Final Report

Adam Page (V00910612), Nathaniel Coster (V00862066), Vedant Manawat (V00904582)
Shawn Nettleton (V00906570), Jesse Lacroix (V00890406)

Introduction

With the large amount of music continuing to be released and available in a range of audio formats, it has become nearly impossible for a manual conversion to sheet music to keep up. This is the process of music transcription. Automatic music transcription has been a core application for music information retrieval since the first explorations of the subfield. Automatic music transcription typically involves the processes of pitch, onset, duration, and instrument detection. For this project we specifically focused on pitch and duration detection as the base key concepts for our music transcription implementation.

Problem

The problem we are tackling is translating musical audio files (wav) created from guitar recordings and identifying the notes played in order to construct sheet music. The issues we needed to address when we created a solution for this problem where:

- Reading in musical data and processing it for a classification problem.
- Comparing multiple classification techniques and identifying an optimal solution.
- Classifying a sequence of musical data as a specific musical note.
- Writing these classifications to sheet music so that a musician can use it.

This problem is important as sheet music is a crucial document for any musician. If a musician is recording their performance it could be translated into a physical copy to be easily accessible for future reference. This would also be useful for those who want the sheet music for a musical piece that may not have existing sheet music to go with it. Using a solution like this one could provide an individual with sheet music for a song they like so they can recreate it themselves on their instrument of choice.

Existing solutions, such as AnthemScore[4], solve the problems we have identified and show a real world demand for audio to sheet music translators. The problem of digital audio to music note classification is complex, shown by the implementation of tools like AnthemScore who use neural networks trained by millions of data points. Throughout our experiments, it was our goal to come close to commercial solutions such as this one with limited funds and computation power to prove that complex solutions can be solved given solid methodology and procedure.

The dataset used for training and testing our models is a collection of short guitar recordings. The GuitarSet dataset [1] has a solo part and a comp part. The solo is a simple single-note lead and the comp is a mixture of chords and notes. We focused mainly on the solos for our experiments as mixtures of chords and notes can make this classification problem much more complex. Each track within the guitar set has information regarding the timing, note frequency and chords being played. The risks regarding our dataset are minimal because of the abundance of data points and the fact that it is open source. Extensive data will lend well to getting low true error.

Background

Before beginning the project, it was important to get a strong understanding of our problem and the solutions and methods that may already exist to tackle it. Dr. George Tzanetakis was instrumental in the approach early on. He directed us towards the implementation of a similar project "Onsets And Frames: Dual-Objective Piano Transcription" [2]. While the direct methods of their project were different, we were able to find key approaches that we were able to apply to our project. Most importantly, this project helped develop our ability to use MIDI files as an intermediate step from classifying the notes from the audio files to the conversion to sheet music. The information provided by Dr. Tzanetakis was used throughout the project as we executed our experiments.

When initially reviewing the dataset, we quickly realized that some additional work was needed to process it into a state that was usable given our knowledge of classification. The first hurdle was the fact that the data was unlabeled. Given our choice of models for this classification problem, it was required that we label our dataset. This was done by iterating through each frame of the dataset and using the midi values to label each note. Now, with the labeled data it was possible to train our models properly for further evaluation. Given our limited computational power, we realized that the size of the dataset and methods of choice was an issue. In order to complete this project we would need to truncate the dataset. As to avoid skewing our results, we selected random subsets from the overall dataset each time we ran our models.

For our validations after training our models, we used two different tools for visualizing our final results against what we were aiming to classify. To generate our validation sheet music, we used AnthemScore[4]. This allowed us to compare our sheet music output with that of a program that is known to work properly. Further comparisons between our results will be done through Librosa[6], which is a music information retrieval library.

Approach and Experiment Results

In order to evaluate the accuracy of our models we conducted three experiments. The first two used the raw data without scaling while the third used scaling on the data. Each experiment utilized two models, a Gaussian SVM and a Neural Network. Lastly, it is important to note that each experiment was repeated three times for each model and for each iteration the number of songs used was increased. Each of the following experiments have a corresponding table containing their results within the Appendix.

Through the process of evaluating our models we realized the dataset is too large and we needed to scale the features to make predictions quicker. We implemented two methods of scaling. The first used sklearn's MinMaxScaler which ensured that each frame's minimum value and maximum value were changed to -1 and 1 respectively. Method two employed a global scaling which divided the entire dataset by the maximum absolute value of the dataset. Lastly, we attempted T-SNE scaling as a method of dimension reduction.

Experiment 1: Raw data with default parameters

To have a benchmark of our models' performance we decided to first use the raw data with default parameter values for the two models. The parameter values used in this section can be found on the documentation page provided by sklearn for the Gaussian SVM and the Neural Network.

The default parameter values found and used from the documentation for this model were as follows:

1. **Regularization parameter (C):** 1
2. **Gamma:** scale
3. **Kernel:** rbf
4. **Iterations:** -1

Additionally the parameter values used for the Neural Network were:

1. **Hidden Layer:** 100
2. **Solver:** Adam
3. **Iterations:** 200
4. **Activation:** relu

Experiment 2: Raw data with optimized parameters

Now that the benchmark has been set we needed to find optimal values for our parameters that would enhance the performance of our models. Due to our data being very dense we could not perform a trial and error approach to find the optimal values. Hence, we relied on data collected from past experiences with Gaussian SVM and Neural Networks.

The optimized parameter values used for this model were as follows:

1. **Regularization parameter (C):** 150
2. **Gamma:** 0.01
3. **Kernel:** rbf
4. **Iterations:** 100000

Additionally the parameter values used for the Neural Network were:

1. **Hidden Layer:** (77,120)
2. **Solver:** Adam
3. **Iterations:** 10000
4. **Activation:** tanh

Experiment 3: Scaled data with optimized parameters

The final experiment consisted of using two different forms of data processing and feature reduction to determine which worked the best on the optimal models. The first method determined a local maximum within each of the training files and scale the data by that factor. The second found a global maximum within all training files and scaled the data with that. Lastly, we used T-SNE feature reduction to see if this would help improve our results.

Output Evaluation

To evaluate the output of the program, this section will look at the resulting sheet music as well as the song that results from playing the sheet music. Both these will be compared against the original song in two factors: note correctness, whether the predicted piece follows a similar melody; and note duration, whether the predicted piece's note has the same timing as the original.

Firstly, below examines the first four lines of sheet music of the prediction against the first three lines of the original.

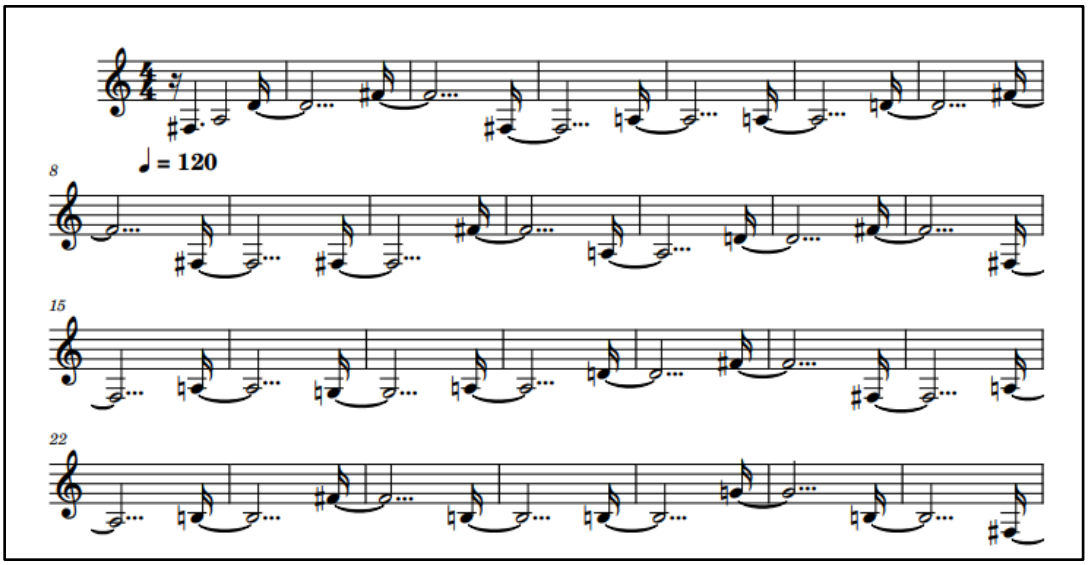

Figure 1: Predicted sheet music

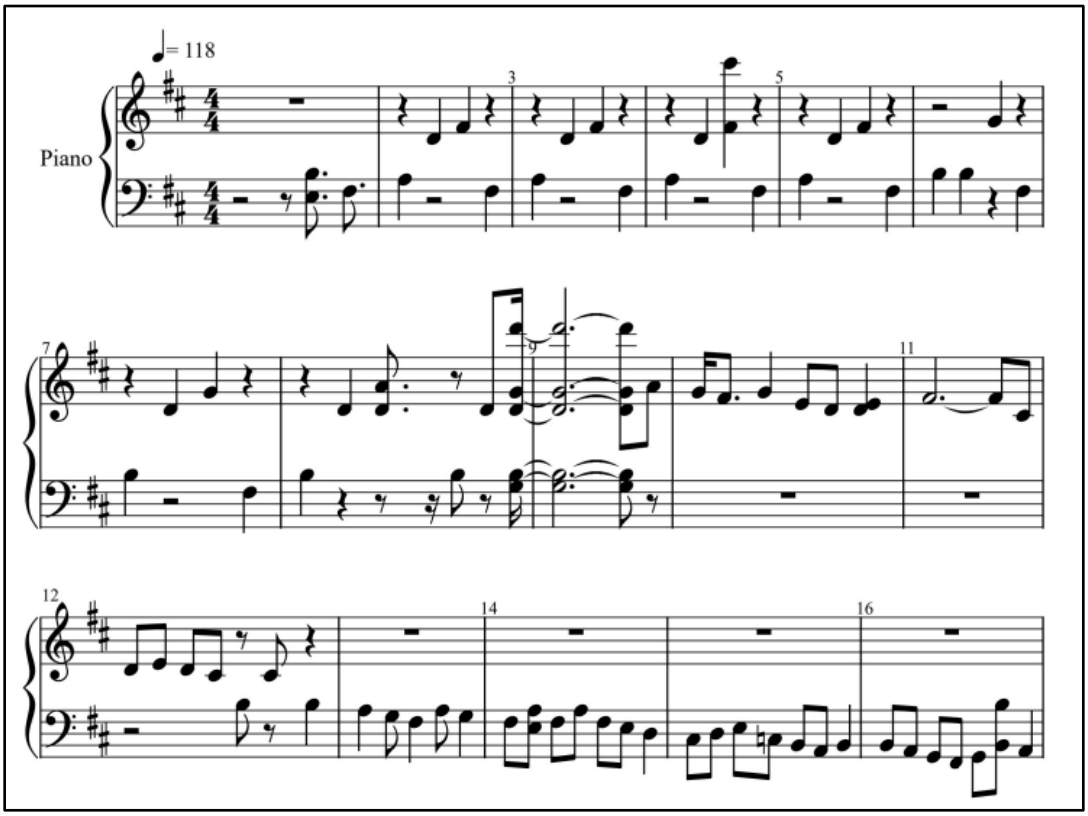

Figure 2: Original Sheet Music generated by AudioScore[4]

For note correctness, the predicted sheet music approximates the original with some success. The predicted music contains a similar note structure of increasing quads. This is expected given the error rates achieved. Discrepancies are encountered with note duration however. Our program is not accurately predicting the correct duration for the

notes. This is seen with how the original sheet music contains much more notes than our prediction. However, the model is not completely at fault, as the notes returned from the model may not be correctly inferred. This task is concerned more with music retrieval information (MIR) rather than machine learning.

To compare the predicted sheet music when played against the original song, time waveforms and cross similarity matrices will be used. Time waveforms describe the amplitude of an audio track and are commonly used in music editing applications. Below is the original and predicted songs' time waveforms:

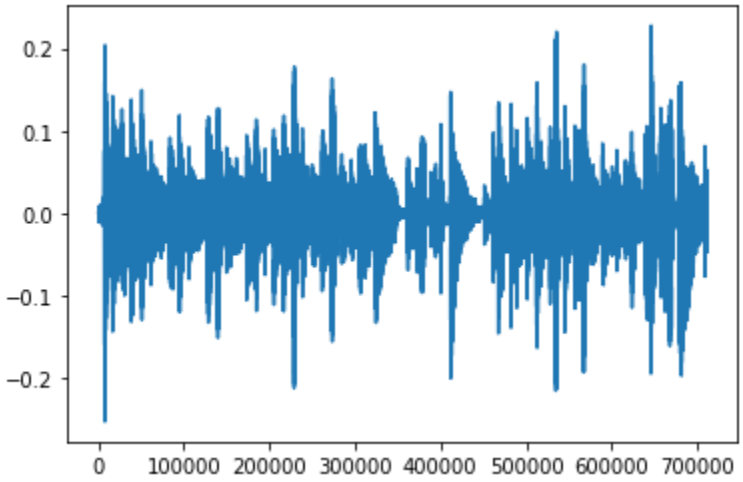

Figure 3: Time waveform of the test song "00-BN3-119-G_solo_mic.wav" from the dataset

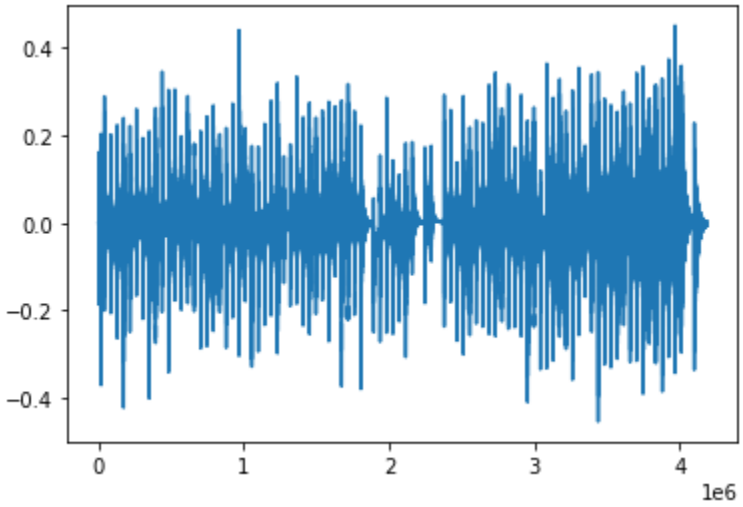

Figure 4: Time waveform of the played predicted sheet music.

Lastly, below shows both songs overlaid on one another.

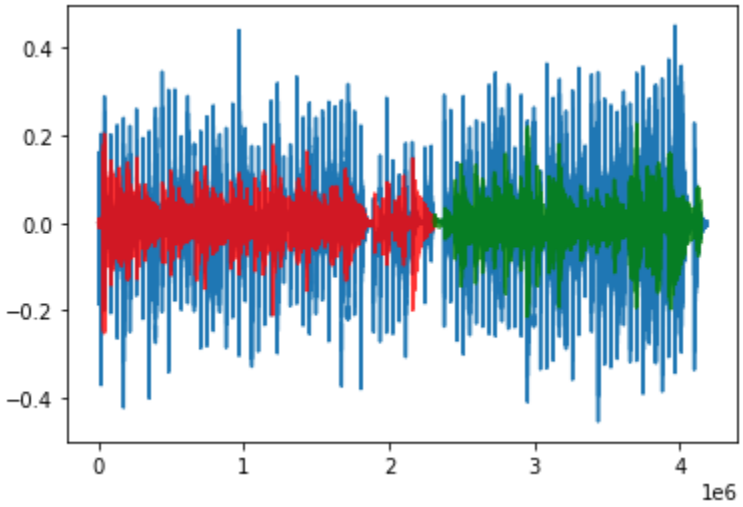

Figure 5: Time waveform overlay

Blue is the predicted audio, and red and green are the original. Due to duration limitations with synthesizing MIDI files, the original song's duration is scaled as follows:

- Red duration multiplied by 5.25
- Green: duration increased by 6.75

From this plot we can clearly see that while some of the timing had to be adjusted, the key features that are present in the original audio plot of the 00_BN3-119-G_solo_mic.wav file are still present in our plot from the sheet music. Some of the changes in shape can be attributed to the artificial instrument used to play back our sheet music which will not perfectly match the manually played guitar in the dataset's recordings. In summary, the time waveform gives good intuition that the predicted song has similarities to the original.

Lastly, to further analyze the similarities between the original and predicted songs, cross similarity matrices featuring the two will be examined, using the music information retrieval library Librosa[6]. Cross similarity matrices are a two dimensional graph where each axis represents a song. The colouring of the graph and diagonal line indicate the similarity between the two songs.

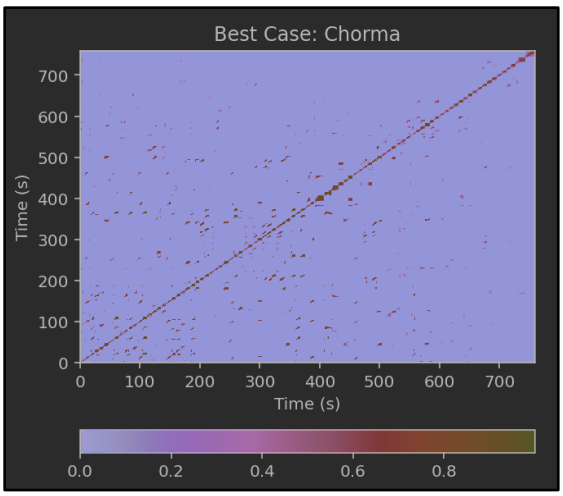

Figure 6: A best case cross similarity matrix

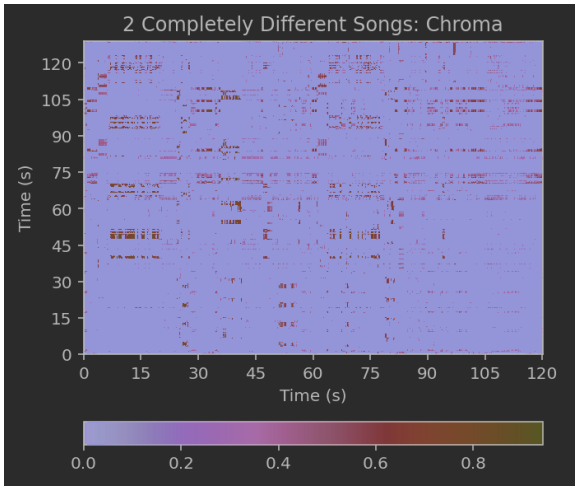

Figure 7: A worst case cross similarity matrix

For Figure 6, notice the perfectly straight diagonal, which indicates each part of the song matches with the other.

For Figure 7, notice no diagonal pattern is present and furthermore, no consistent pattern appears anywhere. This indicates no parts of the song are alike.

Below is the cross-similarity matrix, resulting from our program:

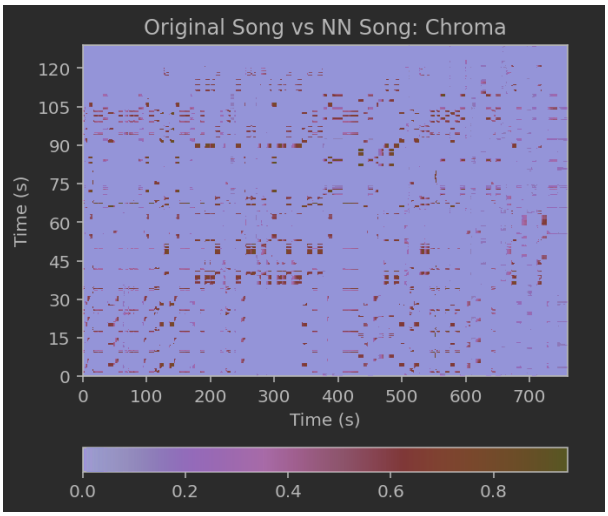

Figure 8: Predicted songs vs original cross similarity matrix

In this cross similarity matrix, we can see some relation between the audio tracks in the form of the diagonal line. Therefore, indicating the songs have similarities. However, this graph measures both note pitch and note duration, therefore, results may not be perfectly telling of similarity.

Overall, the model produces some success in accurately producing sheet music, with note correctness being the most successful area. This is encouraging as this is the area of the project most related to machine learning.

Conclusion and Future Work

Through various experiments we were able to design a model that achieved roughly 30% test error and 20% training error with 5/10/15 songs respectively. This model was a neural network utilizing the aforementioned parameters in the Experiments section. This aligned with our expectations for the optimal model choice but surpassed our error predictions, performing better than we initially hoped given our limited resources.

Although our results achieved a low true error, there are a few key limitations and problems that we have identified upon further retrospection that would require further re-working to effectively tackle this problem. These limitations are as follows:

- Given our limited understanding of music theory as a whole, there are components of this classification problem that we have excluded from our solution as we did not know they existed. Factors such as the key of the song, the bpm, and note duration have all been excluded from the scope of our experiments, and this can be seen in the final product where these theoretical components are missing from the outputed sheet music.
- Our limited computational power led us to only use a maximum of 15 songs for the models. We decided any training that took multiple hours was too long and would halt its progress due to the other experiments we wanted to explore.
- Our limited knowledge of classification techniques led us to use things we have learned in course. Although we did an initial survey of the current solutions offered for similar problems, it was difficult to find details on exactly which methods they were using so we had to make do with the information we had at the time. It is likely solutions like AnthemScore[4] use more nuanced techniques such as deep learning for generating their predictions.

If we where to continue this project in the future we would have the following goals moving forward:

- We would first want to expand our knowledge of music theory so we can incorporate things like bpm, key, and note duration into our predictions. This would vastly increase the similarity between our output sheet music and the expected sheet music.
- We would begin to run our optimized model on larger and larger sets of the data. This should hopefully improve our results further.
- We would incorporate newer and more domain specific classification techniques to this problem hoping to see if we can identify better methods than what we currently have.
- We would expand our use of the dataset to start classifying using the chord progression pieces included in it.

References

[1] - J. P. Bello, R. Bittner, J. Pauwels, Q. Xi, and X. Ye, "Guitarset: A Dataset for Guitar Transcription", in 19th International Society for Music Information Retrieval Conference, Paris, France, Sept. 2018. [Online] Available: https://guitarset.weebly.com/

[2] - D. Eck, E. Elsen, J. Engel, C. Hawthorne, S. Oore, C. Raffel, A. Roberts, I. Simon, J. Song, “Onsets And Frames: Dual-Objective Piano Transcription”, in 19th International Society for Music Information Retrieval Conference, Paris, France. [Online] Available: https://arxiv.org/pdf/1710.11153.pdf

[3] - L. Fritts, “Musical Instrument Samples”. [Online] Available: https://theremin.music.uiowa.edu/MISpiano.html

[4] - “AnthemScore 4”, Lunaverus, 2022. [Online] Available: https://www.lunaverus.com/

[5]- D. Zemsky, “Reading sheet music for you,” *Sheet Music Scanner*. [Online]. Available: https://sheetmusicscanner.com/. [Accessed: 14-Feb-2022].

[6] Brian McFee, “librosa/librosa: 0.9.0”. Zenodo, Feb. 07, 2022. doi: 10.5281/zenodo.5996429

Appendix: Experiment Result Tables

The tables below show the training accuracy and error along with the test accuracy and error of both the models as the number of songs increase.

Experiment 1:

Model	**Gaussian SVM**		
Songs	5	10	15
Train Accuracy	Did not converge	Did not converge	Did not converge
Test Accuracy	Did not converge	Did not converge	Did not converge
Train Error	Did not converge	Did not converge	Did not converge
Test Error	Did not converge	Did not converge	Did not converge

Model	**Neural Network**		
Songs	5	10	15
Train Accuracy	24.01%	22.90%	28.22%
Test Accuracy	23.37%	22.22%	26.85%
Train Error	2135.03	2050.25	2387.43
Test Error	2150.67	2054.40	2389.28

Experiment 2:

Model	**Gaussian SVM**		
Songs	5	10	15
Train Accuracy	100%	100%	95%
Test Accuracy	85.43%	95.52%	56%
Train Error	0.0	0.0	238.37
Test Error	563.38	190.73	503.99

Model	**Neural Network**		
Songs	5	10	15
Train Accuracy	44.92%	37.21%	35.74%
Test Accuracy	30.23%	30.94%	33.21%
Train Error	1309.47	1773.12	1976.48
Test Error	1576.26	1963.52	2032.52

Experiment 3:

Local Scaling:

Model	**Gaussian SVM**		
Songs	5	10	15
Train Accuracy	100%	Did not converge	Did not converge
Test Accuracy	86%	Did not converge	Did not converge
Train Error	0.0	Did not converge	Did not converge
Test Error	543.31	Did not converge	Did not converge

Model	**Neural Network**		
Songs	5	10	15
Train Accuracy	48.13%	36.24%	35.85%
Test Accuracy	39.19%	30.67%	31.81%
Train Error	1170.41	1838.32	2082.82
Test Error	1331.93	1946.47	2166.88

Global Scaling:

Model	**Gaussian SVM**		
Songs	5	10	15
Train Accuracy	94.32%	79.30%	75.76%
Test Accuracy	56.68%	49.92%	52.78%
Train Error	245.21	611.64	705.04
Test Error	470.18	787.37	872.62

Model	**Neural Network**		
Songs	5	10	15
Train Accuracy	79.42%	81.59%	76.40%
Test Accuracy	75.65%	66.08%	69.63%
Train Error	643.07	609.14	726.90
Test Error	700.68	932.54	856.81

T-SNE Feature Reduction:

Model	**Gaussian SVM**		
Songs	5	10	15
Train Accuracy	80.56%	74.67%	69.03%
Test Accuracy	59.31%	63.91%	60.63%
Train Error	423.72	662.11	821.88
Test Error	801.85	880.21	1004.36

Model	**Neural Network**		
Songs	5	10	15
Train Accuracy	40.16%	37.13%	34.99%
Test Accuracy	36.93%	34.35%	31.86%
Train Error	1587.48	1414.53	1890.77
Test Error	1645.89	1776.78	1970.62