Final Project Report
CS7641

Brian Chuo, Ananth Dandibhotla, William Dingler, Iman Haque, Brian Hill
bchuo3@gatech.edu, wdingler3@gatech.edu, ihaque3@gatech.edu,
bhill81@gatech.edu, ananth.dandibhotla@gatech.edu

***Abstract*— Music is a popular form of entertainment media and enjoy on a daily basis. Whether in the car, at the computer, or on a morning jog, music is likely to be incorporated into your daily routine. Most people have a preference for music. As such music is separated into categories called "Genres" and these genres can be used by the listener to find additional music that conforms to their taste. The goal of our project is to use both supervised and unsupervised learning to classify Music into genres. We then measure the accuracy of each model.**

1 INTRODUCTION

Music is a popular form of entertainment media and something that many people consume and enjoy on a daily basis. Whether in the car, at the computer, or on a morning jog, music is likely to be incorporated into your daily routine. This project aims to explore methods of classifying music into genres based on information associated with a given song.

2 PROBLEM DEFINITION

With music being such an intrinsic part of many of our lives, we likely have the desire to find new music to enjoy. Whether it is a garage mixtape or a workout playlist. Music makes the mundane and boring manageable. The genre of music that a person enjoys can directly depend on a person's taste or the task at hand. You may like to listen to heavy metal such as Iron Maiden's 1984 Album Power slave when working out. Or you may enjoy listening to the blues whilst working on your car during a warm summer day. As you listen to the albums you may desire something new.; Something you haven't heard, but that falls within a

certain genre for which you are passionate? How would you classify massive amounts of music into genres so that you wouldn't have to listen to each song and decide it's genre? The answer is Machine Learning. With machine learning, we classify and sort a large amount of music, far faster than a human can. This is the problem we like to solve and the following sections outline our proposed solution.

3 DATA COLLECTION

In this section, we will discuss the ways we collected our data and how we transformed it to make it useable for our project.

3.1 Test Data Collection

For test data collection, we explored several different options. The key factors we took into consideration were ease of use, speed, reliability, and flexibility. We wanted to create a solution that would let is start working and prototype immediately, while also being acceptable for a final submission.

To begin with, we decided on using YouTube music. This is because there are existing libraries that support downloading of YouTube videos, and YouTube music is effectively a different UI over it. The downside of this, is that YouTube music does not categorize songs or albums by genres.

We tried experimented with finding the samples automatically. This comes at the cost of curation but would have allowed us to sample a large amount of data both in terms of diversity and quantity. Unfortunately, since YouTube music provides no genre labels we would have needed some external database or curation platform to achieve this, and we didn't find an acceptable one.

Our middle ground was manually categorizing albums that we chose with genres. These albums were chosen to both represent their genre and the diversity of them, while also maintaining some resemblance to the sample data we trained on.

From these albums, we created a script that uses a unofficial YouTube music API to write the songs, their genres, the artists, and the YouTube URL to a JSON document that can be parsed later.

Once we established the music we wanted to test on, we started work on automating downloading. With the JSON document of the song and genre classifications, we used PyTube to download the videos. PyTube offers the ability to download through different streams, such as mp4 and mp3. We chose to download with mp3.

From here we used ffmpeg to convert the mp3 of the song to an input that would be acceptable to our model. We specifically went from mp3 to one or more wav files with an audio sampling frequency of 22050. These files were 30 second clips of the song that are pulled from random locations, though we make sure they are not the same clip. The script is versatile enough to take multiple clips from each song, but we chose to only take 1 clip from each song as we had enough songs to make a robust test set.

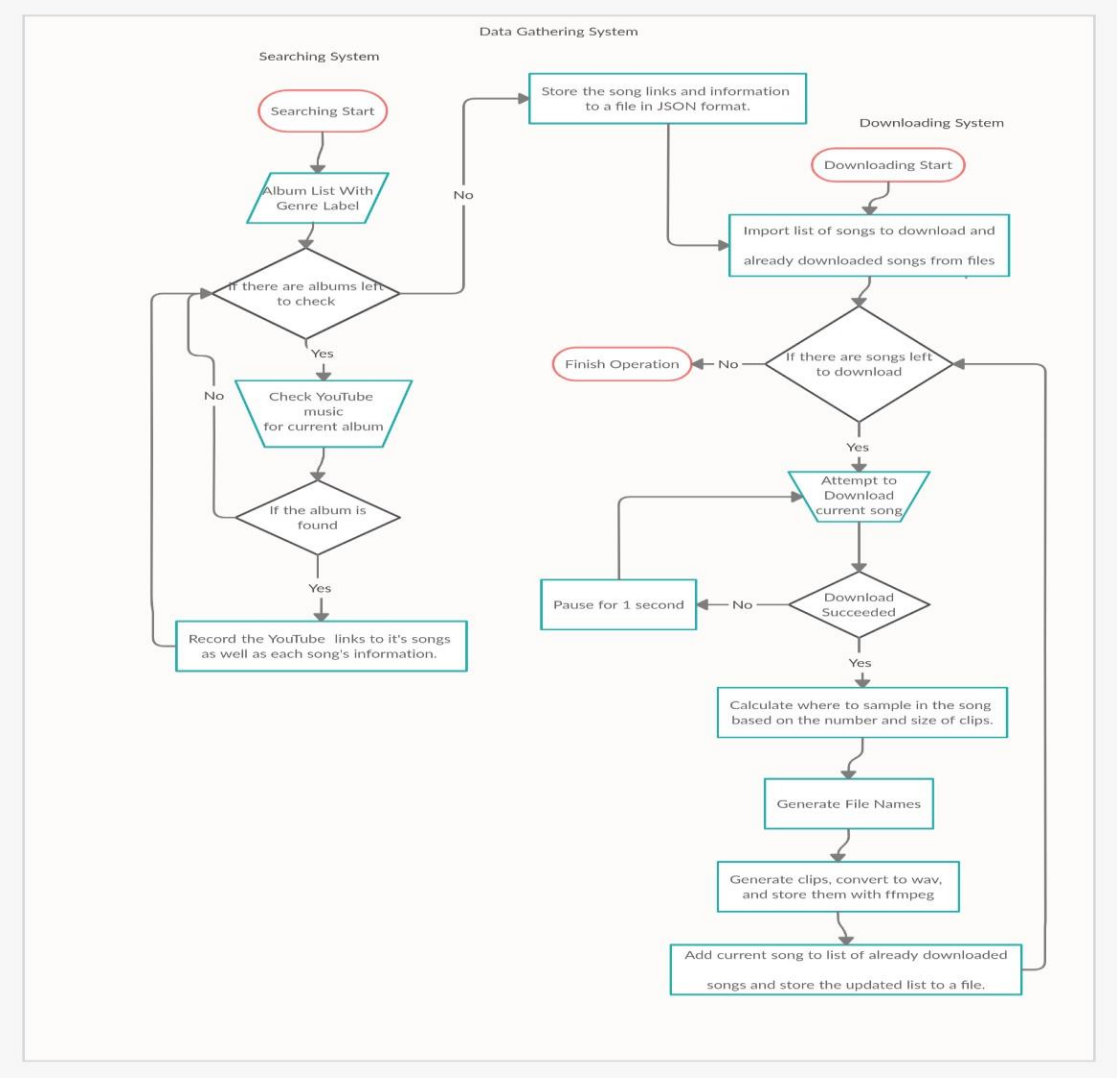
Some issues we encountered included crashing during the downloading phase due to potential rate limiting and unorthodox responses by YouTube. We dealt with this by adding fallbacks to restart operations if needed without stopping the script. Furthermore, we added a tracker that writes to a file to record the already download songs. This meant that the script could be stopped at any time either by crashing or us cancelling operations, and the script could pick up where it left off without any issues and without redownloading anything it had already done.

Another thing interesting to note is the space savings we saved by sampling at an audio rate that is half of the default for ffmpeg (44100 to 22050). This was also done as our test data had an audio rate of 22050.

As a speed metric, we were able to download and convert 1 hour of songs in a little less than 1 minute.

On the next page we have included a very detailed UML Diagram of the data gathering process in order to help the reader better understand the process.

Figure 1— UML Diagram of the data collection process.



3.2 Transforming the data

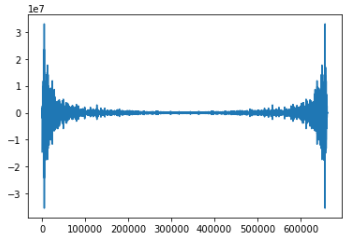
Initially, the data is given to us in the form of mp3. After researching how to analyze and extract the most significant portions of it for classification, we determined that converting these files into wav form was preferable, as it allowed us to use functions in scipy and matplotlib to graph this data, producing images such as this

Figure 2 — Graph after plotting the wav file data in matplotlib

```
rate, data = wav.read('jazz.00000.wav')
```

```
data = data[:,0]
output = np.around(np.fft.fft(data),decimals=5)
plt.plot(range(0,len(data)), output)
# Before slicing data array
```

```
C:\Users\brian\anaconda3\lib\site-packages\numpy\core\_asarray.py:85: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
[<matplotlib.lines.Line2D at 0x23e9f97e160>]
```



Now, this data isn't quite beneficial to us, as we need something that tells us the most important bits of it so that we can use our learning aspect of the project to classify unlabeled genres. We determined that we'd need to utilize a fast Fourier transform, and take the output from it, a 1-D array of floating-point values, and append it to our CSV of data points. Each row represents a single sound clip, and there is a corresponding CSV that contains the labels for these values.

Figure 3 — The data after performing a FFT on a single wav file

```
print(output[:,None].T.shape)
df = pd.DataFrame()
df2 = pd.DataFrame(output[:,None].T)
df = df.append(df2)
df
```

```
(1, 330897)
```

	0	1	2	3	4	5	6	7	8	9	...	330887	330888	330889	330890	330891	330892	330893	330894	330895	330896
0	0.393532	0.093137	0.073648	0.066512	0.134877	0.10798	0.088142	0.021338	0.090255	0.149892	...	0.00083	0.000791	0.000541	0.000331	0.001364	0.001026	0.001017	0.001504	0.000407	0.000581

1 rows × 330897 columns

Much of the difficulty in this phase came from determining how exactly to transform the waveforms into something analyzable. FFTs (fast Fourier transforms) are a commonly used algorithm for waveform analysis, and we decided to use this in our project. This was adapted from Choi et. Al.'s paper on "A Tutorial on Deep Learning for Music Information Retrieval".

Another difficulty was the storage of data. Due to the sheer size of the learning dataset (1000 songs), we needed a way to store all our data into a single space. CSV files were the first that came to mind. With this, we were able to generate the files necessary, though they were unopenable by Excel due to the size of the CSV.

4 METHODS

In this section we will discuss the two methods we used for our project and the results we were able obtain from them.

4.1 Use of a DBN

One of the constraints on our project was the necessary usage of both unsupervised and supervised learning algorithms. When conducting research on the topic, one of the first neural network architectures that came to mind is the Deep Belief Network (DBN). A DBN is a neural network constructed from multiple layers of Restricted Boltzmann Machines. Some neural networks can be too large and cumbersome to easily train, but "the DBN circumvents this problem by performing a greedy layer-wise unsupervised pre-tuning phase" [1].

We chose to use the DBN for multiple reasons. The first being that music information retrieval tasks "depend on the extraction of low-level acoustic features" [1]. Humans cannot choose features that apply to all types of music. There are many features that can be chosen from and usually the most effective features are task-dependent. The second reason that we chose a DBN is that the pre-tuning

phase is a form of unsupervised learning, so this architecture would solve the constraint of using both unsupervised and supervised learning algorithms.

4.2 DBN and Progression to PCA

Although the DBN was consistently performing very well in the unsupervised learning aspect (>99% accuracy), the supervised learning portion of the network would not consistently output values that we would expect. Because the library we were using was from an old github page, we decided to go with a simpler approach to supervised learning.

Since PCA was an effective learning tool for us, we decided to use PCA as the unsupervised portion of this project. We then decided that a Support Vector Machine (SVM) would be a good learning algorithm as it was something that we went over in lecture and everyone in the group was familiar. Our SVM did two things that made us decide it was a better tool for us. Firstly, the outputs we saw from the SVM displayed the values that we expected to see given the labels. Secondly, we were able to get about 80% accuracy using the PCA/SVM combination. We now have a learning tool that can effectively classify between different groups of music, as well as visualize some of the differences in genres.

4.3 Detailing of the process

Prototyping the algorithm with the Kaggle Dataset:

Genre supergrouping: While attempting to learn the 10-genre Kaggle[2] dataset with the DBN, we realized that trying to learn on this many classifications was a difficult task to begin with. We decided to manually agglomerated our genres so that we would be classifying between a few "supergroups". To this end, we categorized "blues" and "metal" under a "rock" supergroup, "hiphop" and "pop" under a "pop" supergroup, and "classical" and "jazz" under a "classical" supergroup. This allowed us to learn the differences between three major music cultural paradigms: classical, pop, and rock music.

Principal Component Analysis:

We initially cut down the 1000-point Kaggle dataset to a 40-point subsample, with 4 examples of each of the 10 genres. This allowed us to prototype our process before expanding the dataset size. The initial conversion to frequency-domain representation of the audio files produce a dataset of a size on the order of shape = (40, ~300000), meaning our data was extremely dimension-heavy. We realized that trying to learn data of this dimensionality would be an expensive task, so we used the unsupervised technique of Principal Component Analysis to reduce the dimensionality. We implemented the PCA with the sklearn toolkit.

The PCA transformation retained 97% of explained variance with 30 components, which allowed us to handle a much smaller data set while observing the meaningful differences in the frequency representation of the music set.

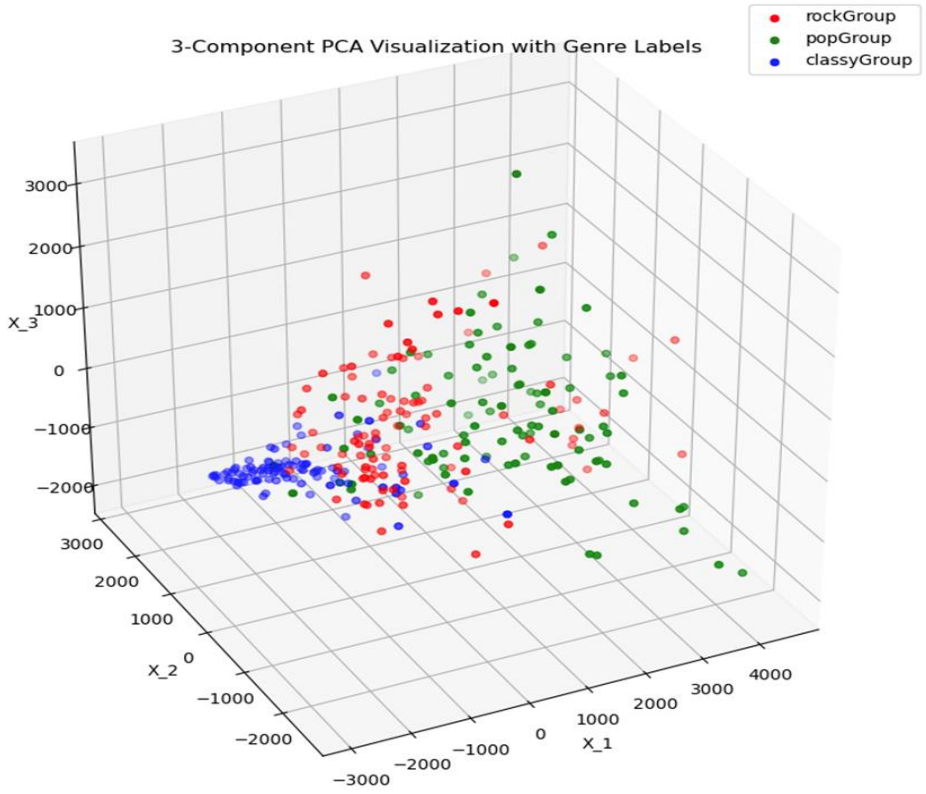


Figure 4— The 3 most significant components returned by the PCA decomposition.

Figure 4 plots the top 3 components of the PCA-transformed songs in the full Kaggle dataset. Marking the datapoints with their supergroup label shows an apparent distinction between the groups. This signaled that our approach did in fact yield a distribution that could distinguish between the genre labels and could be learned on with a supervised model.

While the 3-component PCA transformation produced an informative visual, we found that the marginal difference in explained retained variance between the 2-component and 3-component transformation was only 3.34%, so we decided that the 2-component representation was sufficient for our supervised learning approach. Figure 5 shows the 2-component transformation.

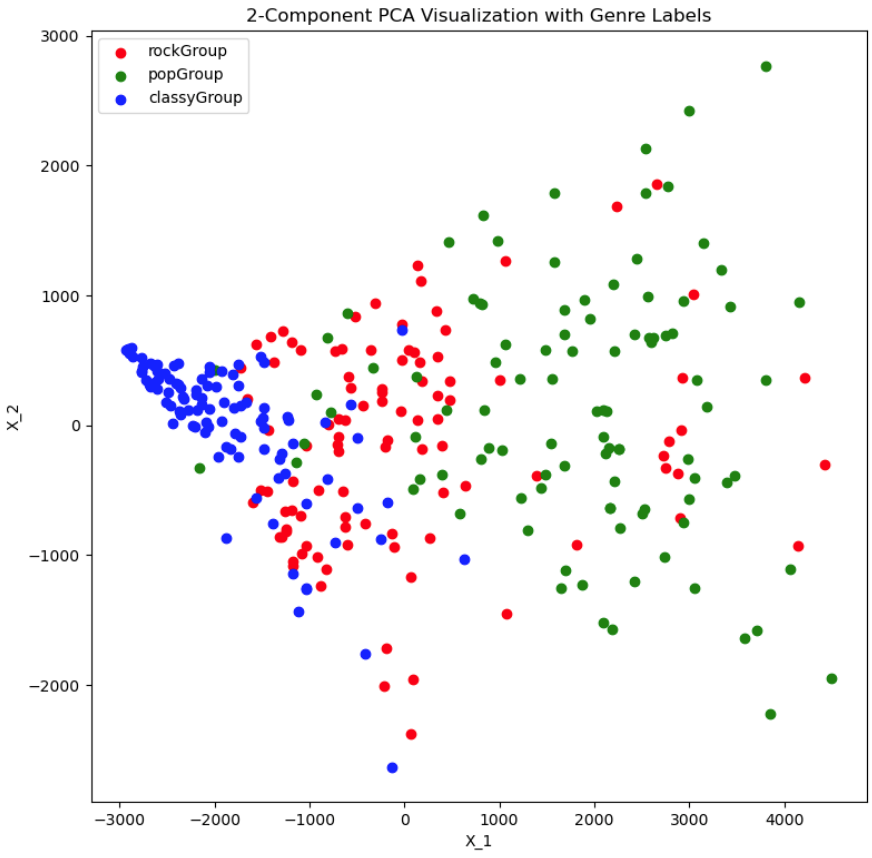


Figure 5 — The 2 most significant components returned by the PCA decomposition.

Support Vector Machine:

After we realized that the DBN was not working the way we needed it to, we decided on performing supervised learning with a more conventional approach, Support Vector Machines. We implemented the SVM package of sklearn, and split the data to test with 20% of the total dataset. The SVM model based on the 2-component PCA consistently yielded an accuracy of 81.66%.

Results:

Our final model is visualized in Figure 6. Again, this model is a prototype based on the readily available Kaggle dataset, which demonstrates a successful implementation of a music genre classifier. We then use our approach on the Youtube-sourced "real" music.

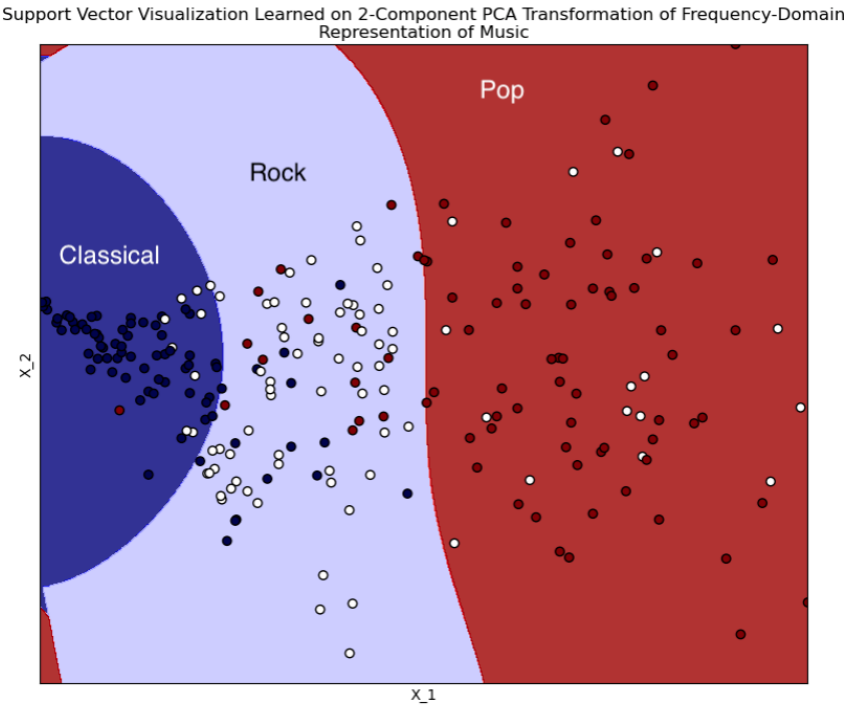


Figure 6 — SVM model visualization, classifying music with an accuracy of 81.66%

Learning on real YouTube data:

Results: The same algorithm as was used on the Kaggle data set was executed on our 1,232 tracks scrubbed from YouTube classified under the same three super-group labels: classical, rock, and pop. We first converted the youtube tracks to frequency domain representation and reduced the dimensionality from 300k to 30, in this case retaining 49.47% of explained variance, significantly lower than in the Kaggle set. The 3 principal components are visualized in Figure 7.

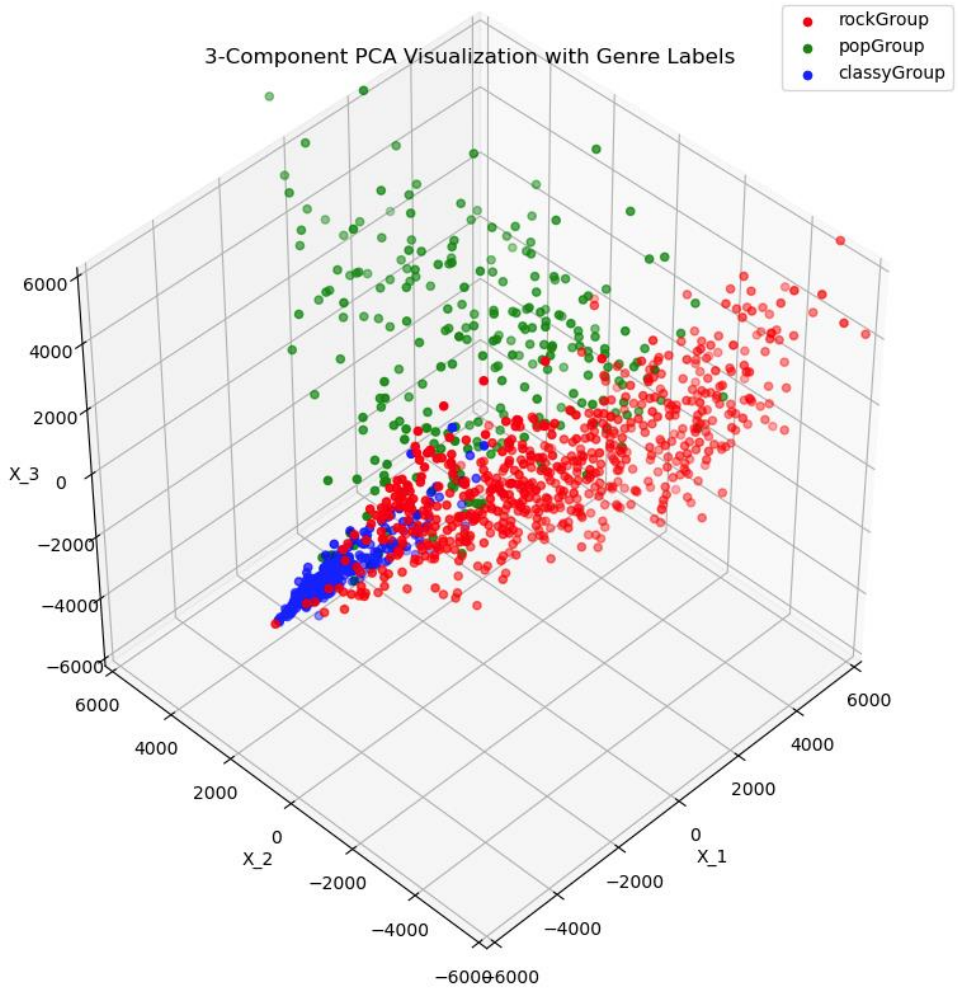


Figure 7— The Youtube music files dimensionally reduced through PCA.

We again tried the Support Vector Machine training with 20% of the data used for testing on both the 2-D and 3-D PCA transformed data, and found that the models yielded and accuracy of 90.68% and 90.28% respectively. Figure 8 visualizes the 2-D PCA transformed data and Figure 9 visualizes the SVM model decision boundaries on that data.

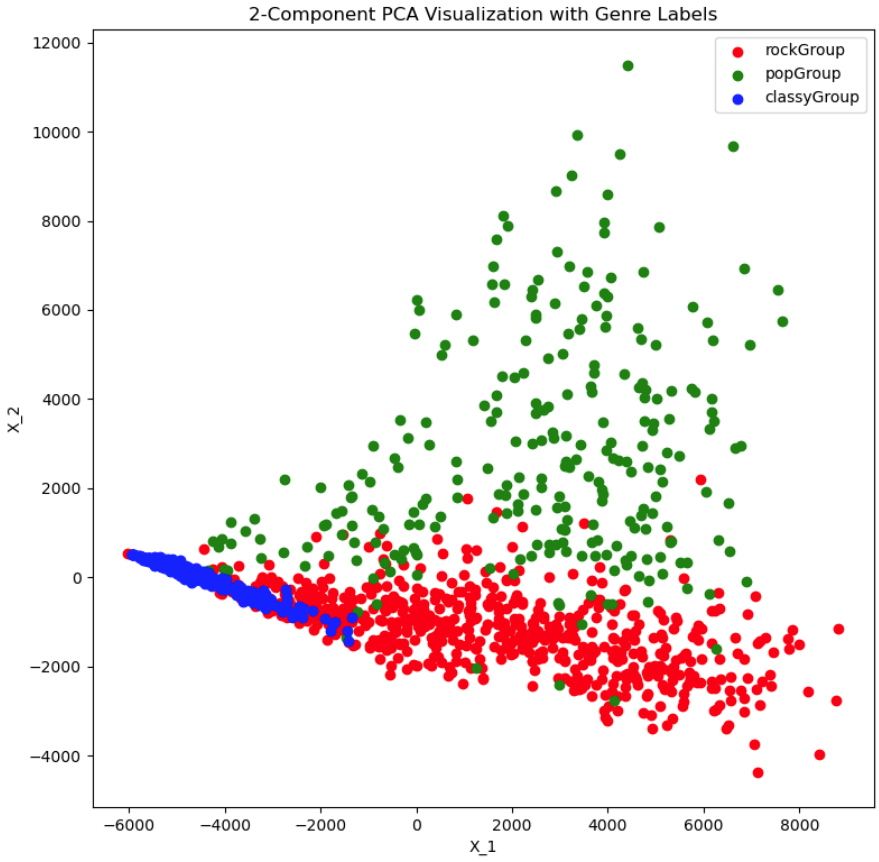


Figure 8— 2D PCA transformed YouTube music data

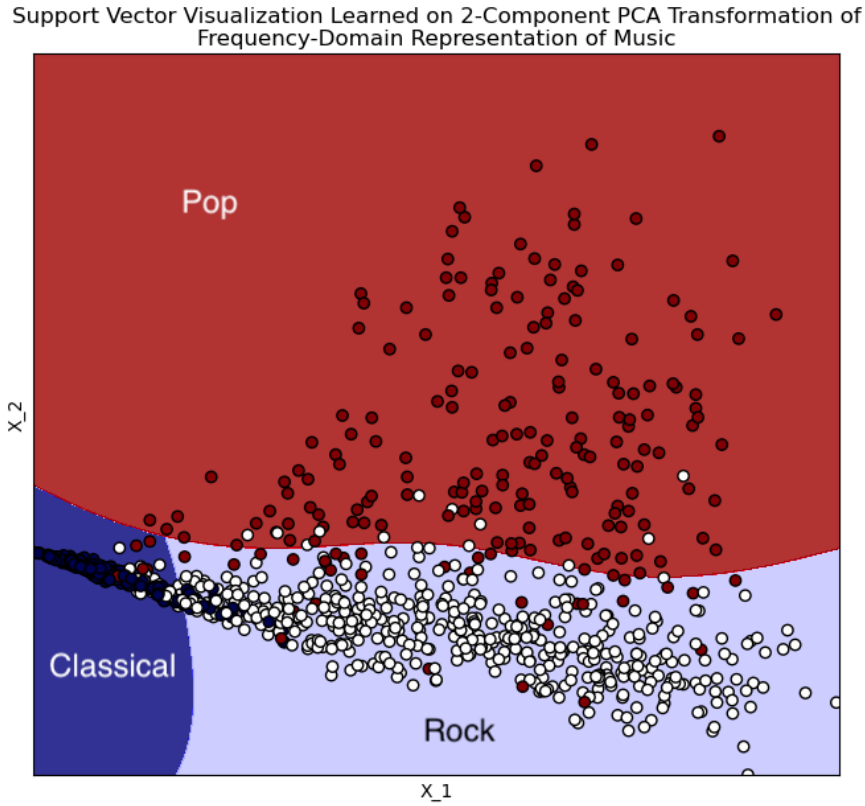


Figure 9 — Support Vector Machine Model with Decision Boundaries

5 DISCUSSION AND CONCLUSION

Although we encountered issues with the project along the way we were able to achieve great success. We were surprised to find out that SVM trained on the YouTube data performed significantly better than the one trained on the Kaggle data (an accuracy gain of 9.02%). We speculate that this has primarily to do with the difference in the data volume we used. However, looking at the distribution of the categories in the two SVM visualizations, some differences in behavior of the data can be seen.

The distribution of the Kaggle data appears to "radiate" outwards from the origin, whereas the Youtube SVM appears to form three distinct "phases". In both cases, it appears that classical music is described by frequencies that tend to reduce to the origin, and the rock and pop categories contain more variances in their frequency distributions. It can also be concluded that there are patterns in the frequency distributions of these classes that allow them to be classified together.

If we had had more time and greater access to more powerful machines, then we could have increased our accuracy. Obviously the download of our music files from YouTube as mp3's causes a loss of quality due to Mp3's inherent compression. This quality will not be gained back when converting the mp3 to a lossless format such as wave. This is due to the fact that the detail much like zooming in on a picture, doesn't exist. This would take a very powerful machine to be able to download many wav files from YouTube in large batch all at once. It would also require fast and rather large storages in order overcome any inherent bottleneck.

We also wish that we could have been able to use the DBN in a greater role, but the issues with getting it to work with real world data made for a lackluster solution. That was something that would have required much more time to ascertain the root of the problem.

6 REFERENCES

1. Hamel, Philippe, and Douglas Eck. "Learning features from music audio with deep belief networks." ISMIR. Vol. 10. 2010.
2. Olteanu, Andrada. “GTZAN Dataset - Music Genre Classification.” *Kaggle*, Kaggle.com, 24 Mar. 2020, www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification.
3. Choi, Keunwoo; Fazekas, György; Cho, Kyunghyun; Sandler, Mark. “A Tutorial on Deep Learning for Music Information Retrieval”. https://arxiv.org/abs/1709.04396