts and eliminates noise from the data. This is given by

$$\sum X_k \geq P \sum X_k$$

The Bandwidth is generally interpreted as the spread of magnitude with respect to the centroid. This is given by

$$R = \sqrt{\frac{\sum X_k (F_k - C)^2}{\sum X_k}}$$

If we look at its formulation, it is reminiscent of the standard deviation of the sample's magnitude (loudness) with respect to the centroid.

Finally, the flux measures the change in the audio's magnitude over time. This is denoted as

$$F(t) = \sqrt{\sum X_k(t) - X_k(t - \Delta t)}$$

where Δt denotes the change in time. For more information on these measurements, I found this resource very helpful.

Methods

When we listen to music, the frequency component of the sound waves encodes how we perceive the noise. These sound waves are oscillations of matter which we perceive over time. We can describe and transform these waves as an approximation of sinusoidal components to change the data from temporally encoded amplitude onto a basis of frequency.

Let ω denote angular frequency. This is the rate at which the wave rotates. Separately, t denotes time, and $i = \sqrt{-1}$. The significance of this term is that it allows us to encode a separate dimension (the imaginary axis) into our approximation. This is significant because sound waves are inherently three dimensional objects. So, we use the relation $e^{ix} = cos(x) + isin(x)$. With all of this information, we can finally decompose our signal into the frequency domain. This is given by

$$F(j\omega) = \int_{\infty}^{\infty} f(t) e^{-i\omega t} dt$$

However, although sound waves are continuous, we can only encode discrete data into a computer's memory. Therefore, we will approximate this transformation using Discrete fourier Transformation (DFT). That is,

$$F(j\omega) = \sum_{k=1-\frac{N}{2}}^{\frac{N}{2}} f(k) e^{\frac{-i\omega t}{N}}$$

Then, we use MFCC (Mel-Frequency Cepstral Coefficient)

$$Y_t[m] = \sum_{k=1}^{N} W_m[k] \, |X_t[k]|^2$$

$$\text{where} \quad \begin{aligned} k &: \text{DFT bin number } (1, \ldots, N) \\ m &: \text{mel-filter bank number } (1, \ldots, M) \end{aligned}$$

in order to create feature vectors P_v of each instance such that

$$P_v \in \mathbb{R}^{\omega t \times 1}$$

then, we will create an augmented matrix of each instance. We will call this matrix J such that

$$J = \begin{bmatrix} P_{v_1} & P_{v_2} & \dots P_{v_m} \end{bmatrix}$$

where

$$J \in \mathbb{R}^{\omega t \times m}$$

Since our data has very high dimensionality, we will use PCA and the k-means algorithm to find k clusters within our data such that the distance of each data point to its nearest centroid is minimized. This will allow us to group features based on their components. We can create a mapping from features to these clusters in order to further use the algorithm as a reccomendation algorithm. Separately, to compare my methods to modern literature on clustering audio data, we will use Spectral Clustering. This process begins by computing a Similarity Matrix of our data. This is a matrix describing a graph where the nodes are audio instances and the edges are the magnitude of their cosine similarity.

This is, given our feature matrix J, we compute the cosine kernel

$$\frac{a^T b}{||a|| \cdot ||b||}$$

for each row instance such that the entries of $W_i j$ are the weights of the edges between two audio instances i and j. More, since we don't want to record the similarity of a song with itself, the diagonal of W must be zero.

Next, we compute the degree matrix. this is the total weight of sums for each audio instance Ji. This is given by the diagonal matrix

$$D = \begin{bmatrix} \sum_j W_{1j} & 0 & \dots \\ 0 & \sum_j W_{2j} & \dots \\ \vdots & \vdots & \dots \\ 0 & \dots & W_{mj} \end{bmatrix}$$

Finally, we compute the Laplacian matrix L such that

$$L = D - W$$

$$J = PBP^{-1}$$

This matrix encodes how each instance is connected to another node in a graph. Thus, our matrix L contains condensed information of the signal instances into a matrix that we can further perform clustering on. For simplicity's sake, we will use the k-means algorithm. The results of both algorithms are depicted below.

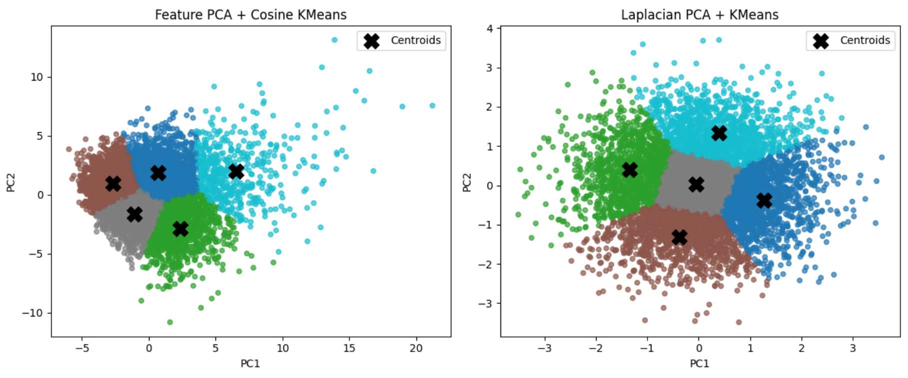

Figure 1: Clustering Using Laplacian Matrix and Feature Matrix

From this, we see that both methods adequately clustered the data, but the scaling of the feature vectors left the data prone to have larger outliers. We have also recorded the following data regarding each cluster

cluster	1	2	3	4	5
mfcc variance	1144.221	881.596	1413.041	1341.132	4406.463
mfcc mean	220.581	-506.668	350.730	84.154	214.254
rolloff	115.775	-506.668	350.730	-208.167	640.711
centroid	62.583	-190.984	131.645	-115.697	-430.471
bandwidth	46.945	-162.135	-83.972	-60.637	390.260

We can use these cluster mappings to group new music by locating the closest cluster (or sample instance) from its values of variance, mean, rolloff, centroid, and bandwidth.

Result

This project is far from over, but I am content with the backbone which I have created in the allotted time. This algorithm groups a large set of music to be further used for either classification purposes or recommendation. In the future, I would like to further implement the k-nearest-neighbors algorithm to find a song within the set that is most-related to a given input. For now, however, I would like to carry forward by creating a visual network of related music. Using the clusters which I have created, I can map the original inputs (songs with metadata on artist and title) to the nodes within the PCA graph. One such example of this is the https://cprimozic.net/blog/building-music-galaxy/. Thank you for your interest in my project. I am very pleased with this opportunity, and I look forward to continually improving my work.