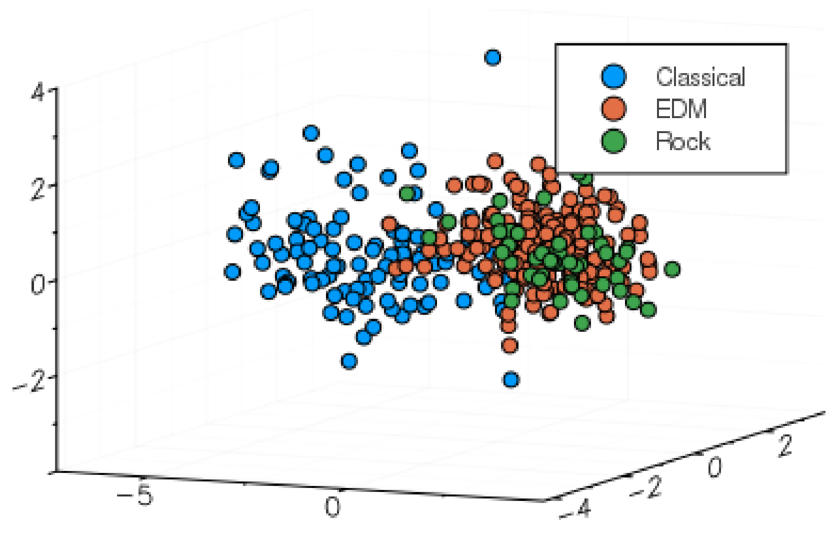
Insights on Song Genres with PCA Analysis of Spectrograms



Gilbert Fan
Student
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: gsfan@andrew.cmu.edu

Edward Li
Student
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: edwardli@andrew.cmu.edu

December 7, 2019

ABSTRACT

Music is an integral part of the lives of millions every day. However, in the age of digital music, listeners have come to expect high quality customized playlists and mixes that present current music that a user might like. A large portion of this playlist customization involves the categorization of music by genre.

The purpose of our research is to use matrices and linear algebra to analyze the characteristics of three genres of music: classical, EDM, and rock. We will perform a principal component analysis on the data of about a hundred songs per genre to gain insight on what are the primary attributes of each and the features that makes song unique. This will involve first converting each song's audio information into the format of a spectrogram through fast fourier transformation. We will then refine the spectrogram image data so it can be combined with the data of other songs into a matrix to be analyzed altogether. We will take the singular value decomposition of our data matrices to find our principal components, and then project the data onto these principal components to create a graphical representation of our results that we can draw conclusions from.

We find that the most important factor in characterizing these three genres is the amount of bass frequencies being used. The first principal component vector, which represents the bass, by far explains most of the variance in each song's data, and is a defining feature with separates the classical genre from EDM and rock. Thus, song suggestion programs can look to emphasize the relatedness of bass lines with a listener's current preference when considering suggesting a new song, as bass frequencies are the primary identifier of genre for an unclassified song.

Nomenclature

Fourier Nomenclature:

$X(\omega)$ The Discrete Fourier Transformation at frequency ω
R hop size of Short-Time Fourier Transformation. In other words, the spacing between two consecutive STFTs
$X_m(\omega)$ The Short-Time Fourier Transformation at frequency ω centered at time mR
M length of the window function
$w(n)$ window function of fixed length M at time n. For our purposes, this is a Hamming window function
$x(n)$ signal of song at time n

PCA Nomenclature:

A_1 Our 367×288 matrix where each row is a song and each entry in the row is the average spectrogram value at the given frequency for the entirety of the song.
B_1 Our 367×288 matrix that is the column-centered and normalized version of A_1
U_1, σ_1, V_1 Parts given by Singular Value Decomposition on B_1
A_2 Our 367×5760 matrix where each row is a song and each entry in the row is the average spectrogram value at the given frequency for a given $1/20^{th}$ portion of the song.
B_2 Our 367×288 matrix that is the column-centered and normalized version of A_2
U_2, σ_2, V_2 Parts given by Singular Value Decomposition on B_2
$y1$ These are the blue points on the scatterplots which represent where the classical songs fall when projected on principal components
$y2$ These are the orange points on the scatterplots which represent where the EDM songs fall when projected on principal components

$y3$ These are the green points on the scatterplots which represent where the rock songs fall when projected on principal components

1 Introduction

Music is a part of the daily lives of millions each day. In the US alone, there are 61.1 million paying subscribers to music streaming services [1], and this number is growing year over year, contributing to a market worth billions. Therefore, developing music playlists and live mixes tailored to specific consumers, using relatively current music, is critical to maintain user growth and retention. Additionally, these playlists should be equitable to both already established musicians, as well as emerging ones.

Previous work in recommender systems to build these playlists, especially as discussed in class, mainly centered on **collaborative filtering techniques**. However, collaborative filtering techniques often suffer from the **cold start problem** [2]. This problem centers around the inability for collaborative filtering systems to produce reasonable recommendations for new songs and songs from emerging artists, due to the lack of user ratings on new songs. The optimal recommender system would ideally be able to recommend both well-established artists as well as completely new ones, as long as it matches a user's preferences. It is clear that a good music recommender system must be able to use the song content itself to recommend new songs to users. One of the most important features of song content is labelling every song by its genre.

The purpose of this project is to use linear algebra to develop a proof-of-concept genre analysis tool. that can both reveal some insights about song genres and demonstrate how a song genre classifier might work. First, we will explore different ways to turn songs from three genres (Classical, EDM, and Rock) into workable data and use them to produce a set of training data from a list of pre-labelled songs through **fast fourier transform** and spectrogram based approaches. We will then break down the data into a **principal component analysis** in order to create a better visualization of what characterizes each genre and gain insight into the features of each genre.

2 Methods Overview

Code can be found at https://github.com/mooey5775/song-classification.

2.1 Genre and Song Selection

Genres were intentially selected to have sufficiently distinct styles. The three genres chosen were classical, electronic dance music (EDM), and rock. 103 classical music songs were selected from the internet [3]. Additionally, 216 EDM and 48 rock music songs were chosen from a DJ pool.

2.2 Song Preprocessing

To convert songs into a useful format for analysis, multiple techniques can be used. While taking raw samples of song data would be the easiest, the high resolution of sample data (22050 per second) would produce too much data to efficiently run analysis. Additionally, the physical representation of waveforms would likely not result in a fruitful analysis.

Therefore, we turn to the scientific literature surrounding song analysis, most using convolutional neural network based approaches, to determine how best to preprocess songs for analysis. A wide variety of literature centers around converting songs to the **frequency domain** for analysis. In particular, most papers use **discrete short-time fast fourier transform** to create a **spectrogram** of song data, which is a frequency vs. time graph for each song. Additionally, the frequency scale is often modified to form a **mel-spectrogram**, which analyzes frequencies with a log-scale to better replicate our human log-response to sound [4–6]. Details about this algorithm will be described in later sections.

2.3 PCA Analysis

Mel-spectrograms result in an image 288 pixels in height (288 frequencies) and variable width, depending on song length. In order to run PCA on mel-spectrograms, multiple design decisions and comparisons must be made to use a manageable number of variables.

The first technique used is to average each frequency across the entire song, running PCA on the average levels of each frequency across the song (288-variable PCA). This technique is especially good at focusing on frequency data, but loses information on song structure over time.

A second technique involves resizing the width of the mel-spectrogram into a more manageable size, forming a 288×20 image that is then flattened into a 5,760 variable PCA. This allows both frequency and time to be analyzed, allowing PCA to exploit possible variance among song structure across time. Details about this algorithm will be explained in later sections as well.

3 Mel Spectrogram Creation

The motivation behind creating a Mel spectrogram is to convert the large amount of data in a song into a form that is relevant to our genre processing goal. A large differentiating factor in genres tends to be use of instruments and intensity, which translates well into occurences of different frequencies in the songs of different genres. Therefore, a conversion of our song into a graph of frequencies over time would be a useful compression of our song into something more analyzable. This is called a **spectrogram**.

A **Mel spectrogram** is another transformation of our spectrogram such that it better matches human perceptions of sound. Because humans perceive sound on a log scale, it would make sense for the frequency axis to be log-scale as well. We perform this transformation after we produce a linear spectrogram initially.

3.1 Definition of Short-Time Fourier Transform

The underlying concept behind the Mel spectrogram is the time-domain to frequency-domain transformation that is performed. In other words, the first step of producing a Mel spectrogram is the **Fourier Transform**. We begin with the definition of the normal Discrete Fourier Transform [7]:

$$X(\omega) = \sum_{n=0}^{N-1} x(n) e^{-i\omega n}$$

In this equation, N is the number of samples in time to take for the Discrete Fourier Transform, $x(n)$ is the signal at time n, and ω is the frequency sample we are taking. These are all defined in the nomenclature section.

However, the Discrete Fourier Transform will sample across the entire song, which removes all time information from our song. Our genre analysis is interested in not only frequency, but frequency and time. Therefore, we use the **short-time fourier transform** instead to retain this information. The Short-time Fourier Transform is extremely similar to the Discrete Fourier Transform. However, instead of taking the Discrete Fourier Transform on the entire song, we only perform DFT on a small window of time which we slide across the entire song, performing DFT once per window of time.

Mathematically, we implement this by introducing a new function $w(n)$ called the **window function** (discussion of window function selection will occur in section 3.2. $w(n)$ is a zero centered "mask" of the input signal with size M. To slide our window function across our song, we will translate our song so that our window is centered at 0, then multiply it sample-by-sample with our input song. This gives rise to the short-time fourier transform (STFT) [7], which is just the Discrete Fourier Transform run on the *input signal multiplied by the window function*:

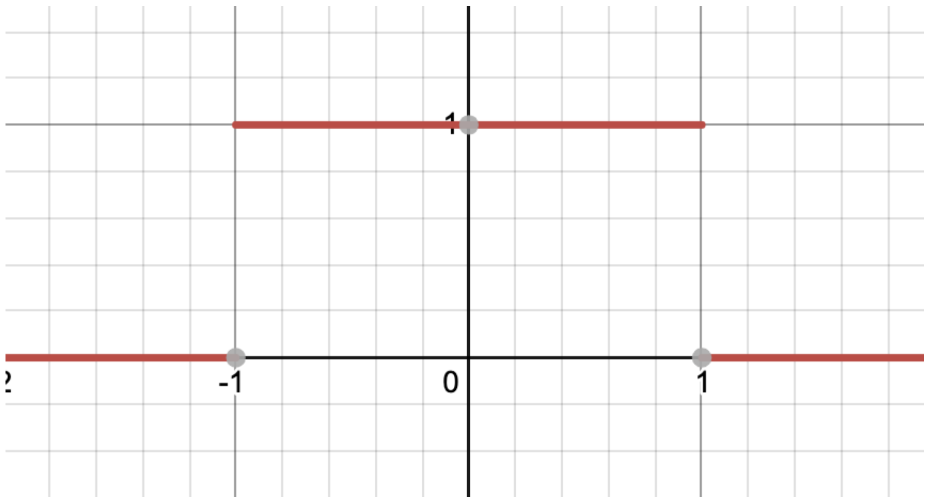
$$\boxed{X_m(\omega) = \sum_{n=-M/2}^{M/2-1} x(n+mR)w(n)e^{-i\omega n}}$$

This is the short-time fourier transform for frequency ω at time mR. More precisely, R is the amount we slide the window by each time, while m is our current offset. When computing STFT, m starts at 0 and continues until the end of the song is reached.

M is the size of our window function. Since our window function is centered at 0, it is defined in $[-M/2, M/2]$. Therefore, STFT runs Discrete Fourier Transform, sampling between $-M/2$ and $M/2$, as can be seen in the summation limits.

3.2 Window Selection for Short-Time Fourier Transform

The most obvious choice of a window for the STFT is a rectangular window. In other words, the most obvious choice is a function that is 1 in $[-M/2, M/2]$ and 0 elsewhere. This produces a function that looks like:

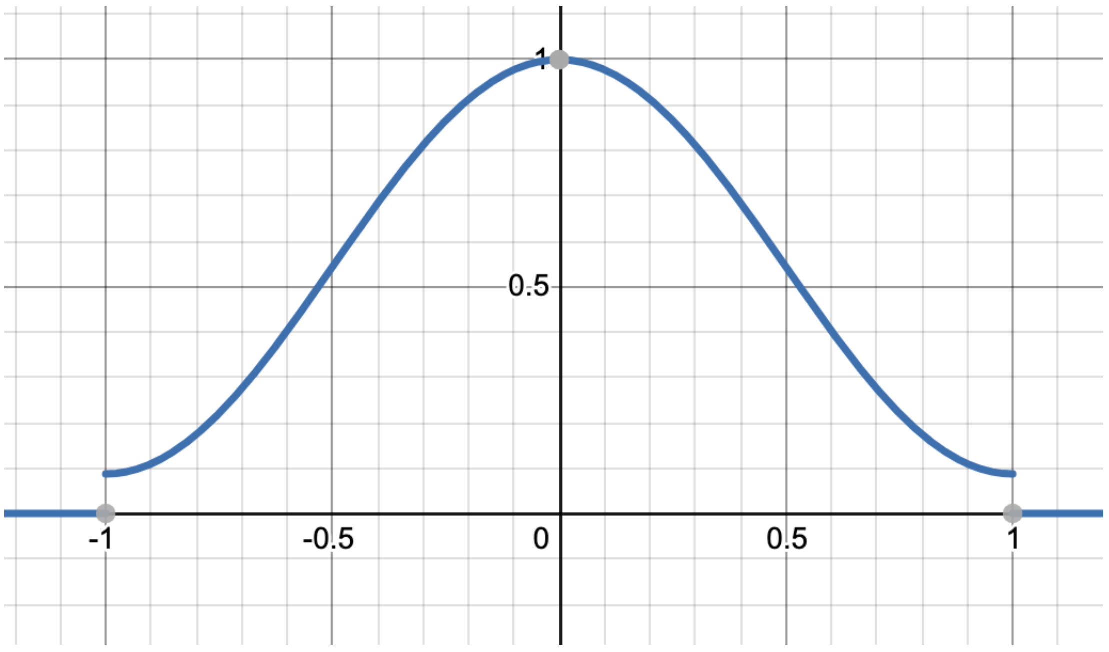


However, while this does give quite good frequency resolution, it is commonly found to create large side-lobes, which can be undesirable (due to the sharp dropoff from 1 to 0) [8]. More concretely, this means that frequencies that do not actually appear in the data can appear in the results of Discrete Fourier Transform.

Therefore, we instead use the **Hamming window**, which is a portion of a cosine curve, raised by a constant. To minimize side-lobes, the Hamming window has constants [7]:

$$w(n) = \frac{25}{46} + (1 - \frac{25}{46})\cos(\frac{2\pi n}{M})$$

where M is the window size. Additionally, $w(n)$ is truncated to fit within $[-M/2, M/2]$



3.3 Computation of Short-Time Fourier Transform

An implementation of the short-time fourier transform is out of the scope of the project. However, the general idea is as follows, implemented in pseudocode:

```
M = 4096 # filter window size
R = 512 # hop size

# define w(n) as window function

for song_file in songs
    song = load_song(song_file) # Step 1
    for m in sliding_windows
        offset = m * R
        song_frame = song[offset : M + offset] # load song frame for
    length of filter window
        song_frame = song_frame *. w # elementwise multiplication of
    song_frame with window function

        frequencies = norm((fft(song_frame))
    end
end
```

More concretely, we conduct the following steps:

1. Load the song into memory
2. Chunk the song into small windows
3. Apply FFT to each window
4. Take the norm of the complex result to get the real magnitude of FFT

The **linear algebra application** of this method comes in with FFT, which uses the fast method of computing the multiplication by the fourier matrix that we discussed in class.

Unfortunately, we were unable to load the songs into memory with Julia 1.0. Therefore, the spectrogram creation steps were done with Python and librosa. However, all data analysis (section 4) is done with Julia.

In Python, we load the songs into memory with:

```
song, sr = librosa.load(os.path.join('music', genre, song))
```

Then, we complete steps 2-4 with a built in function from librosa:

```
D = np.abs(librosa.stft(song, n_fft=M, hop_length=R))
```

This yields a 2D matrix for us, with rows being each frequency and columns being time points.

3.4 Conversion from Amplitude to dB

After completing FFT, we get the amplitude at each step. However, the amplitude of the waveform is not an accurate way to represent perception of sound. Humans hear sound power on a log scale. Therefore, we must convert our amplitude to dB, a log measure of power.

To do this, we convert amplitude to power (power = amplitude^2) and then power to dB (dB = $10 \cdot \log(\text{power})$). This means that:

$$\text{dB} = 10 \cdot \log(\text{amplitude}^2)$$

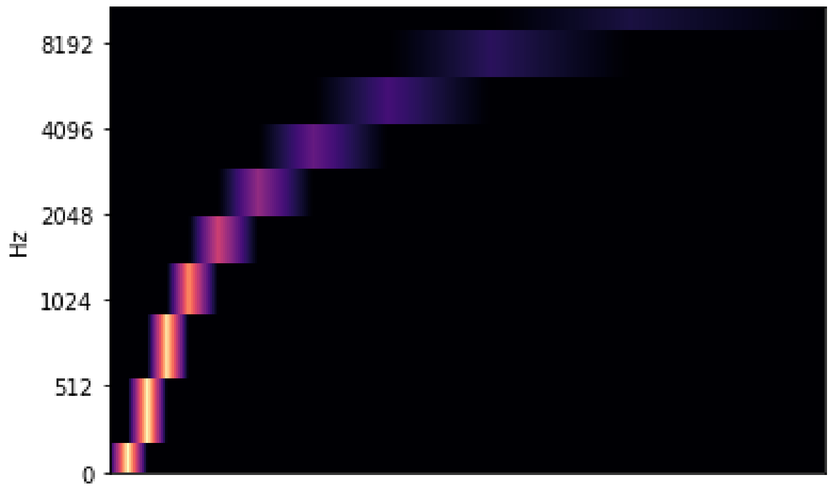
We accomplish this in code by running the function:

```
DB = librosa.power_to_db(D, ref=np.max)
```

3.5 Mel transformation

Finally, we would like to convert our spectrogram into a mel spectrogram. This involves bucketizing our frequency scale with a log curve. Increasing the number of buckets used will give us finer resolution.

We can implement this by getting a linear transformation matrix L, which will transform our completed spectrogram into a mel spectrogram. To gain a better intuition of this transformation L, we will look at an example L with only 10-buckets (low resolution):



We can clearly see that this forms a log scale. To convert our spectrogram into a mel spectrogram, we simply run:

$$\text{mel spectrogram} = L \cdot \text{spectrogram}$$

This, as a matrix multiplication, acts almost like a permutation matrix does. The top few rows will select a large portion of the high frequencies from the spectrogram, while lower rows will select smaller slices of low frequencies from the spectrogram, forming a mel spectrogram.

This can be implemented as:

```
mel = L.dot(DB)
```

In reality, we skip the multiple steps of first computing the spectrogram, converting it, then running a mel transformation, because librosa has a built-in mel spectrogram function that completes all steps in one line:

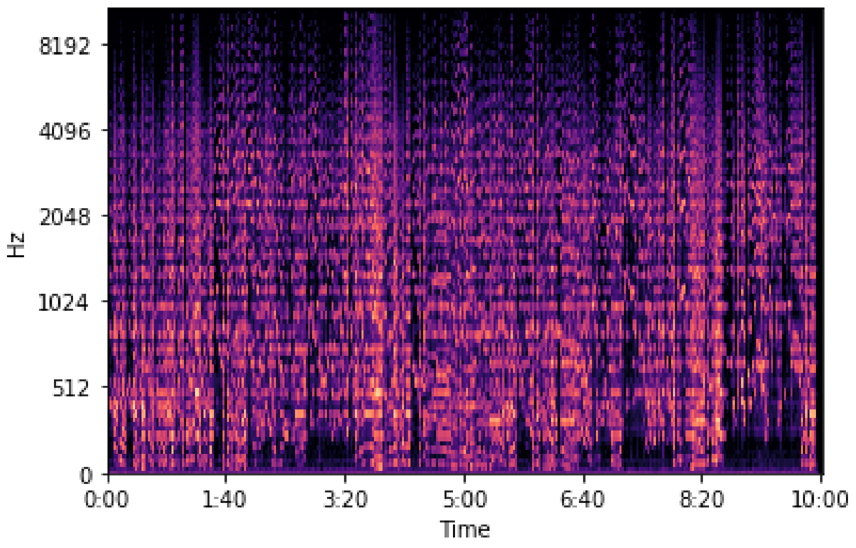
```
mel = librosa.feature.melspectrogram(song, sr=sr, n_fft=M, hop_length
=R, n_mels=128)
```

In our code, mel spectrograms are exported into image files for every song in our dataset.

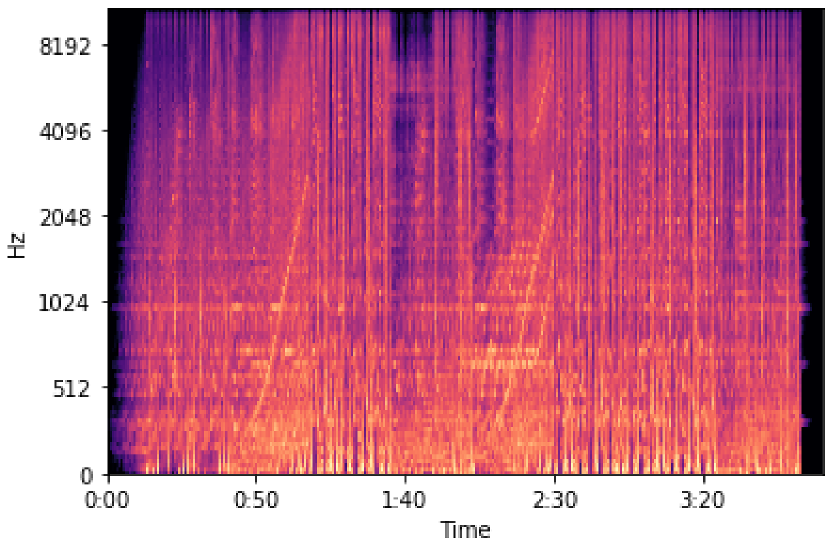
3.5.1 Example mel spectrograms

Let's view a few example mel spectrograms to get an intuition of what they look like before moving onto the next section:

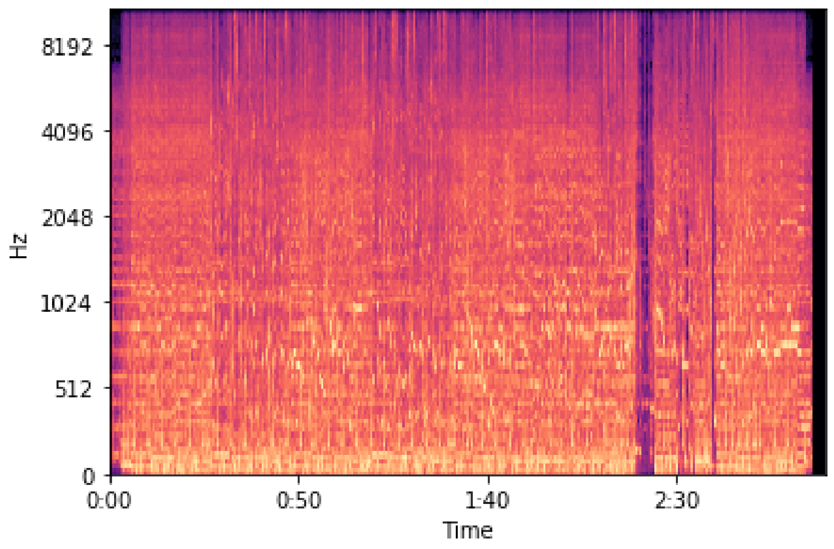
First, we look at classical:



Now, an EDM example:



Finally, a rock sample:

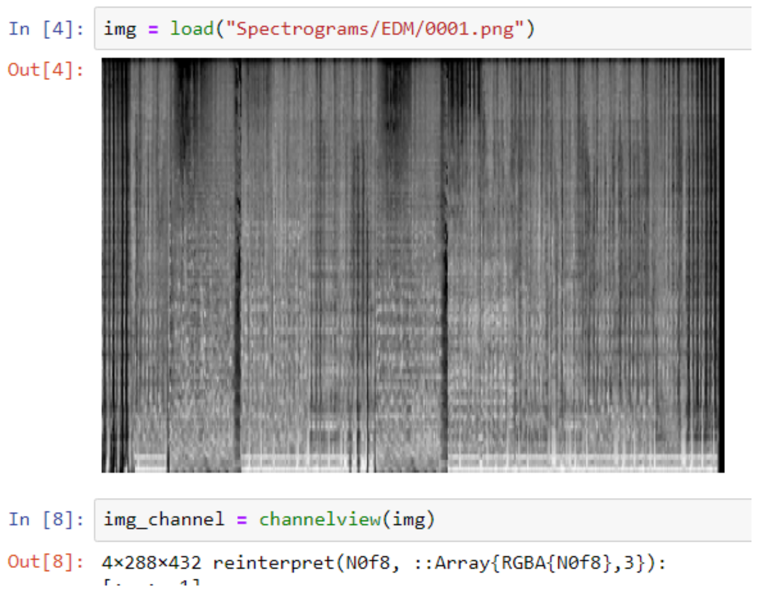


We can see that classical begins as relatively sparse, with little bass. On the other hand, rock is extremely dense, with loud noise. Finally, EDM is somewhere in between. These differences seem promising as we move into PCA analysis.

4 Compiling All Spectrograms Into Matrix Format

After we have created spectrogram pngs for each song, we will need to organize the spectrogram into matrices.

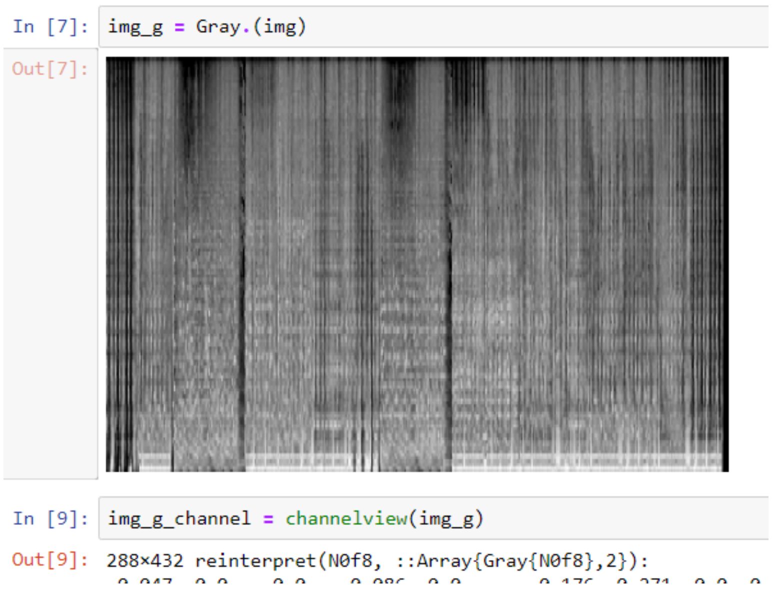
First, we need to turn the spectrogram images into vectors to be put into a matrix. Because images are loaded into Julia with all RGB values intact, it is first necessary to convert the image into greyscale, to get rid of the extraneous 3 channel data. This will also enable our next step, which is to turn the greyscale image into a workable matrix of greyscale values.



```
In [4]: img = load("Spectrograms/EDM/0001.png")
Out[4]:
```

```
In [8]: img_channel = channelview(img)
Out[8]: 4×288×432 reinterpret(N0f8, ::Array{RGBA{N0f8},3}):
```

Notice how when we view channel, we see each entry in this matrix representation has four coordinates. We want to reduce this to just one.



```
In [7]: img_g = Gray.(img)
Out[7]:
```

```
In [9]: img_g_channel = channelview(img_g)
Out[9]: 288×432 reinterpret(N0f8, ::Array{Gray{N0f8},2}):
```

Now we have a matrix where each entry is just one value. By taking the channelview of this grayed image, we also allow this matrix to store negative values which would otherwise make no sense for a matrix of colors.

Next, we need to convert the matrices produced from each image into vector sizes so we can join them all into one data matrix. We will do this in two ways.

First, we can average the entire image's grayscale values by row. The following code creates a vector whose entries are row averages from the channelview of a grayscale image:

```
ra = [mean(im[i, :]) for i = 1:size(im)[1]]
```

We can then use these vectors as the rows of the data matrices. Thus our final result will be a 367 by 288 matrix, where 367 is the number of songs we are using as our data set and 288 is the number of rows in each spectrogram png. We will consider this matrix A_1.

Our second method will also take average of the image by row, but will first divide the song into 20 even sections, and take the average of each section. Julia can automatically divide the image into 20 evenly spaced vertical pieces and take their averages by reshaping the image to have a width of 20 pixels. The following code creates a vector whose entries are the average of the rows of one of the 20 parts of the image, followed by the row averages of the next of the 20 parts of the image.

```
ra = reshape(channelview(Gray.(imresize(load(joinpath(SPEC_DIR,
    genre, image)), (288, 20)))), 1, 288 * 20)
```

We can then combine these vectors as the rows of the data matrices. Our final result will be a 367 by 5760 matrix. Note 288 rows * 20 parts = 5760. We will consider this matrix A_2

The reason we chose to divide the image into 20 parts is primarily due to technical constraints. We wanted to choose the largest number of parts Julia could reasonably compute without significant delays, and after a guess and check process settled on 20. We hypothesize the matrix generated by this second method will produce more accurate results.

5 Principal Component Analysis

Now, we want to create a visual representation of our song data. This is both to determine if there is actually a significant difference in genres to be gleaned from our spectrograms and gain intuition on what factors the most into discriminating genres. Principal component analysis is perfect for allowing us to examine these two ideas. This information could potentially be used in the future to allow for an educated genre classifier.

5.1 Centering and Normalizing Data

Before we begin our principal component analysis, we need to make sure our data is centered and normalized. We only want to focus on aspects of the song directly related to its inherent composition, and limit variance in the spectrogram due to extraneous variables do to the audio file or recording. For example, we want to differentiate songs based on the range of volumes they employ, but not on the fact some sound files may be mastered to shift everything louder. Therefore, centering and normalizing data will ensure we can compare song data with as little influence from post-recording audio-engineering as possible.

We begin with the A matrices, which contain unpolished data. To center our matrix, we need to take the average of a column in the matrix, subtract this average from each entry in that column of the matrix, and then repeat this process for all columns of the matrix. The code can achieve this as follows:

```
A_centered = [A[i, j] - mean(A[:, j]) for i=1:size(A)[1], j=1:size(A
    )[2]]
```

We then want to take our centered A matrices and normalize them. This will involve computing the magnitude of each column vector, and then dividing every entry in the column by this magnitude. The code is as follows:

```
ColLength = [sqrt(sum(A_centered[i, j]^2 for i=1:size(A)[1])) for j
    =1:size(A)[2]]
B = [A_centered[i, j] / ColLength[j] for i=1:size(A)[1], j=1:size(A)
    [2]]
```

Where the first line creates an array of column magnitudes and the second line does the division, creating refined matrix B.

5.2 Finding the Principal Component Vectors

Our B matrices give us information where each song is a row. Next, we want to know which areas of the frequency characterize all the songs the most. If we treat the song's information as a vector, we want to find the unit vector, v, which can be as close as parallel to the vectors of the song's information. In computation, we are essentially trying to maximize the dot product of v and the song's information, thus showing it describes the song the best. Over the course of all the songs, we can sum this up as finding v to maximize the outcome of $\sum_{i=1}^{m} b_i * v$, where b_i is the i^{th} row of B. However, this neglects the possibility that the dot product may be negative, which does not disqualify the vector to represent the data. Therefore, we will instead attempt to maximize $\sum_{i=1}^{m}(b_i * v)^2$.

Note that $\sum_{i=1}^{m}(b_i * v)^2 = ||B * v|| = (B * v)^T * (B * v) = v^T * B^T * B * v$. We know $B^T B$ is a symmetric matrix, which by the Spectral Theorem can be decomposed into $Q * \Lambda * Q^T$. Substituting this into the previous expression we have:

$$v^T * B^T * B * v = v^T * Q * \Lambda * Q^T * v$$

Since Q is orthonormal, we know the products on each side of the lambda, $v^T * Q$ and $Q^T * v$, are also unit vectors. Recognize this means the entire product, $v^T * Q * \Lambda * Q^T * v$ is simply the sum of multiples of the parts of Λ, which is a diagonal matrix of eigenvalues with the largest at the top left. Therefore, we see that we maximize this product when $v^T * Q$ and $Q^T * v$ is the unit vector with 1 as its first coordinate. This also means v is the equal to the first column of Q, by Q's orthonormal property.

Now notice by Singular Value Decomposition on B, we find that $B^T B = V * \Sigma^T * \Sigma * V^T$. Recognize $\Sigma^T * \Sigma$ multiplies to a matrix of the squared eigenvalues of $B^T B$, otherwise known as Λ. Thus, we can see there is an equality:

$$Q * \Lambda * Q^T = V * \Sigma^T * \Sigma * V^T$$

By substituting in Λ for $\Sigma^T * \Sigma$, we get:

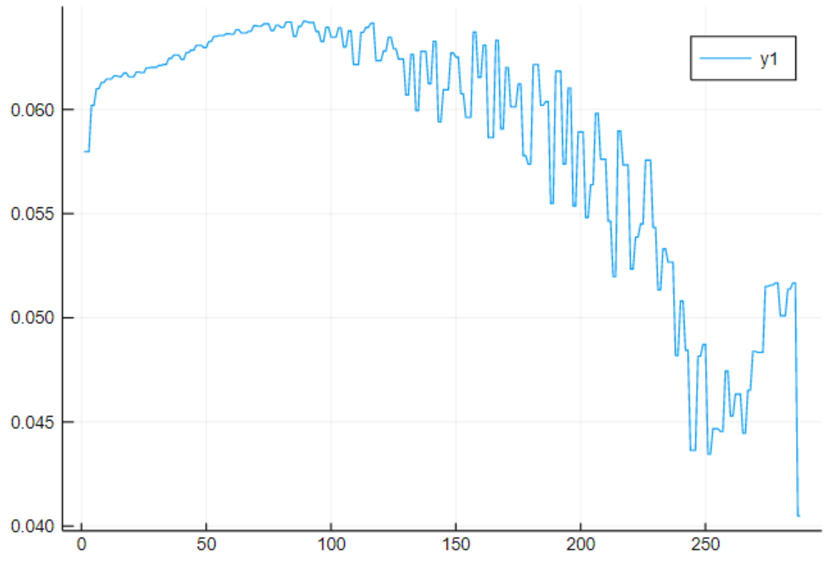
$$Q * \Lambda * Q^T = V * \Lambda * V^T$$

So we see that Q is the same as V, and since v is the first column of Q, then v must also be the first column of V. Therefore, all we need to do the find the proper principal components to describe our data is to compute the singular value decomposition of B and choose however many columns of the resulting V as we would like to examine. Luckily, taking the singular value decomposition in Julia is much simpler than reasoning through principal value decomposition:

```
U, sigma, V = svd(B, full=true)
```

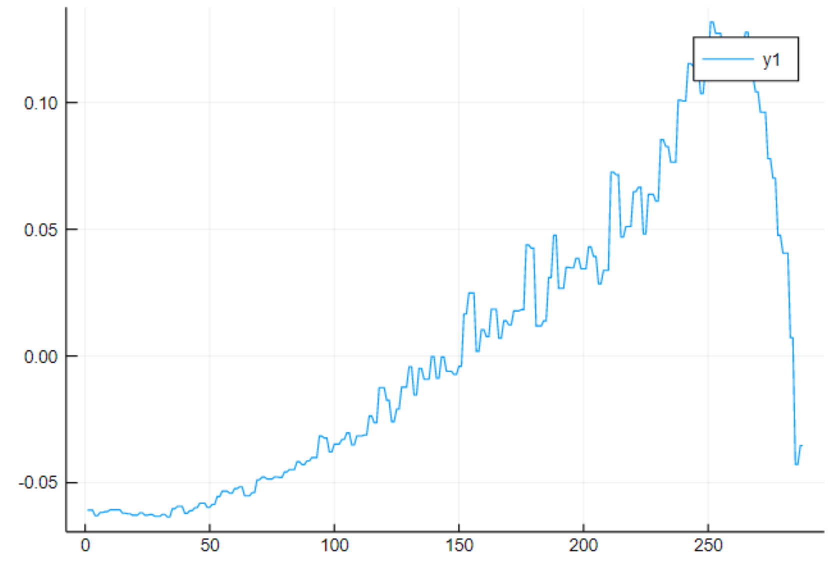
5.3 Interpreting Principal Component Vectors

In addition to analyzing the projections of the songs onto the principal component vectors, looking at the principal component vectors themselves can also give us some insight. To begin with, consider the first column of V_1, v_{11}, taken from the singular value decomposition of B_1. We can graph this vector with the number of the entry on the x-axis and the value of the entry on the y-axis:



Notice the largest values are predominantly on the left end of the graph. These smaller numbers on the x-axis correspond to the lower frequencies of each spectrogram. Therefore, we can see that a large part of the variance in our data is due to the lower frequencies. This makes sense, as classical music in particular has a significantly different use of bass than rock or EDM, and rock and EDM both uses different varieties of bass drums.

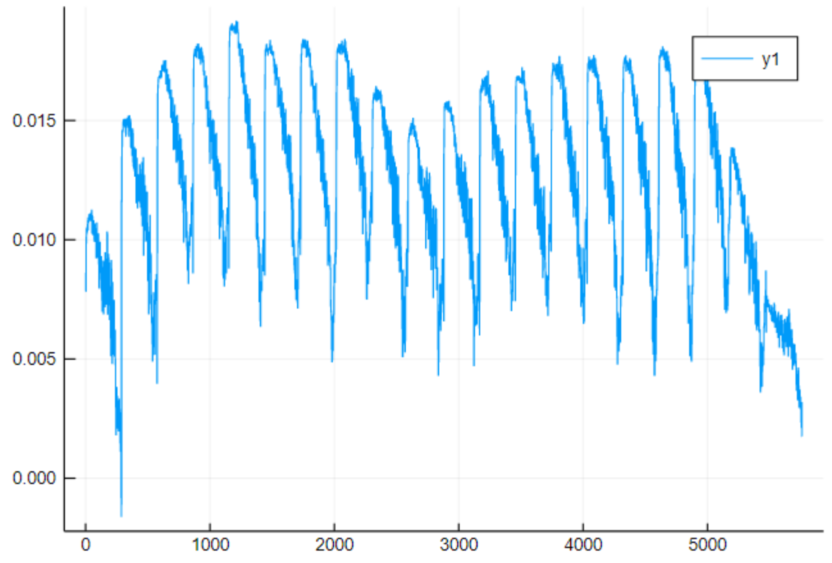
Now lets look at the second column of V_1, v_{12}:



This graph has its largest values on the right side, where high x-values correspond to higher frequency sound. Thus, v_2 seems to correspond to the treble frequencies, which is expected to continue to flesh out the range of sounds in addition to the v_1. Intuitively, it follows that classical, rock, and EDM genres differ on treble sounds after bass, because bass has the highest variance in usage within the three genres. Furthermore, the fact that treble characterizes the genres slightly better than mid ranges is also intuitive as classical employs a large range of frequencies and would likely have outlier songs along high frequencies, while EDM and rock would be somewhat more consistent in high ranges, which rock having the lowest variance in high ranges, which can be gathered by computing standard deviation of the rock music against the average projection on v_2.

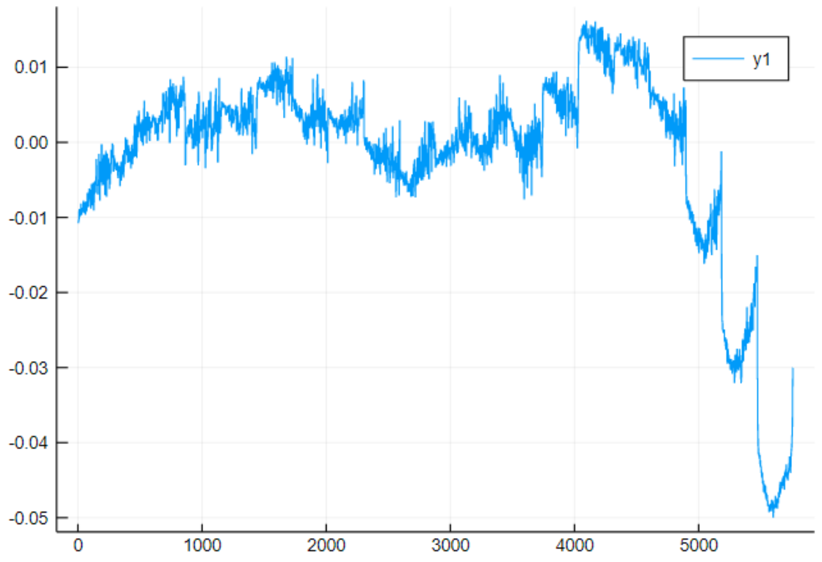
Next, we will examine the first two principal component vectors of our larger matrix, B_2. We will call these v_{21} and v_{22}.

The first principal component graphed by the number of the entry is as follows:



First we must clarify how to interpret the data. Recall the matrices takes the row averages of one of the twenty parts of the matrices before moving onto the next part. Therefore, the x-axis is organized such that there are twenty groups, each group representing a different part of the song. From the graph, we can view these groups as evenly spread along the x-axis, and in this graph specifically they are marked by a significant spike in y-value at the beginning of each group. This gives us a similar conclusion as the graph of the first principal component of the row averages across the entire song. The beginning of each of the groups here is again related to low frequencies at each of the twenty sections, and the large y-values attached to the front of all the groups iterates upon the notion the songs are primarily characterized by bass usage.

The graph of the second principal component, v_{22} is as follows:

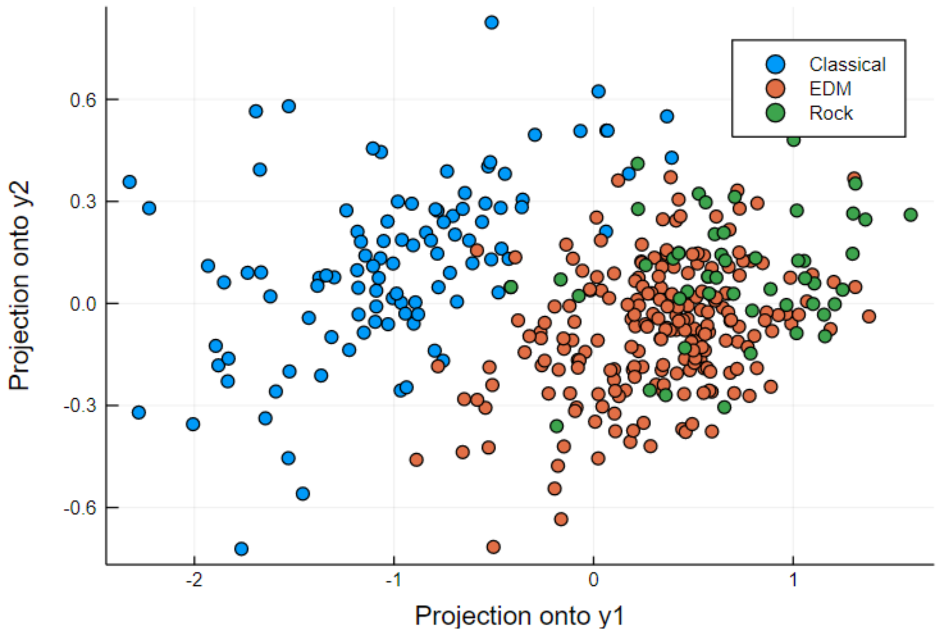


This gives us slightly new insight compared to the other graphs. Recall the graph is still divided into twenty even groups based on time. Therefore, the relatively consistent and low magnitude numbers at the beginning of the graph show that for the most part, every song has a similar beginning and core. The largest part of the variance in the songs, second to the bass, is actually in the last three twentieths of the song. This raises questions because it does not make direct sense for classical, EDM, and rock music to differ significantly based on the end of the song. Possible explanations may be that some of the more modern songs like rock and EDM are released with more immediate endings or the use of a lo-pass filter to remove everything but the bass, before finally fading to quiet. On the other hand, classical music typically has more creative endings that may involve a greater variety of frequencies.

Another explanation may be that some of the songs may be recorded with the end of the music significantly before the end of the audio file. We will further analyze this possiblitiy in the next section.

5.4 Projection on the Principal Component Vectors

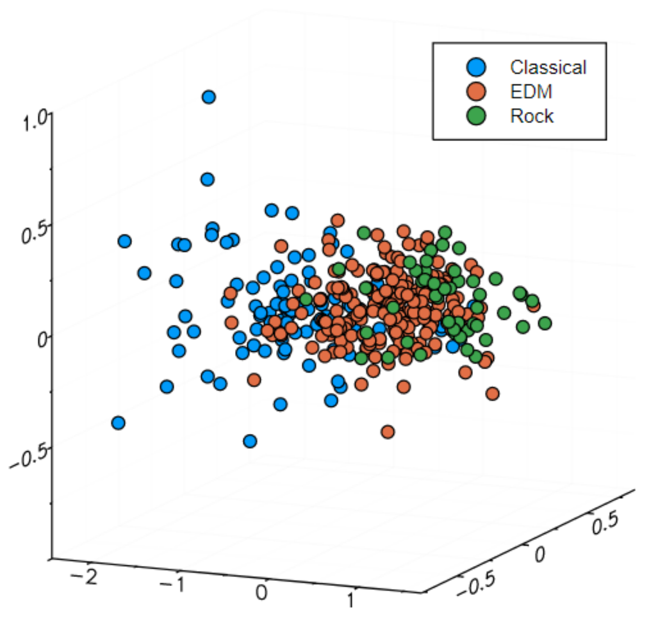
We will begin our analysis of results by looking at the projection of the B_1 onto its first two principal component vectors.



The blue points represent classical songs, while the orange represent EDM, and green represent rock. Immediately, we can discriminate the classical songs from the rock and edm songs in this representation. The blue points are pretty clearly grouped to the left of the graph, whereas the other colors make up most of the right portion. These locations correspond primarily to with the first principal vector, which we will explain in the previous section to be descriptive of bass usage in each song. Thus, the most obvious difference between classical and edm and rock is the bassline, which is also the most prominent difference when it comes to humans listening to these genres of songs.

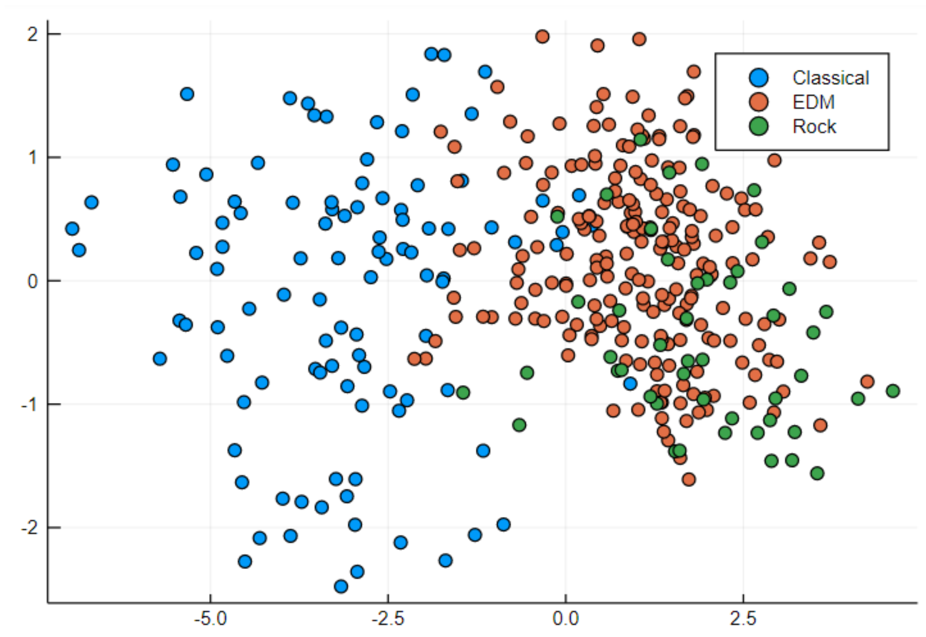
Differences along the y-axes are much harder to explain. Primarily, it seems that classical music lack a distinguishing feature compared to the other two genres when it comes to the second principal component, which we later explain to be descriptive of each song's treble frequencies. This makes sense, since classical music's usage of high and midrange instruments is highly varied and will be very unique for every song. While the projection onto the second principal component may not further distinguish classical songs, it can help us determine the differences between the rock and EDM genres. Taking a look at only the y-values, we see that rock songs trends more toward the positive outcomes when projected onto the second principal component, while EDM is more spread and typically closer to zero or negative values. Thus, we can consider treble to be a distinguishing feature between rock and EDM music. Listening to these genres, it is clear that the instruments used in EDM to produce midrange and high frequency sound is significantly different than those used in most rock songs, thus manifesting this difference in treble between EDM and rock.

The following is the projection of the same B_1 matrix onto its first three principal component vectors:



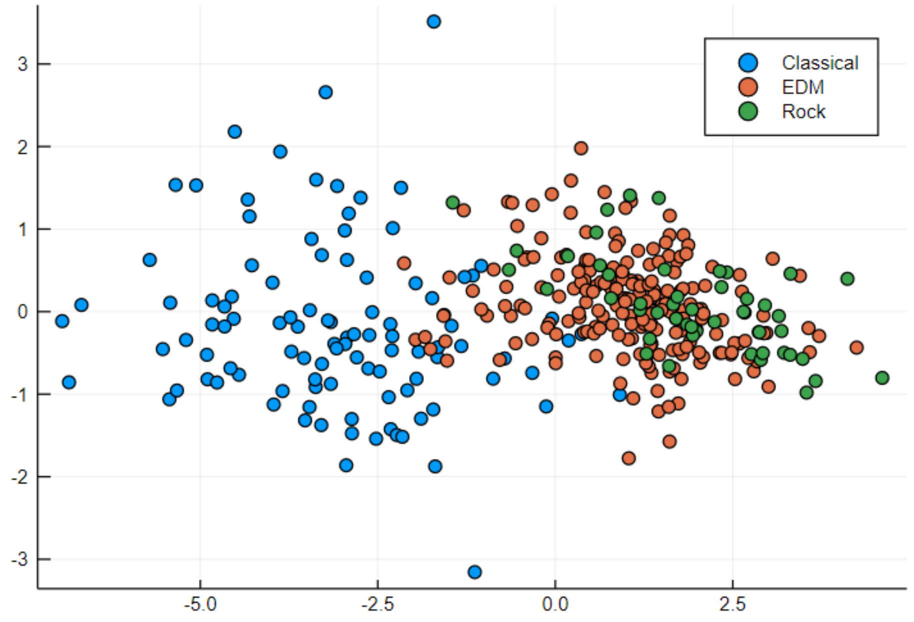
The third principal component's projections lies on the vertical axes, while the first principal component's projections are on the longer of the two horizontal axes. With this in mind, we see that outside of the one or two outliers in the classical genre, this representation of B_1 has most of the songs all grouped around the vertical center, signifying the the third principal component does not really do much to account for variance in our data. This view, however, does allow us to better see the difference between the rock and EDM genres. From this perspective, it is more obvious how the rock songs seems to lie further toward positive results when projected on the second principal component compared to EDM.

Next, we can examine the projection of the B_2 matrix onto its first two principal component vectors:

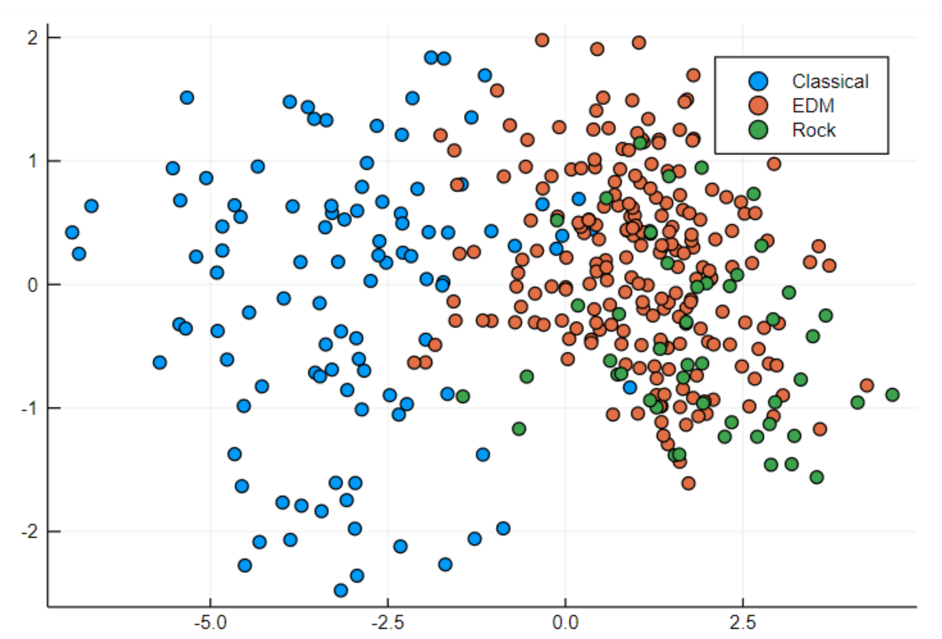


Again, we see rock and EDM song sharing more similar projections while classical songs are more distinct from the other two genres. Most of the differentiation between genres occur along the x-values, or the first principal component. In the next section, we recognize the first principal component is representative of bass usage. The negative x-values of most of the classical songs, then, correlate to classical more frugal usage of bass. EDM, on the other hand, tends to employ a very consistent bass wobble in the background, while many rock songs have bass guitar, thus causing these genres to have more positive x-values, differentiating them from classical.

Along the y-axis, we see less distinguishable trends among the genres. As stated in the previous section, there is a possibility that the second principal component vector relates more with the audio file's length than then actual characteristics of the song itself. Since this is a variable that has more to do with the way the music was recorded rather than its genre, we would want to find a way to overlook it if this is the case. One attempt to analyze our data without considering the variance due to the end of the song is to simply project it onto the first and third principal component, ignoring the second one. We will do this with our matrix B_2 data:



With projections onto the first principal component on the x-axis and projections onto the third principal component on the y-axis, we see the difference in location between the three genre's has not changed by much. In fact, besides classical songs being spread more along the y-axis, this graph has EDM and rock music more densely grouped around the 0. Therefore, we can conclude that the endings of songs may have more of a significance than just being the product of the recording methods.



Since we have confirmed the second principal component to have significant value in characterizing the genres, we can reexamine the data with this knowledge. As we hypothesized, classical music has the greatest range of values along the y-axis, confirming how classical music will employ the most variety of ending. Furthermore, we can recognize how rock and EDM have much more similar y-values, which might correspond with modern music becoming more streamlined in their ending sequences. Rock is also plotted slightly more towards the negatives than EDM, which might suggest how rock songs take longer to end, and might reflect how some rock songs are composed with long guitar or instrumentals at the end of the song, thus causing their endings to look a little different than EDM endings.

It is worth also taking a look at the averages of these genre's projection on each principal component. The following lists each genre's mean for their songs' projection onto the given principal component, followed by the median, and then the standard deviation:

1^{st} Principal Component:

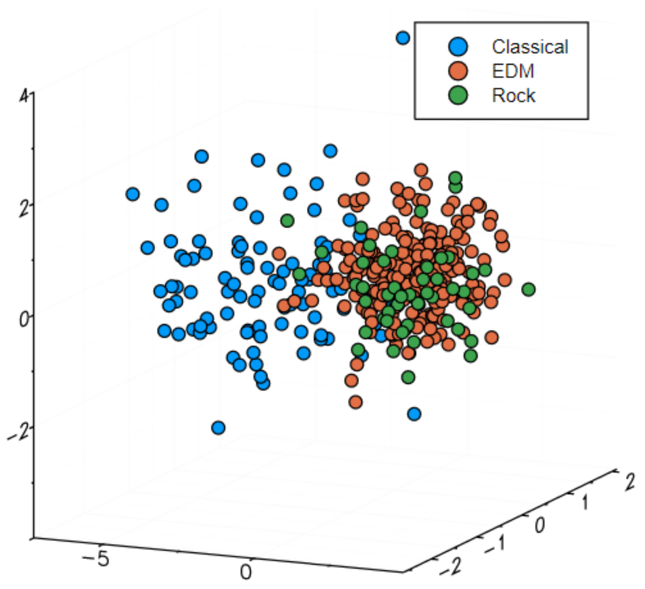
Classical: -3.09, -3.07, 1.56 EDM: 1.05, 1.13, 1.14 Rock: 1.92, 1.92, 1.28

2^{nd} Principal Component:

Classical: -0.13, 0.18, 1.09 EDM: 0.16, 0.18, 0.74 Rock: -0.45, -0.65, 0.75

These values give us a better idea of the differences between each genre. We see that rock is clearly the heaviest user of bass, and there is actually a significant difference between all three genres based on low frequencies. classical music is has the largest standard deviations across both principal components, which is expected as we considered it to be the most diverse genre in terms of instrument usage. A surprising result, however, is that the classical and EDM songs have the same median for the second principal component. Since the second principal component corresponds to frequencies at the end of the song, we can gather that EDM and classical songs have similar endings, which initially seems to make no sense. However, many EDM song outros involve removing parts unique to EDM slowly, often times leaving only a simple instrumental and a beat. This may sound somewhat similar to the instruments employed in classical music, accounting for their identical medians.

Finally, let us look at the projection of B_2 onto its first three principal components:



It would seem that the genres are more mixed together in this version compared to the projection of the matrix of entire song averages. However, looking at the ranges of the axes and comparing them tells us that this version actually details the songs much better and illustrates each song with much higher variance. The twenty cuts of the song is clearly gives the B_2 matrix much more information that taking just one average of the entire song for each song in the B_1 matrix, so this second three principal component representation must be more true to the identity of each song. Therefore, the larger axes ranges tell us that the variance in each song cannot be simply explained by just the first two principal components, and even the third principal component plays a large part in describing the song data. This means each song may unique beyond just their genres, and have many individual characteristics that come into play when analyzing them in this way. This suggests a much deeper analysis is necessary in the future to better quantify the differences between each song, or alternatively that to understand the core properties of each genre we cannot break the songs into too many pieces that will emphasize their individual compositions rather than their genre's.

6 Conclusion

We have gathered some significant information from taking the principal component analysis of songs in the genres of classical, EDM, and rock. From our multiple approaches of considering the data, we see that the most significant distinguishing feature of these three genres is the amount of bass used within them. This fact may be especially useful for classifying the limited-bass classical music from the other two genres. Furthermore, we saw that rock and EDM are most quantifiably distinguished by their treble melodies, and that songs may also have characteristic ways of ending which may not be immediately considered as a factor in revealing genre.

Furthermore, we noticed that as we got more information for each song, the grouping of songs in the same genre became less dense, and there were more outliers found in locations where we would expect another genre to be. This suggests that even songs within the same genre can have widely unique spectrogram data. For future analysis, this means we may either want to try different kinds of methods for converting audio information into matrix data, or we may need an even larger data source for each genre to reduce variance due to outliers.

In terms of practical impact, or results imply that music suggestion programs will want to rely most heavily on suggesting music with similar bass frequencies listeners tend to like. Our results support the musical understanding that the bass provides the foundation for most songs, and while bass parts are not typically memorized by listeners in comparison to treble melodies, knowing what bass frequencies a listener enjoys may be more useful than knowing the melodies they are familiar with when suggesting new tracks.

Acknowledgements

Thanks go to Professor D. Offner for inspiring this final project and instilling in us the beauty of math and linear algebra. We would also like to thank Gilbert Strang for his creation of a wonderful textbook for our education and for having a nice name. Thanks also to Kunal Joshi for help waking us up in the morning and to Konwoo Kim for helping clarify our section headings.

References

[1] U.S. paid music subscribers 2019. (n.d.). Retrieved from https://www.statista.com/statistics/707103/paid-streaming-music-subscribers-usa/.

[2] Ahn, H. J. (2008). A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences, 178*(1), 37–51. doi: 10.1016/j.ins.2007.07.024

[3] 100 Classical Music Masterpieces : Free Download, Borrow, and Streaming. (n.d.). Retrieved from https://archive.org/details/100ClassicalMusicMasterpieces.

[4] Ullrich, K., Schlüter, J., Grill, T. (2014, October). Boundary Detection in Music Structure Analysis using Convolutional Neural Networks. In *ISMIR* (pp. 417-422).

[5] Sandsten, M., Brynolfsson, J. (2017). Classification of bird song syllables using Wigner-Ville ambiguity function cross-terms. In *2017 25th European Signal Processing Conference (EUSIPCO)* (pp. 1739-1743). IEEE.

[6] Pons, J., Slizovskaia, O., Gong, R., Gómez, E., Serra, X. (2017, August). Timbre analysis of music audio signals with convolutional neural networks. In *2017 25th European Signal Processing Conference (EUSIPCO)* (pp. 2744-2748). IEEE.

[7] Smith, J. O. (2011). *Spectral audio signal processing.* Stanford, CA: W3K.

[8] Short Time Fourier Transform (STFT). (n.d.). Retrieved December 6, 2019, from https://faculty.nps.edu/rcristi/EO3404/B-Discrete-Fourier-Transform/text/3-STFT.pdf.

[9] Strang, G. (2016). *Introduction to linear algebra*. Wellesley: Wellesley-Cambridge Press.

Appendix PCA Code