Generating MIDI songs using an RNN

Presented to

Professor Mark Stamp

Department of Computer Science

San Jose State University

In Fulfillment

of the Requirements of Class

Spring-2021: CS 271

By

Matthew Gerlits

Alex Wolski

May 2021

Background

Deep learning models automatically detect and train on features in a data set. A Recurrent Neural Network (RNN) is a deep learning model that implements a primitive form of memory. RNNs, and especially the Long Short-Term Memory (LSTM) architecture, excel at learning from sequential data where a data point can influence another data point further in the sequence. Because of this quality, RNNs are often used for natural language processing. The architecture is able to learn the syntax and grammar of a language and apply that knowledge to generate new sentences [1]. Similar to a language, music also follows a set of rules and patterns. The different genres of music are analogous to different languages. And the key signatures of a musical piece are analogous to grammar rules. The obvious question then arises: *Can an RNN be used to generate music?*

Project Overview

The objective of this project is to train an RNN that generates a specific genre of music and to quantify its success. The two genres we chose were classical and jazz. We first constructed a corpus of MIDI song files for each genre. For each corpus, a separate RNN was trained to predict the next note in a sequence of notes. The trained RNN was used to repeatedly predict the next note of a random sequence and build a unique song. Lastly, we trained a hybrid multiclass classifier of traditional machine learning classifiers to score the songs generated by the RNN.

Music Corpus

We chose to train our models on MIDI files. MIDI is a file format that represents a song as a series of pitches and timings. The simplicity of a MIDI file makes it far easier to train on than audio recordings. However, MIDI and other similar file formats have largely fallen out of popularity. This made it difficult to find a labeled dataset of MIDI files that would fit our needs. Instead, we acquired MIDI files from various sources and created our own corpus.

Several different web-scraper programs were used to download over 12,000 MIDI song files from across 12 websites. All duplicate files were removed, but variations of the same song were not. We then used the music21 musicology library to run preprocessing on the files and simplify them further. We limited our corpus to only piano parts and threw out any other instruments. The treble clef and bass clef were separated in order to break apart large chords. And we exported the note pitches in each song to a text file. The exported files did not contain note duration, so all timing information was discarded. After some initial testing, we found that songs from certain sources were giving us poor results. So we manually analyzed and curated the corpus. The resulting corpus contained 8,564 classical samples and 8,465 jazz samples.

Feature Extraction

Our traditional machine learning classifier needed to be trained on features extracted from the dataset. To extract these features, we agian used the music21 musicology library [2]. The library implements 73 features of the 153 outlined in the jSymbolic system [3]. Because of the heavy simplification we performed, only 30 of the features were applicable to our dataset. These features capture statistical data on the note pitches and chords present in a song. A table of the features used can be seen below.

Table 1
List of Features

AmountOfArpeggiation	MostCommonMelodicInterval
AverageMelodicInterval	MostCommonMelodicIntervalPrevalence
ChromaticMotion	MostCommonPitchClassPrevalence
DirectionOfMotion	MostCommonPitchPrevalence
DistanceBetweenMostCommonMelodicIntervals	NumberOfCommonMelodicIntervals
DurationOfMelodicArcs	NumberOfCommonPitches
ImportanceOfBassRegister	PitchClassVariety
ImportanceOfHighRegister	PitchVariety
ImportanceOfMiddleRegister	PrimaryRegister
IntervalBetweenStrongestPitchClasses	Range
IntervalBetweenStrongestPitches	RelativeStrengthOfTopPitchClasses
MelodicFifths	RelativeStrengthOfTopPitches
MelodicOctaves	RepeatedNotes
MelodicThirds	SizeOfMelodicArcs
MelodicTritones	StepwiseMotion

Genre Classification

We trained five different classification models on the extracted features of the music corpus. The four most performant models were combined using a hybrid multiclass classifier method to boost their performance. Used the scikit-learn python library implementations for all the models.

A. PCA

Principal Component Analysis (PCA) is a feature reduction method that determines the directions of most variance in the feature space. We trained two PCA models: one on the classical corpus and the other on the jazz corpus. We performed some testing on the number of principal components and found that 22 components was optimal for this problem. This produced a weight vector for each genre. To classify a sample, we compared the euclidean distance between its weight vector and the weight vector of each training set. The sample was classified as the genre associated with the closest training weight vector.

The PCA model achieved an accuracy of 74.2% and an AUC of 0.83. The training process was fast. But the scoring process involved intense computations and was too slow to be useful. Although the model was relatively successful, we decided to omit it from the hybrid multiclass classifier.

Figure 1
PCA: Number of Principal Components

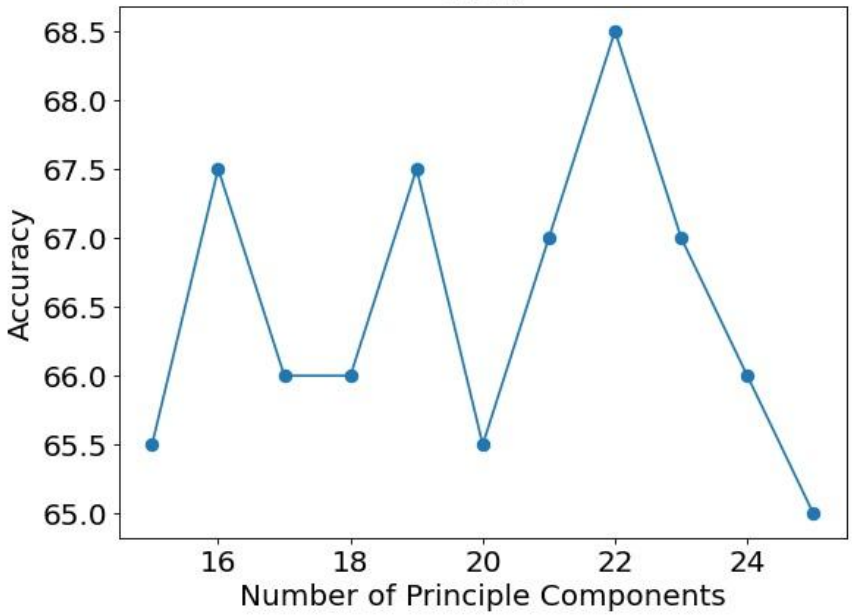
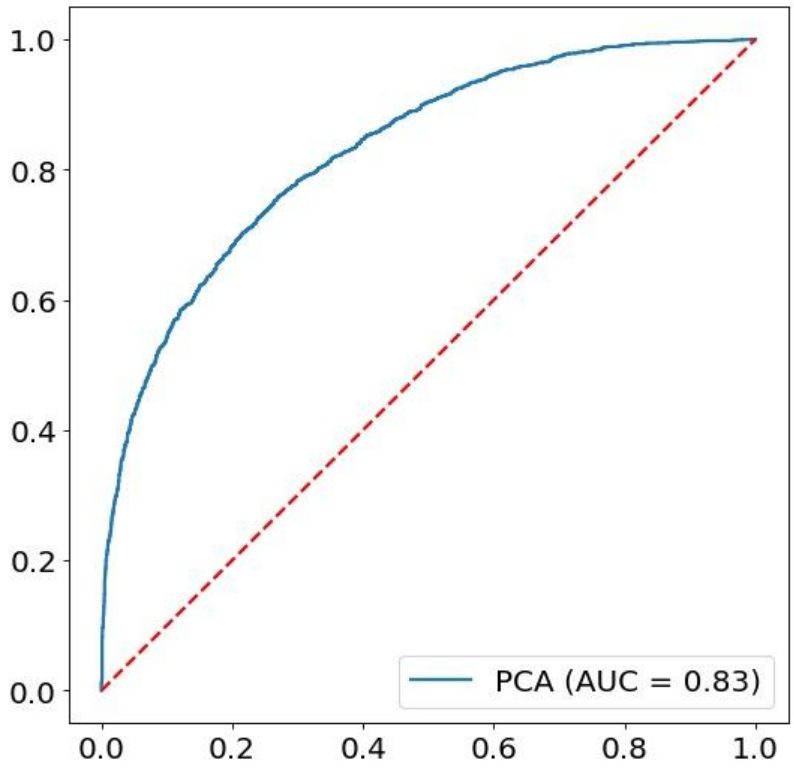


Figure 2
PCA: ROC Curve



B. SVM

The Support Vector Machine (SVM) technique uses a separating hyperplane to classify samples. Non-linear data can be accurately modeled through the use of a kernel that transforms the data. We experimented with three different kernels and tuned the hyperparameters to find the model that provided the best accuracy.

We found that the Radial Bias Function (RBF) kernel performed the best. The polynomial kernel was a close second and the Linear kernel performed significantly worse. The best model achieved 78.9% accuracy with an AUC of 0.87.

Figure 3
Linear Kernel: Margin Softness Tuning

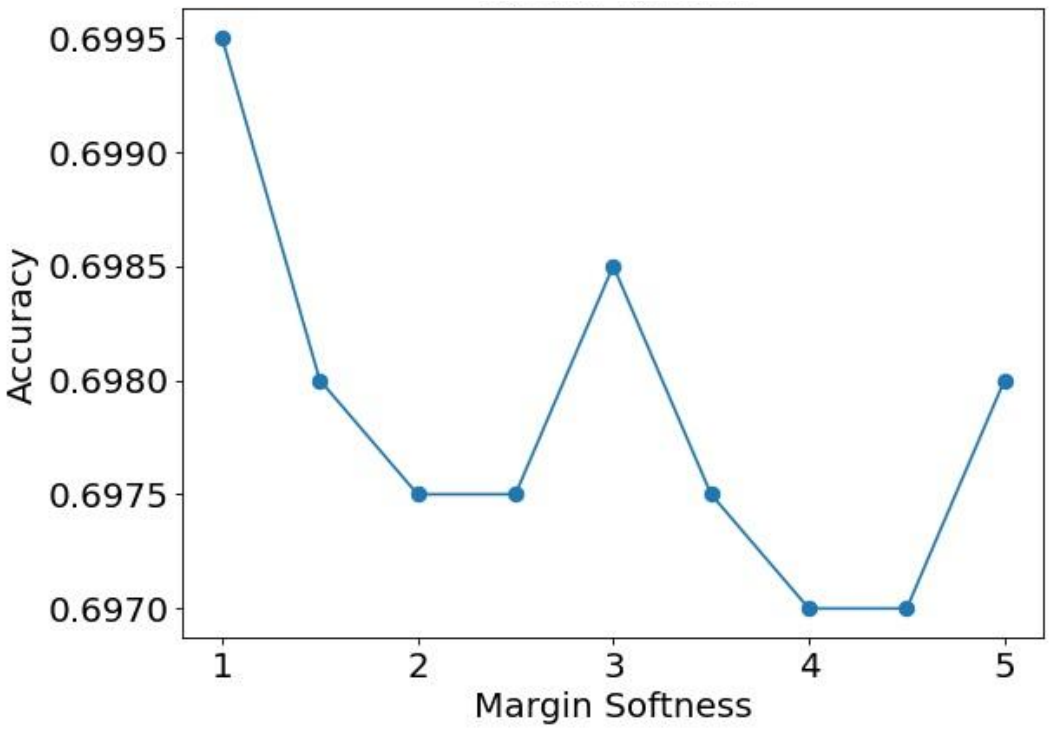


Figure 4
Polynomial Kernel: Parameter Tuning

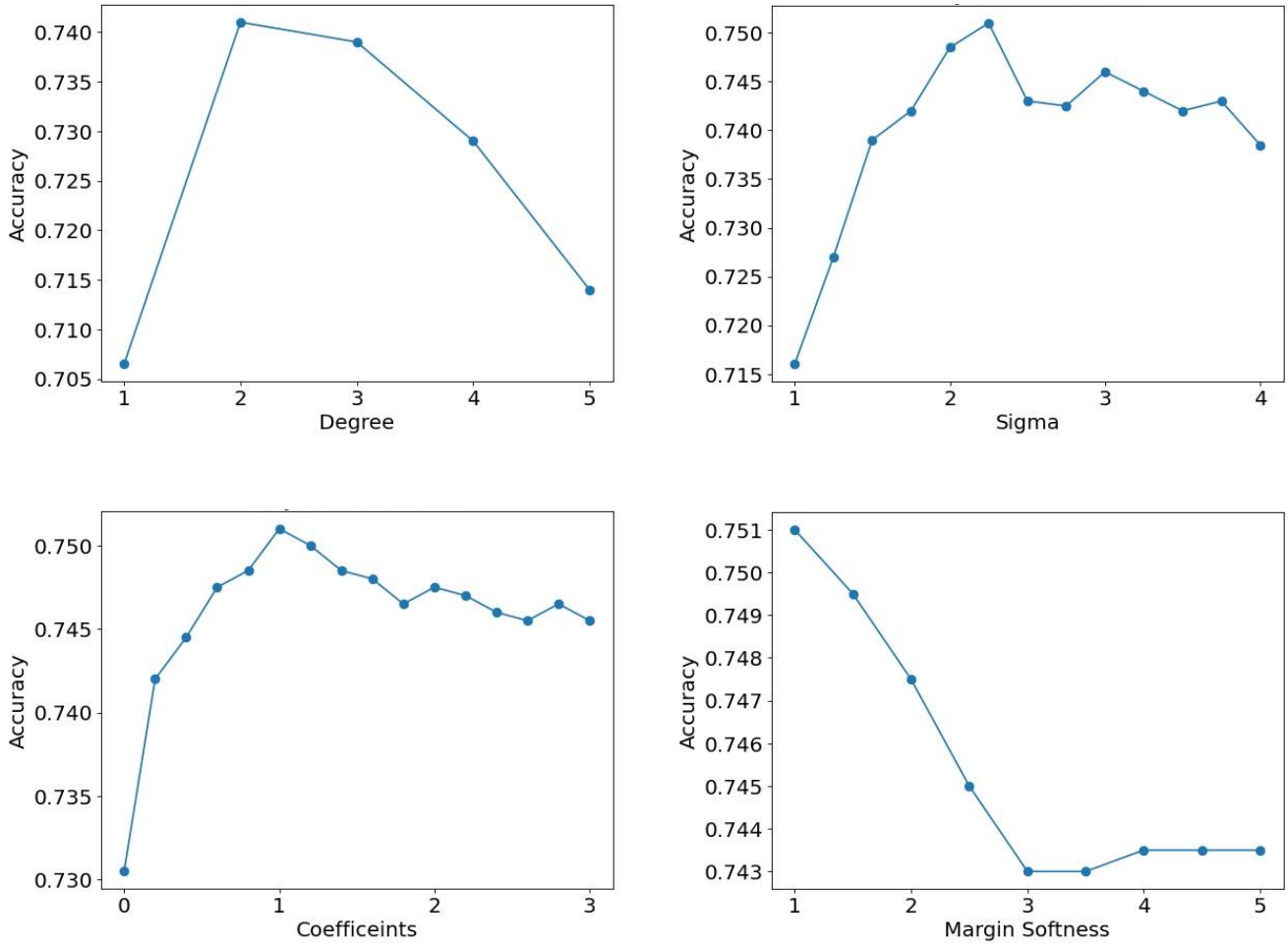


Figure 5
RBF Kernel: Parameter Tuning

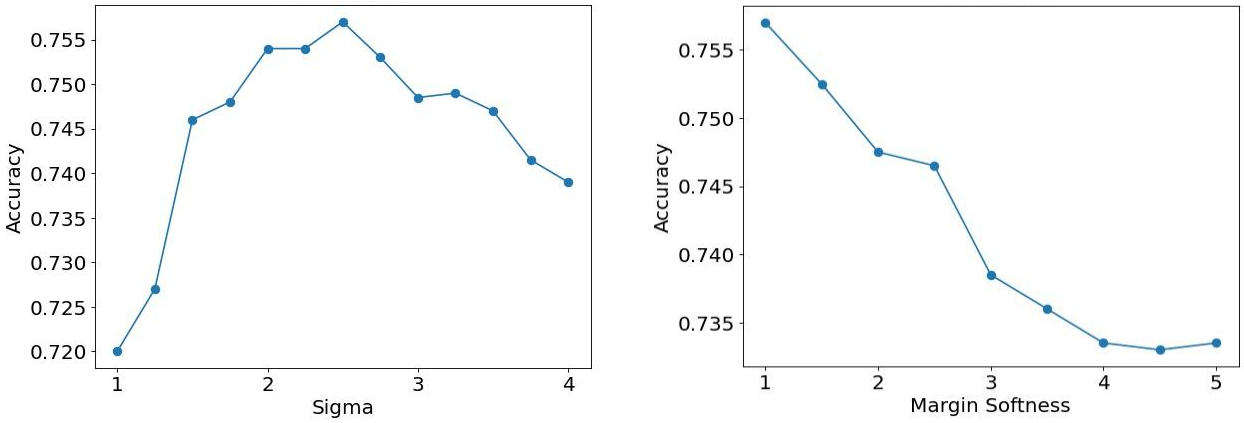


Figure 6
SVM: Kernel Performance

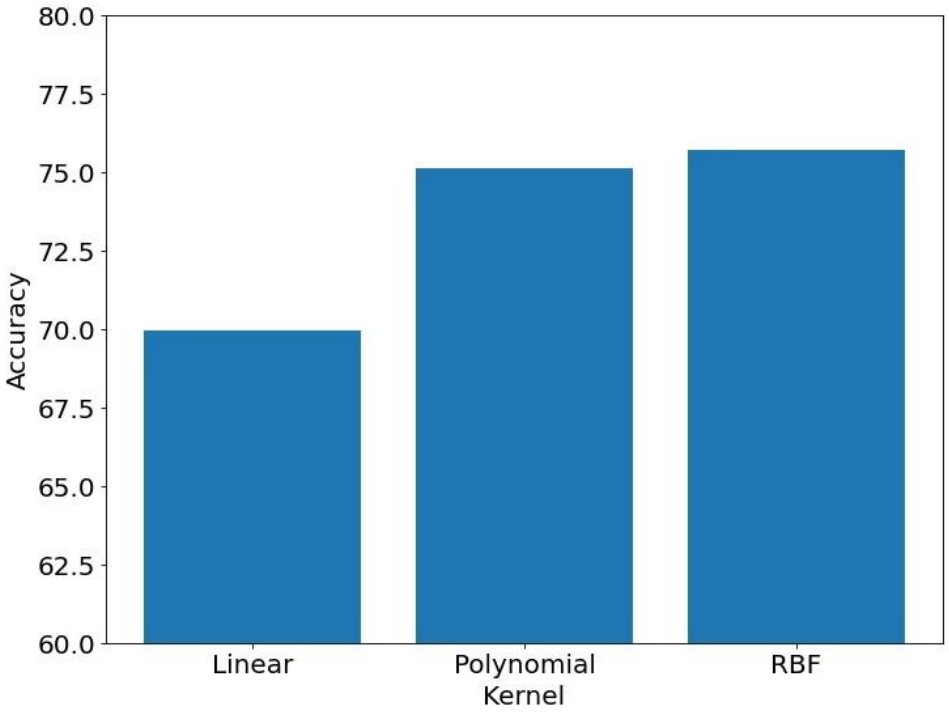
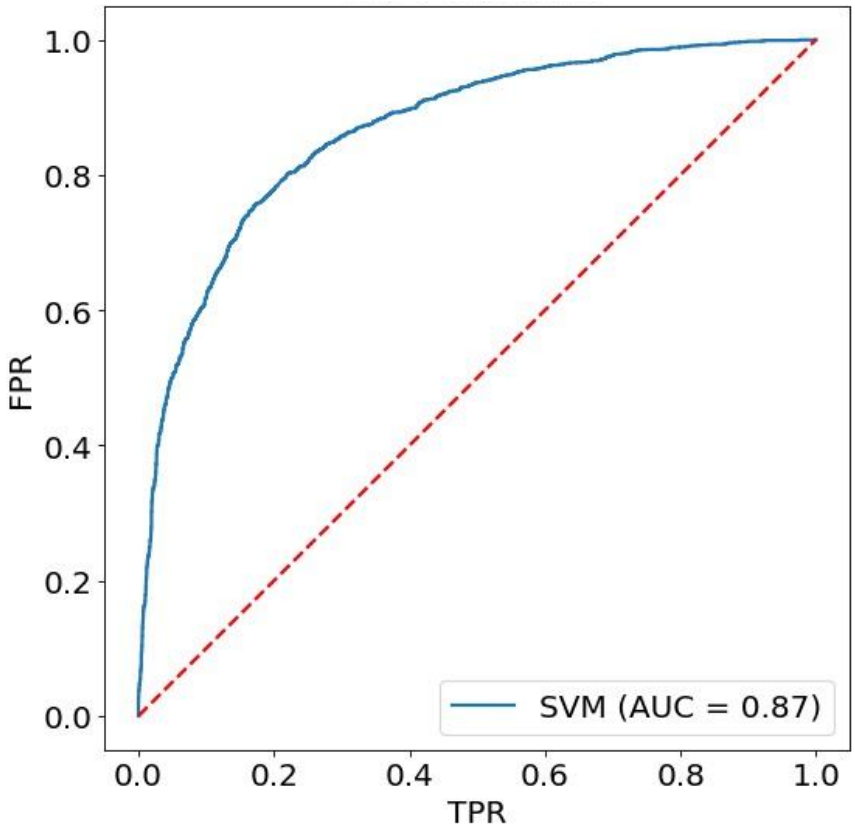


Figure 7
SVM: ROC Curve



C. KNN

The K-Nearest Neighbors algorithm determines the classification of a sample by observing the nearby samples. Despite being a simplistic algorithm, it performed relatively well for our application. The tree algorithm used didn't have much of an impact on performance, so we arbitrarily chose the kd-Tree algorithm.The best model achieved an accuracy of 76.3% with an AUC of 0.84.

Figure 8
KNN: Tree Algorithm

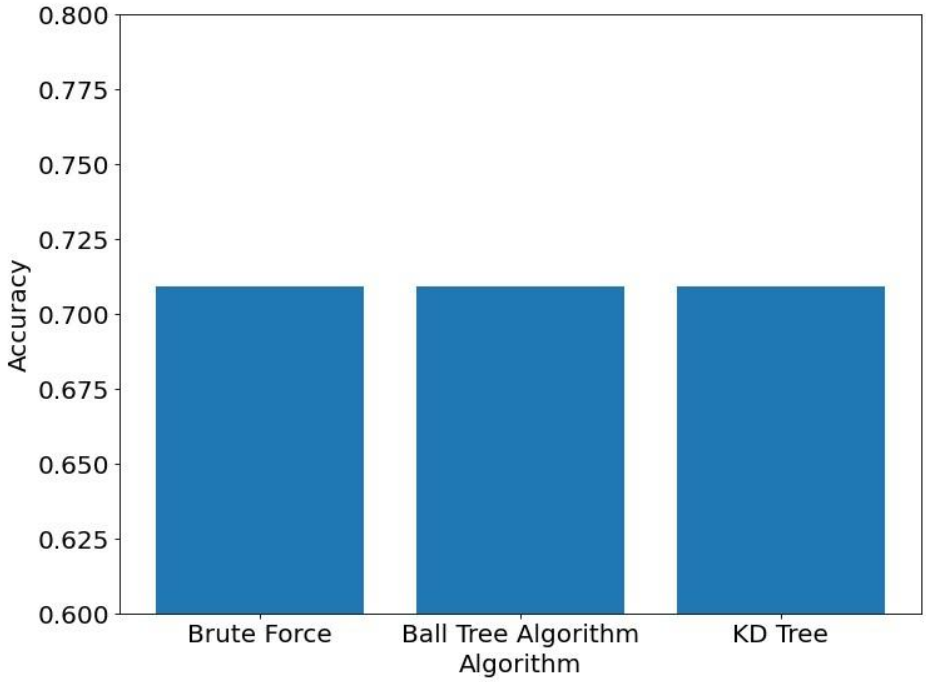


Figure 9
KNN: Parameter Testing

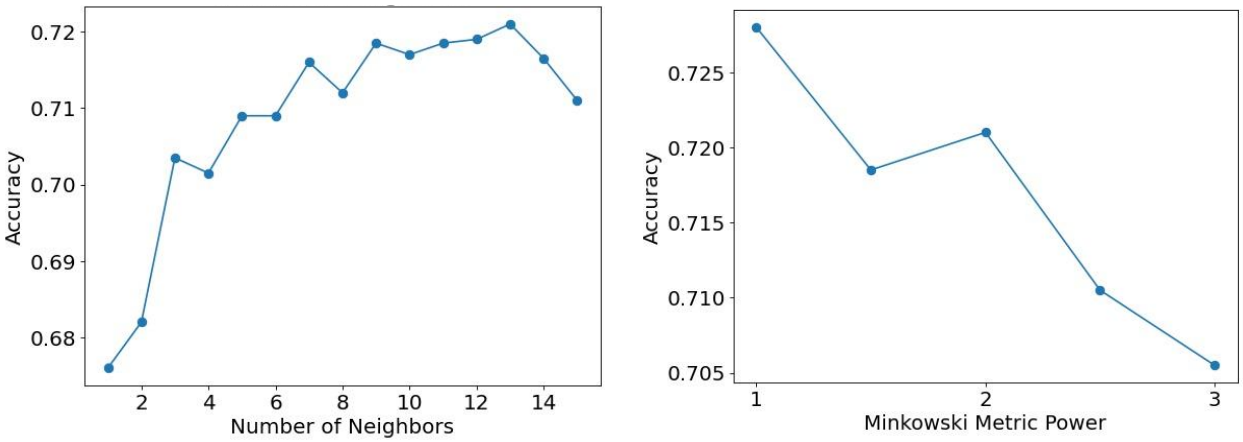
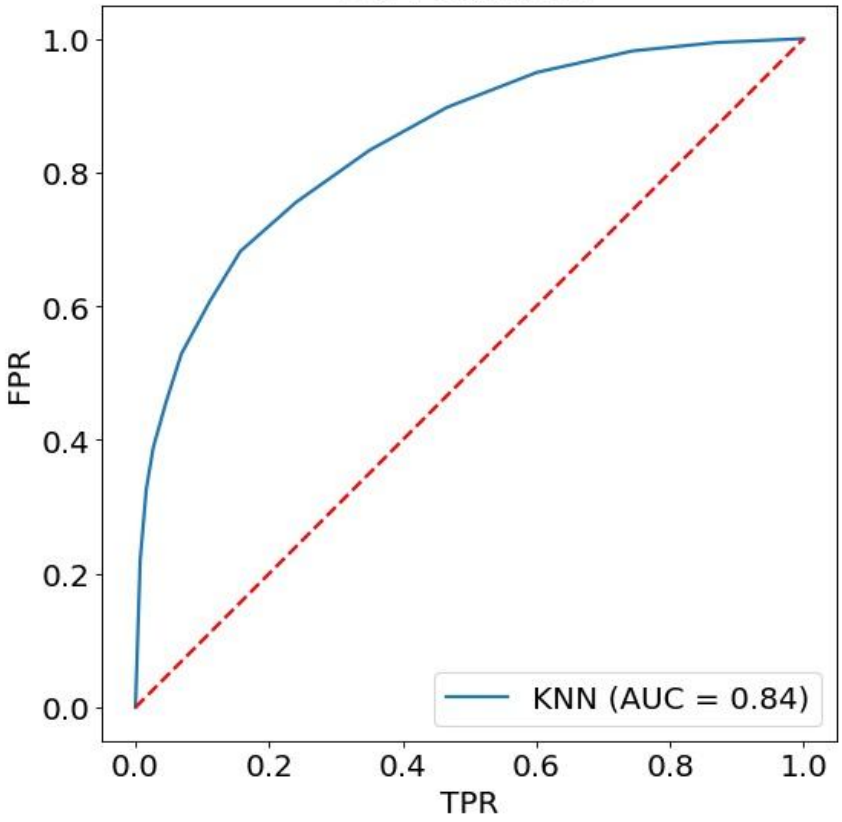


Figure 10
KNN: ROC Curve



D. LDA

Linear Discriminant Analysis (LDA) is a feature reduction technique that projects data samples onto a hyperplane. The implementation we used didn't provide many tunable parameters, so we only experimented with the solver algorithm. The SVD and LSQR algorithms performed nearly identically on our dataset. It achieved an accuracy of 74.2% with an AUC of 0.83.

Figure 11
LDA: Solver Algorithm

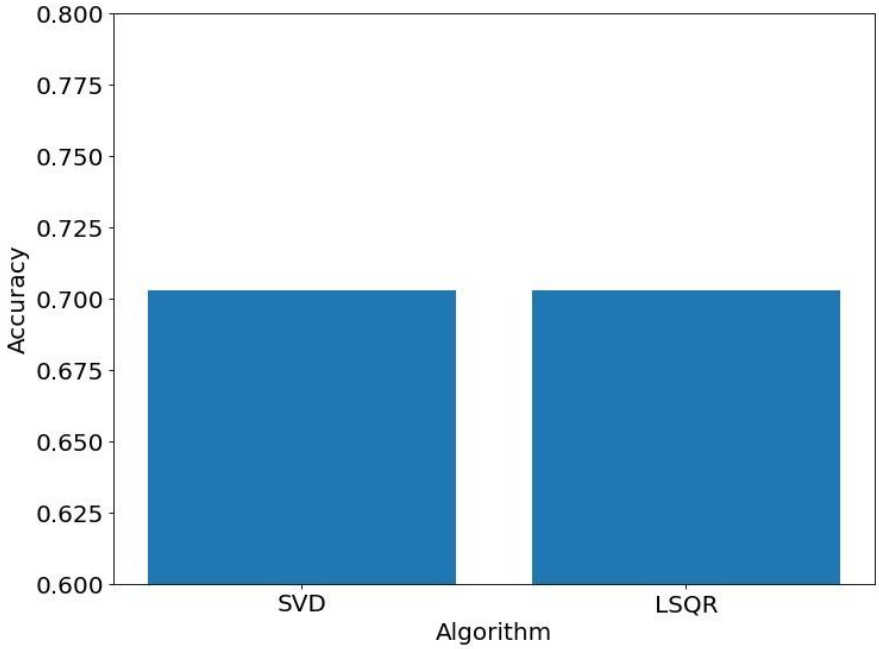
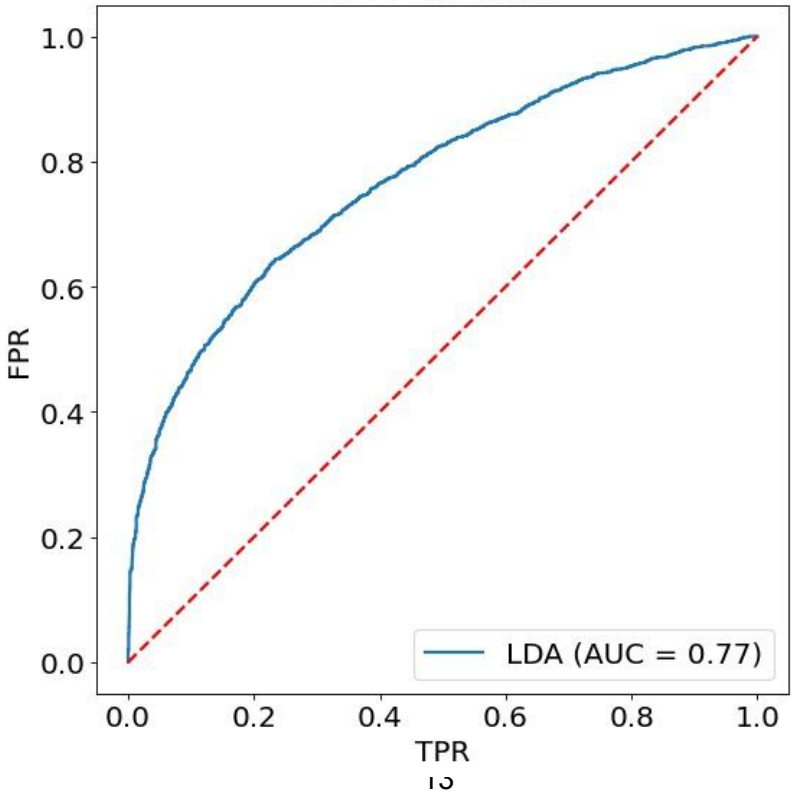


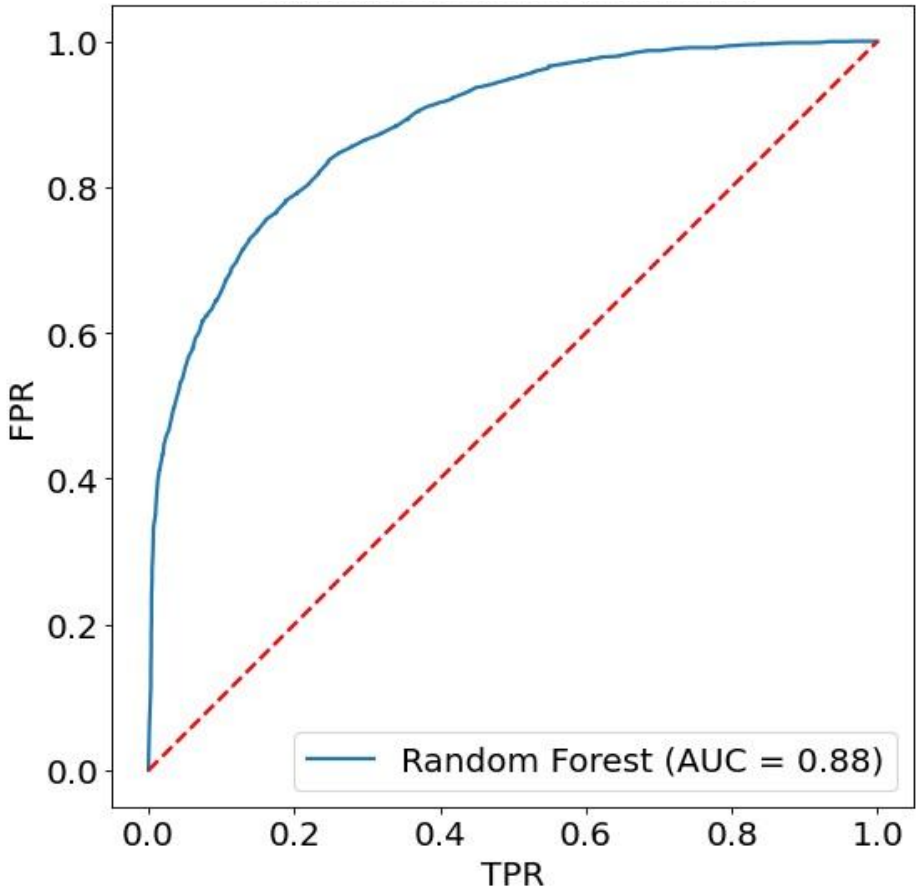
Figure 12
LDA: ROC Curve



E. Random Forest

A random forest is an ensemble of decision tree estimators. We had difficulty testing the parameters because each model was randomized and would perform drastically differently. Using the default setting of 10 estimators, the model reached an average accuracy of 79.2% and an AUC of 0.88. This is the best performing model out of all the base learners we trained.

Figure 13
Random Forest: ROC Curve



F. Stacking

Because he had four accurate and efficient classifiers, we decided to combine them using a hybrid multiclass classifier technique. A boosting method could also be used, but they are better applied to large numbers of weak classifiers. We used the Stack Generalization (Stacking) technique, which feeds the output of a base-learner model as the input to another model in order to create a more stable model [4]. We chose the SVM, KNN, LDA, and random forest models as the base-learners in the hybrid model. Because the stacking model took a significant amount of time, we did not perform additional parameter testing on the base classifiers.

The stacking model had the best accuracy and AUC at 79.9% and 0.89 respectively. However, this is only a marginal improvement over the best base classifier. The long training time of the model is hard to justify considering the minimal performance increase. Had we experimented with different parameters and architectures, we may have achieved better results.

Figure 14
Stacking: ROC Curve

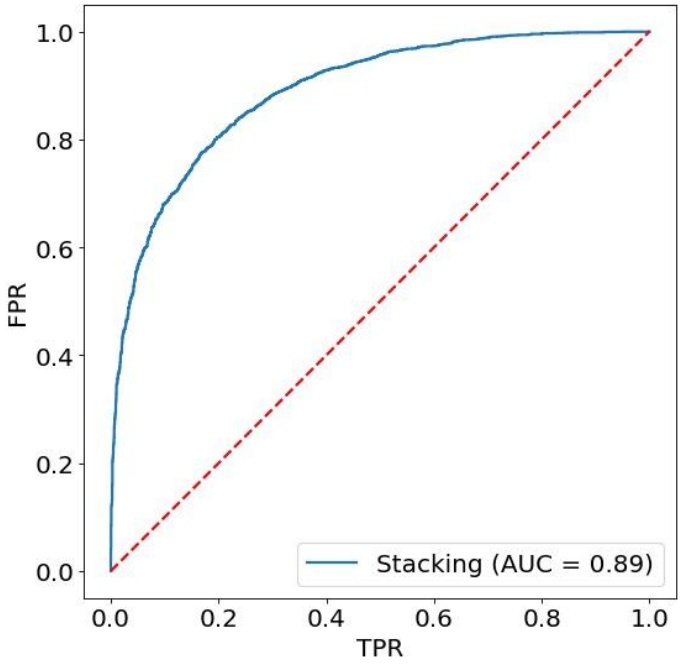
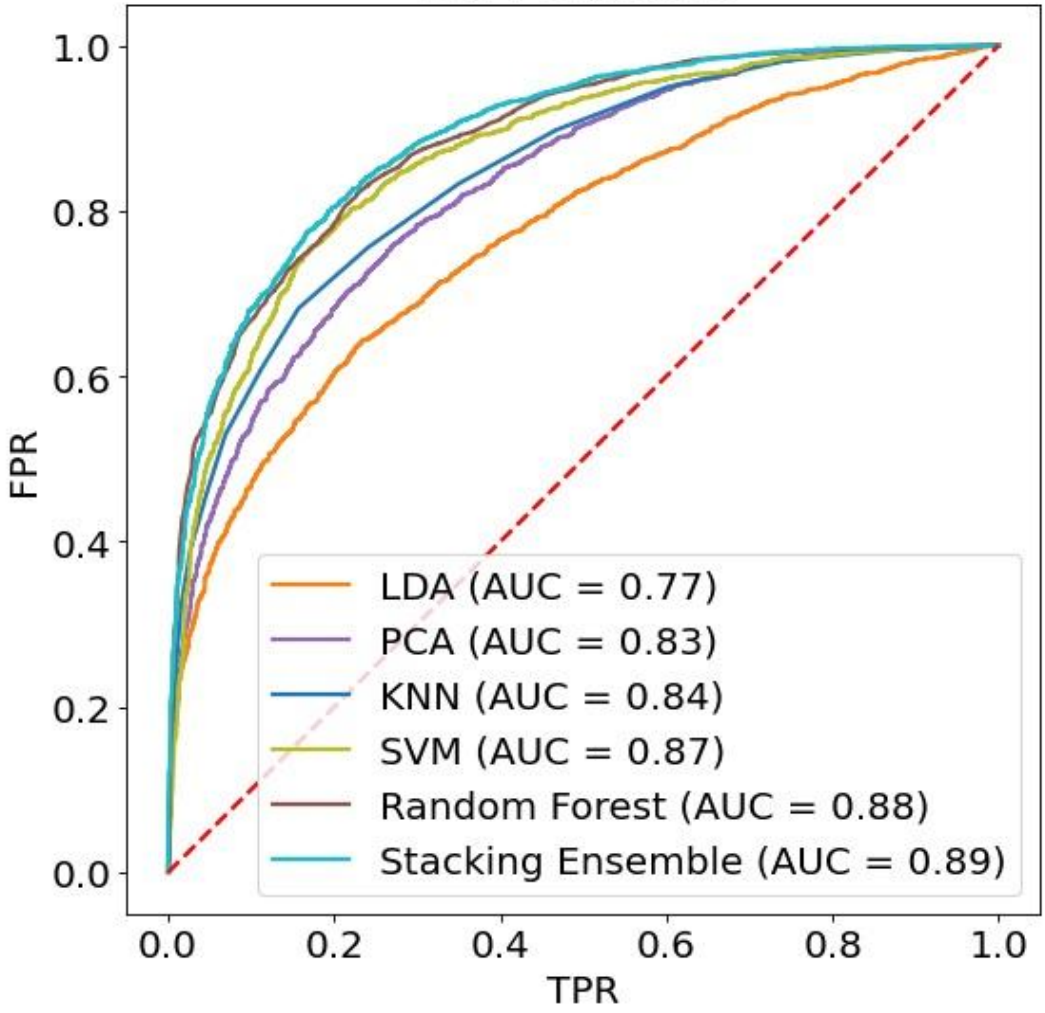


Figure 15
All ROC Curves



Classifier Performance

Our results show that traditional machine learning techniques can be used to classify the genre of a MIDI song. With more training samples and base learners in the hybrid multiclass classifier, the performance can be increased. However, the performance isn't as good as we had hoped for. This is most likely caused by an oversimplification of the training data. With less preprocessing and more extracted features, we expect the classifiers to approach 100% accuracy.

RNN-LSTM Design

Understanding music requires an understanding of the piece as a whole. Each movement in the piece has purpose and each chord is a part of a progression. Certain chord progressions are known to compliment each other. And knowing the proper sequence of chords is essential to making appealing music. RNN-LSTM models have a longer memory than most, allowing it to develop a greater understanding of how a song evolves.

We utilized a model proposed by Skuli, which uses 3 LSTM layers with supporting dropout and dense layers. This implementation uses the Categorical Cross Entropy loss function, which excels in multiclass classification problems[1].

The two primary parameters of the model are the sequence length and sample length. The sequence length is the number of notes fed into the RNN at a time. This determines how far back the neural network's memory reaches. And the sample length is the number of notes that the model was trained on. In order to find the optimal combination of sequence length and sample length, we performed a grid search. Sample lengths of 1000, 2000, and 3000 notes and sequence lengths of 10, 50, 100, 200, and 300 notes were used for a total of 15 combinations. We trained a separate model for each combination of parameters, training for 40 epochs and generating 20 songs. We then analyzed the loss of the model and classification score of the generated songs to determine performance. For the input to the model, we constructed a sequence of random notes and chords from the corpus. This ensured that the model would always produce a unique song while avoiding issues caused by notes that the model hasn't been trained on.

A sample length of 3000 notes is a small fraction of the over 4 million notes each of the corpuses. And only 20 generated songs per model is not ideal for analyzing performance. However, these were necessary sacrifices made in order to keep the training times reasonable.

Song Generation Results

A. Qualitative Analysis

A qualitative assessment of the generated songs resulted in mixed results. Some songs appeared to have structure, proper chord progressions, and aspects of the target gene. Other songs, however, would get stuck repeating the same note. This is a common issue with text generation models and occurs when the most likely continuation of a sequence is the sequence itself. We attempted to mitigate this problem by introducing a small amount of randomness when selecting the best note to continue a sequence. This addition made it less likely for the model to get stuck in a loop. However, the amount of randomness wasn't enough to completely remove the issue.

The classical music generator produced more consistent results than the jazz song generator. Fewer of the generated classical songs contained repeating notes. And the songs that did repeat were able to break out of the loop after a short while. The repetition issue was far more prevalent in the jazz song generator, with a majority of the songs getting stuck in a loop. This is most likely due to bad samples in the jazz corpus. Some jazz songs use the piano as a background instrument, where it repeats the same few chords over and over. The model may have picked on this repetition and copied it when generating new songs.

Lastly, a few of the generated songs showed signs of overfitting. In particular, one of the generated classical songs contained sequences of two or three notes that also appear in Beethoven's “Für Elise.” We found that the classical song corpus contained 28 different renditions of Für Elise. Although none of the renditions were identical, they were similar enough to create an unwanted bias in the model.

Figure 16
Verse Structure in a Generated Jazz Song

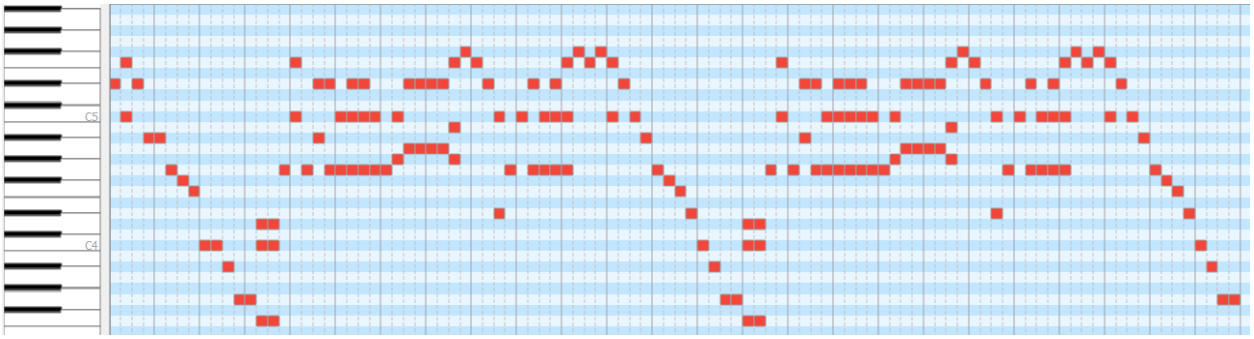
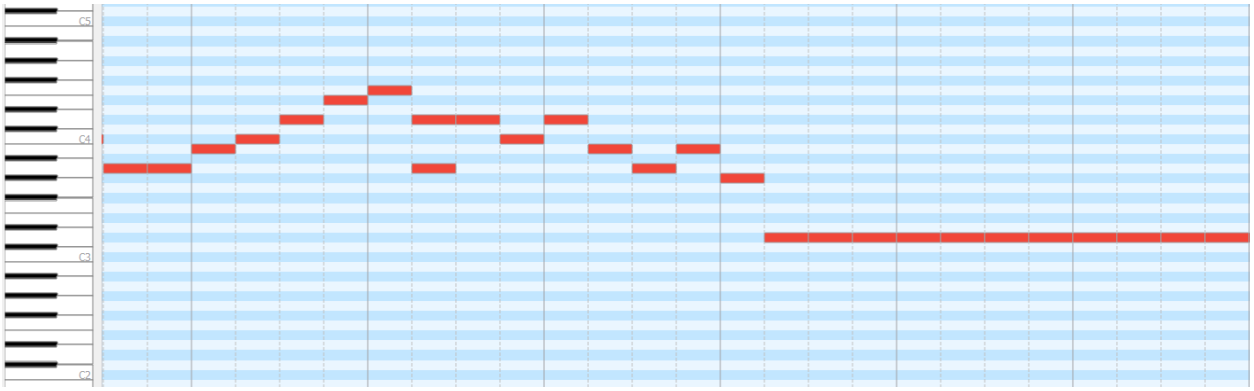


Figure 17
Repetitive Sample in the Jazz Corpus



A. Quantitative Analysis

In order to quantitatively measure the success of the generative models, we used the classifier described earlier in this paper. We defined the accuracy of a model as the percentage of the generated songs that were classified as the target genre. And we defined the score of each model as the average confidence of the classifications. A score of 0 would mean that the classifier is certain the song has the wrong style. And a score of 1 would mean that the classifier is certain the song has the style of the target genre.

The classifier was partially successful in evaluating the performance of the song generator models. The performance analysis of the classical song generator shows a clear correlation between the sample length parameter and the score and accuracy of the model. Contradictory to our expectations, the performance of the model increased as the number of training samples decreased. The best accuracy was 85%, achieved by a model with a sample length of 1000. And the worst accuracy was 20%, attributed to a model with a sequence length of 3000. We believe that the models with shorter sample lengths had better performance because they overfitted the data. With only a couple of songs to train on, the models would generate songs very similar to the training data. And since the only performance metric that the classifier analyzed is similarity to the training set, the overfitting models were deemed to have a high performance. The generator failed to produce unique outputs when trained on a small dataset. And the classifier failed to measure the uniqueness of a generated song.

Another unexpected result was that the performance of a model wasn't correlated to the sequence length it was trained on. We expected that models with longer memories would be able to produce higher-quality songs. But the evaluated performance of the models seemed to fluctuate randomly regardless of the sequence length. This could be a failure of the model to learn the structure of a song and apply it to a generated song. But it is also likely that the classifier was unable to detect the movements in a song. The features we extracted were mainly focused on the frequency of certain pitches, and few of them captured shifts in the music.

Lastly, we observed that the accuracy and average score of each model were similar. Computing the score of a song requires additional training and computations. So just calculating the accuracy would be sufficient.

Figure 18
Classical Song Generator Accuracies

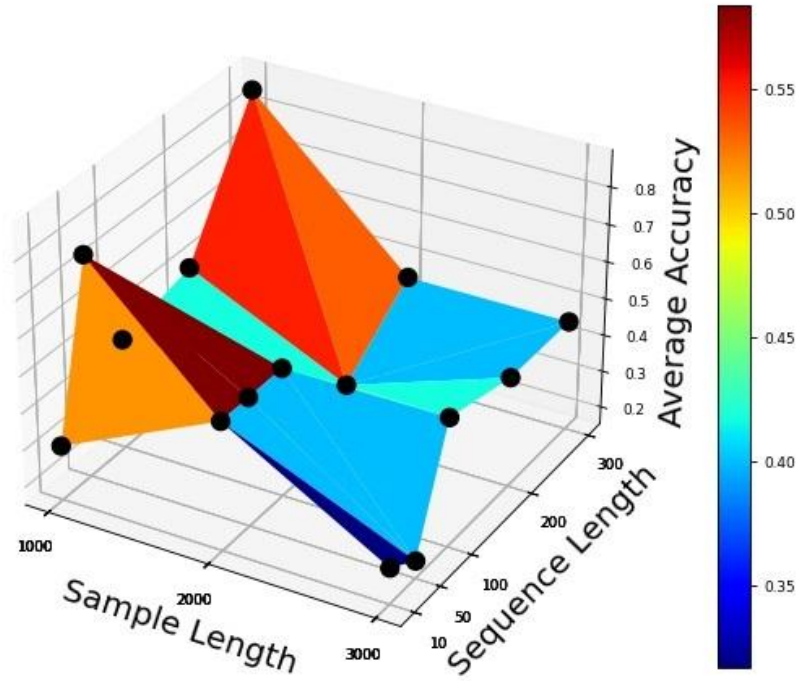
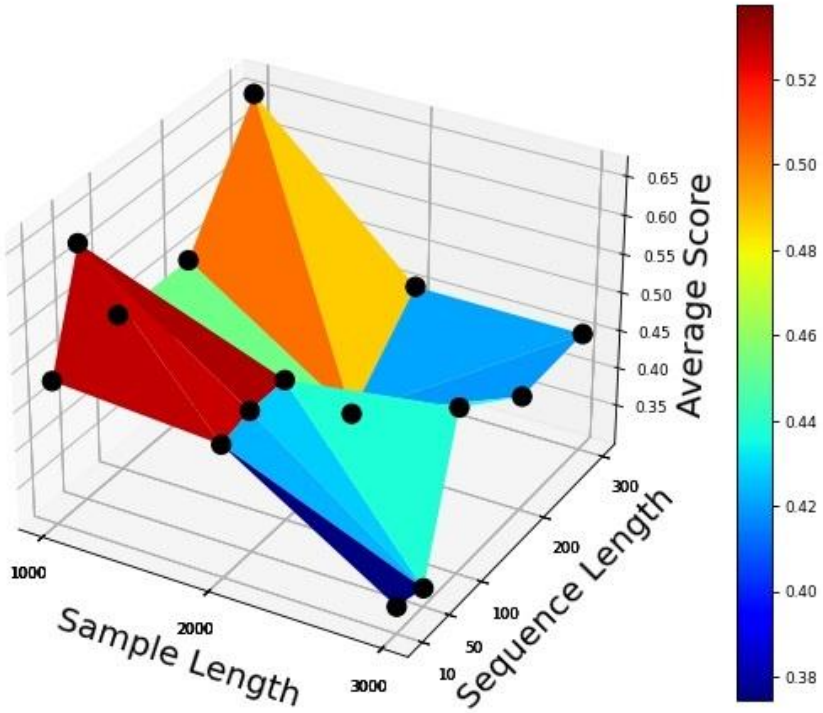


Figure 19
Classical Song Generator Scores



The performance analysis of the jazz song generator was significantly worse. We observed peaks in performance as high as 100% accuracy. But the models achieving these results were producing songs that contained only one or two repeating notes. Meanwhile, the models that produced songs we thought were subjectively good had accuracies between 20% and 60%. We believe that this issue is caused by bad samples in the jazz corpus. As mentioned earlier, some songs in the corpus would repeat the same note over and over. This means that the classifier associated repeating notes with jazz music. And the generator would sometimes use this pattern to create songs with repeating notes.

It could be argued that the classifier would be partially if the criteria is inverted. Any songs that are assigned a low accuracy or score are less likely to contain repetitive notes and are more likely to be enjoyed by a listener. But more importantly these results show that the quality of the training data is vital for a successful music generator or classifier.

Figure 20
Jazz Song Generator Accuracies

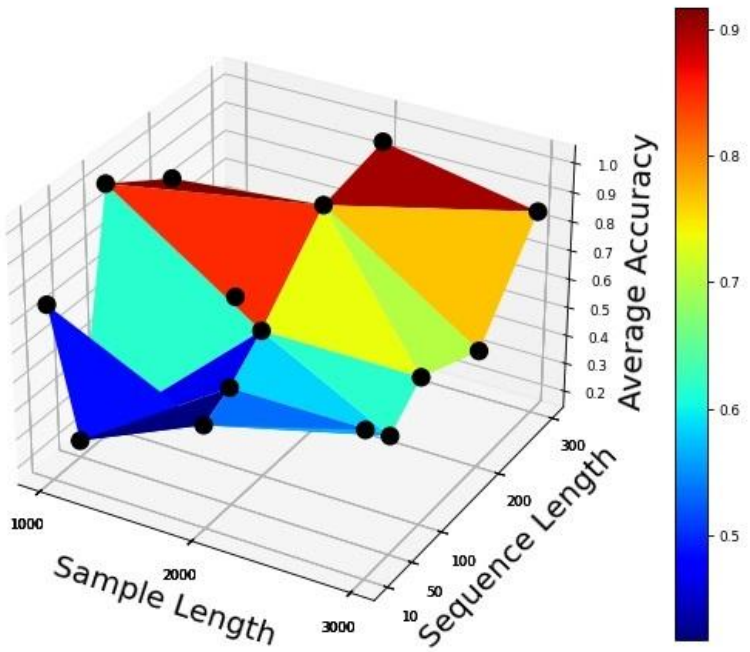
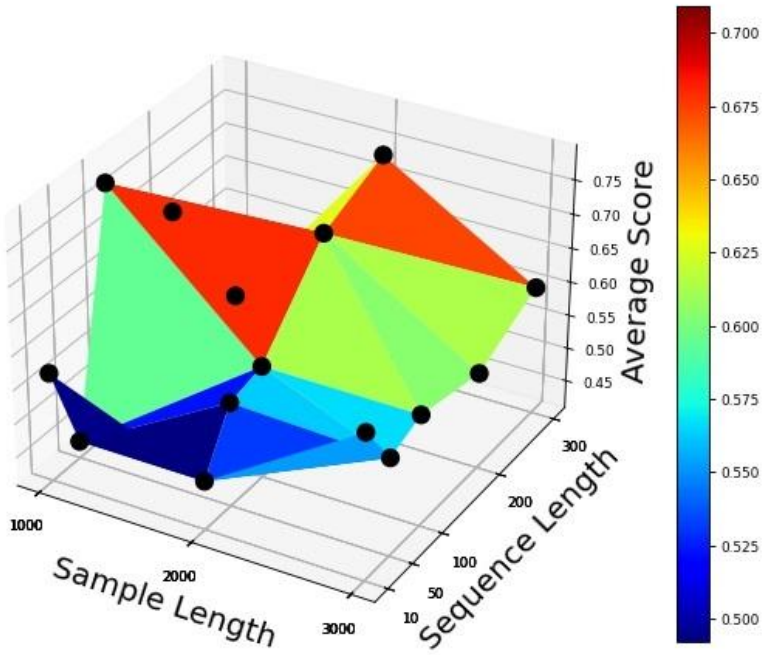


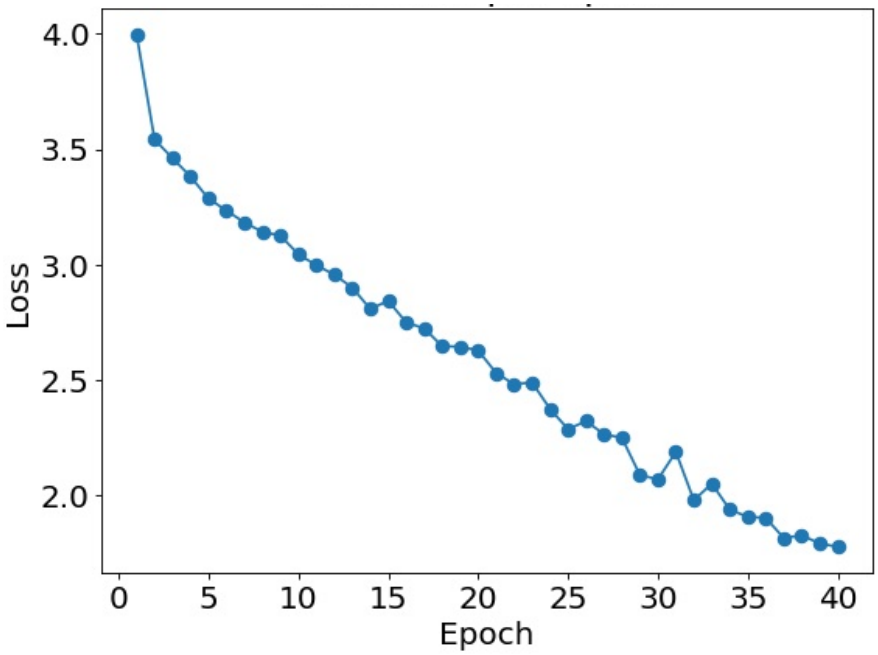
Figure 21
Jazz Song Generator Scores



The Jazz corpus scores and accuracy follow the same trend of having a positive correlation, but differ in its peaks. This corpus peaks at 1000,200 but also has high scores at 2000,300. While the Jazz corpus appears to have higher scores overall, the generated music sounds random. We discovered that songs in the Jazz dataset consisted of many repeating notes, which may be giving our classifier misleading results, as it was trained on the same dataset. We observed that songs that repeat the same note were always classified as Jazz, which indicates that the repeating songs trained on the dataset likely lead to overfitting.

A final observation is that the loss of a model, or the predicted error, decreases as we train on more epochs. This number starts to flatten off around 40 epochs. This is why we chose to train all of our models with 40 epochs. If we feed the neural network a dataset of more complex songs, we would need to train for significantly more epochs before the loss started to flatten out.

Figure 22
Loss vs Number of Epochs



Conclusion

Our results confirmed that an RNN can be an effective way to generate new music. Depending on the input sequence, the model was able to produce songs with good qualitative results. We also showed that traditional machine learning models are capable of learning the differences between music genres and classifying songs by genre. However, we failed to show that a traditional machine learning classifier can accurately determine the quality of a generated song. The classifier was unable to detect when the model was overfitting. And both the classifier and generator were very sensitive to the quality of the dataset.

Future Work

In the future, we plan to address a number of the challenges we faced when training our models. While training, we opted to reduce the complexity of the songs but only extracting the pitch of each note and stripping away all other information such as note lengths, rests, and tempo. This reduced the amount of information that could be learned from the data and resulted in primitive songs being generated. This was an acceptable tradeoff for the initial experimentation we performed in this project. But in future projects, we reintroduce the more complex features to the dataset and attempt to generate more sophisticated songs.

An important finding of this project was the importance of a good dataset. For future works, we will develop techniques to better curate the corpus. We can easily detect songs that are too repetitive by analyzing the frequency of each note in the song. And we can detect if a song is a variation on an existing song in the corpus by analyzing their similarity.

We would also like to experiment with different architectures that are better at determining the quality of a generated song. One of the most promising architectures is a Generative Adversarial Network (GAN.) This is a technique that trains a generator and a classifier against each other. The classifier is trained on the output of the generator and vice versa. This results in a classifier that can adapt to the generator as it improves and can overcome the issues that we faced, such as overfitting.

Lastly, we would like to dedicate more resources to training the model. In this project, we were limited on computing power and were only able to train models on a small subset of our corpus. In the future, we plan to gain access to a High Performance Computing (HPC) cluster to enable more thorough training of the neural network.

References

[1] S. Skúli, "How to Generate Music using a LSTM Neural Network in Keras", Medium, 2017. [Online]. Available: https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5. [Accessed: 04- May- 2021]

[2] M. Cuthbert, C. Ariza and L. Friedland, "Feature Extraction and Machine Learning on Symbolic Music Using the music21 Toolkit", in 12th International Society For Music Information Retrieval Conference, Miami, Florida, 2011, pp. 387-392.

[3] C. McKay, "Automatic Genre Classification of MIDI Recordings", Graduate, McGill University, 2004.

[4] "Dream team: Combining classifiers", Quantdare, 2021. [Online]. Available: https://quantdare.com/dream-team-combining-classifiers/. [Accessed: 08- May- 2021]

[5] M. Stamp, 2021.