5-2019

Deep Learning vs Markov Model in Music Generation

Jeffrey Cruz

Student: Jeffrey Cruz

Advisor: Professor David Paul Benjamin

Major: Computer Science

Graduation Date: May 2019

Pace University Honors Thesis:

Deep Learning vs Markov Model in Music Generation

Abstract:

Artificial intelligence is one of the fastest growing fields at the moment in Computer Science. This is mainly due to the recent advances in machine learning and deep learning algorithms. As a result of these advances, deep learning has been used extensively in applications related to computerized audio/music generation. The main body of this thesis is an experiment. This experiment was based on a similar experiment done by Mike Kayser of Stanford University in 2013 for his thesis "Generative Models of Music" where he used Hidden Markov Models and tested the quality/accuracy of the music he generated using a music composer classifier. The experiment involves creating Markov models for music generation and then creating new models that use deep learning algorithms. These models were trained on midi files of piano music from various composers and were used to generate new music in a similar style to the composer it was trained on. In order to compare the results of these models quantitatively, the music generated by these models was passed to a classifier in order to see which technique create a model that makes music that is correctly classified as being from the composer they were trained on. The results of this experiment showed that the classifier was able to more accurately label music generated by the deep learning model than the Markov model as being from the composer the model was trained on.

Table of Contents:

List of Figures:

All figures in this paper were produced by me, using the matplotlib and seaborns python packages.

Introduction:

Artificial intelligence is one of the most widespread and fastest growing fields out there. Some out there refer to A.I as the “new electricity” of the modern era. This is a reference to how A.I is being spread and applied to almost every field imaginable including banking, entertainment, driving, medicine, and almost anything else you can think of. One of the most exciting applications of A.I, in my opinion, is the use of A.I in the generation of music and art. Most of A.Is developments are really due to a specific subfield of A.I called deep learning. Deep learning uses a specific A.I algorithm called neural networks in order to learn and do what they do. These neural networks are also sometimes called feed-forward networks since they take input in the first layer, its fed to the middle sections of the network where all of the analyzing and learning happens, and then you get your desired (hopefully) output at the end layer. This architecture is made to mimic the architecture of the real brain, with the neural networks being analogous to the connections between the axons and dendrites of real brain cells. Deep learning is really good at recognizing and learning complex patterns, which makes it really good at generating things that follow patterns such as music and images. When it comes to A.I, one of the great fears is that it will grow so good that it will be better than humans and replace them, making those humans lose their job. This fear is especially prevalent in music generation, where many believe that one day computers will make music better than any human and that all music will end up being made by computers. Music generation is not one of the fields where A.I performs best, but it is one of the most interesting fields in which it's being applied. This paper does a test to see if deep learning could produce music that could fool a trained classifier into thinking that music generated by a computer was actually composed by a specific composer. However, instead of classifying the music as merely “Real” vs “Fake (generated by a

computer)", I opt to do classification between the 4 composers. I theorize that if deep learning is at a level where it could learn the patterns that are very iconic to a composer's style, then it should generate music that gets a high accuracy of being classified as being from that composer instead of the other ones. At the same time, I compare the results of deep learning to Markov models; an older algorithm that has been applied to music generation before.

Literature Review:

Computer generated music has been the topic of many computer science research papers. Due to the explosion of deep learnings popularity in recent years, many researchers have done studies on the application of deep learning on computer music generation. In March of 2017, Li-Chia Yang and others built on top of the existing Google WaveNet model. Their research investigated the use of convolutional neural networks for generating melody as a series of MIDI notes in a symbolic domain. In addition, they also experiment with generative adversarial network architectures to learn the distribution of melodies and generate melodies. In Yang's paper, they also propose a mechanism to exploit available prior knowledge that allows for models to generate melodies from scratch, by following a chord sequence, or by conditioning on the melody of previous bars. In summation, they call the resulting model MidiNet. In Yang's paper, the researchers conducted a user study to compare the melodies generated by MidiNet and Google's MelodyRNN model and find that the results show that both models had comparable results. Users found that MidiNet created more interesting melodies, while MelodyRNN created more realistic ones. That is to say that, MidiNet created melodies that were more appealing to the users they surveyed, however, those melodies had elements to them that made it clear it was produced algorithmically. MelodyRNN, on the other hand, created melodies that had fooled

many people into thinking it was made by a human, it that sense it created more “realistic” music.

Mike Kayser was a student at Stanford in 2013. In his thesis, Kayser proposed two Markov-based music generation models. His models capture key and chord progression information and are trained in a completely unsupervised setting. In his thesis, Kayser sets up an experiment in which his models produce music meant to mimic the styles of famous composers. He sets up a composer classifier model that is trained to identify music from different composers and sees if the music generated by his models is accurately classified as music by the composers he was trying to mimic. The results of his experiment find that his models perform reasonably well, both models reaching an average of around 60% accuracy in composer classification. He finishes his thesis by mentioning the room for improvement in his models and experiment. This thesis was the inspiration for the experiment that my thesis focuses around. I build on top of what Kayser did by adding in a deep learning model to the experiment for comparison to results achieved by a Markov model.

Research Question:

The question being researched through this paper is “Can deep learning be applied to music generation?” With a particular focus on the follow-up question “If so, does it perform better than traditional techniques in computer science used for music generation?”. The computational generation of music has been done and researched extensively using state space theory, genetic algorithms, etc. Before the recent boom in machine and deep learning, a majority of computational music generation was done using mainly Hidden Markov Models. My research will show if and how deep learning can also be applied to music generation, and whether or not it

can be shown quantitatively that deep learning performs better than Markov models at music generation. Past research has been done in recent years that use deep learning for music generation, for example, a Google research company based in London called DeepMind developed WaveNet. Wavenet is a tool that uses deep learning to generate raw audio. Other researchers have built on top of WaveNet such as Li-Chia Yang, who developed MidiNet which uses a recently developed type of neural network called a Generative Adversarial Network in order to produce algorithmically-generated audio that is made to be as identical as possible to authentic audio.

Hypothesis:

I expect the results of my experiment, and overall the results of this paper, to show that the models using deep learning algorithms are able to better generate music of a higher quality than the music generated by the hidden Markov model. While “of a higher quality” is subjective, and can vary from person to person, quantitatively I will attempt to show that music generated by deep learning will be more “authentic” and more closely resemble the original source audio. If this is true, then the classifier will have a higher accuracy when classifying the composer of the music generated via deep learning, and a lower accuracy when classifying the composer of the music generated via Markov models.

Methodology:

Gathering Data:

In order to train any music generative model, we need data for it to learn on. The first thing I did was scour online database repositories such Kaggles online datasets, the University of California Irvine Machine Learning Repository, and other online resources of this nature. However, I was searching specifically for music files of specific composers that were very scarce in these database repositories which were general purpose and hosted millions of kinds of datasets. To make things even harder, I needed the music files to be MIDI type files, so music files such as MP3s were useless to me for the purposes of this experiment. I also specifically only wanted pieces that were entirely or at least mostly played by piano. I then specifically looked at websites dedicated to hosting music or websites dedicated to certain composers in order to find the data I needed. This search led me to online music repositories which had just the kind of data I was looking for: midi files of classical composers. I found other websites that hosted free midi files and I downloaded them into folders which resulted in me getting 503 midi files of piano music composed by Johann Sebastian Bach, 122 midi files of music composed by Wolfgang Amadeus Mozart, 245 midi files of music composed by Frédéric Chopin, and 200 midi files of music composed by Ludwig van Beethoven. While originally I had wished to include more composers such as Johannes Brahms and Tchaikovsky, I was only able to find a substantial amount of data for the 4 composers mentioned earlier.

Creating the Deep Learning Models:

The next thing I did was work on the deep learning model for generating music. From my previous studies and experiences in using deep learning, I immediately decided that using a

Recurrent Neural Network (RNN) would be the best type of deep learning model to use on music data. This is because Recurrent Neural Networks are built in order to have a concept of "memory", meaning that as it processes parts of the data, it can take into account the part of the data that came before it. This is crucial when processing music and especially when generating music as the key concepts of music such as harmony and rhythm all take into account more than just one note of music at a time. I had attempted building a number of different Recurrent Neural Network models myself, however, it was difficult determining which specific architecture to use. Neural Networks are made of layers of "neurons", and there are a huge amount of different combinations and formations of layers that can be put together. One of the biggest challenges in deep learning is that many aspects of it are still being figured out. There is no exact science or formula yet that can tell you what architecture/layering will give you a good model. The only thing you can do is trial and error, that is to just try an architecture and see what results it gives you. So I decided to search on GitHub and research other papers about music generative models for studies that had positive results using a Recurrent Neural Network for music generation in order to look at the architecture of their models. This search led me to research by Daniel Johnson, a machine learning engineer from San Francisco.

In Johnson's research, he proposes his recurrent neural network architecture which he calls a "Biaxial Recurrent Neural Network". You can read more about the details of his specific architecture from his paper, but what I did was recreate his model architecture in Theano; a Python library for optimized machine and deep learning algorithms. I did, however, have to make some changes to the model's design. The original architecture of the model was made to work on a computer that had greater capabilities than I have access to. I had to cut down on the size of some of the layers in order to get the model to run on my hardware. This did not impact

my results or have any significant effect on the performance of the model, if anything, by cutting down on the size of the layers I reduced the risk of overfitting in my model. Once I had the model built, the next step was to create 4 copies of it, one for each composer. Each copy then had to be trained on the collection of music for their respective composer. That is to say that there was 1 model dedicated to learning from the Mozart midi files, another model dedicated to learning from the Beethoven midi files, and so on and so forth. This was in the hopes that the models would then learn how to generate music in the same "style" as the composer. Training time is another one of the hurdles that come with using deep learning. I trained the 4 models on my personal laptop which has an Intel i7-7700HQ CPU, and an Nvidia GTX 1060 GPU. Theano allowed me to use my GPU to help make processing faster. Each model was trained for 10,000 iterations. which took roughly 12 hours, meaning that it took 2 days to finish training the 4 models. While training, the models gave a sample of music from 5-30 seconds of length every 100 training iterations below 1000, and then for every 500 iterations past 1000. It also gave a parameter file which showed in binary the parameters that the model was learning and fine-tuning as it trained.

Training the Deep Learning Models and Generating Music:

Now that I had 4 fully trained deep learning models for Bach, Beethoven, Chopin, and Mozart. I had each model generate 100 songs. This took roughly 24 hours to do in total, which I separated over a couple of days. The music generated from these models were very impressive from my standpoint, I found it very pleasant to listen to and no two songs were similar to each other. Each song generated by the model was very distinctive. The only detriment I can see here which makes it very telling that this is still music generated by a computer instead of a human is that at times it can be repetitive or stuck on a certain pattern for a while. There are also abrupt

and out of place silences at times. Overall, I was very happy with the music generated by the deep learning model and was ready to move on to the next part of the experiment.

Markov Model for Music Generation:

The next part needed for this thesis is the Markov model. Hidden Markov Models used to be the standard for audio/music generative models because the processes behind Markov models are very suitable for music composition. Markov models capture the statistics behind how likely the model is to be in a certain state based on the other states that are in its "Markov blanket". When applied to music it basically works by calculating the probability of playing a certain note based on what note was just played. The Markov Model is not necessarily "trained" in the same way a machine learning model or deep learning model is. While machine learning models such as the classifier made before are "model-based", the Markov model is more "instance-based". That is because it does not necessarily learn directly from data, but rather the data is used to populate a table that the model uses to make its output. Therefore, the data needs to be processed again but in a different way than it was for the classifier or for the deep learning models. Again the music21 Python library is used to help make the Markov model by using its midi processing capabilities. Basically, it goes through all the data and populates a table that tells us how frequently each note follows every other note. It also captures harmony and duration in order to make the music pleasant to the ear and to make it sound as realistic as possible. Four copies of this model were made for each composer, and were trained solely on the music from each composer in order to generate music in the same "style" as the composer it was trained on. I use these models to generate 100 songs for each composer, just like with the deep learning model.

This completed my generation of the test-set for this experiment, with 800 generated songs. 200 for each composer made evenly by the deep learning model and the Markov model.

Feature Extraction for Classifier Model:

Before I can begin working on the classifier, I need to prepare my data and format it so that it will be readable by a classifier model. The hardest part of making a classifier model is usually the data processing and feature extraction. A classification model cannot look at the raw midi file, details or features of the data must be extracted from the raw midi file in order to be processed by the classifier. In order to extract these features, I wrote a python script that utilized methods from the music21 library developed at MIT. The following is a description of the music21 library from their official website:

> "Music21 is a set of tools for helping scholars and other active listeners answer questions about music quickly and simply. If you've ever asked yourself a question like, "I wonder how often Bach does that" or "I wish I knew which band was the first to use these chords in this order," or "I'll bet we'd know more about Renaissance counterpoint (or Indian ragas or post-tonal pitch structures or the form of minuets) if I could write a program to automatically write more of them," then music21 can help you with your work." (*MIT, referenced from https://web.mit.edu/music21/)*

This aptly describes the current problem I am trying to solve; from the raw data how can I tell what Bach does that makes his music so "Bach-like"? And similar for the other composers I am working with. Music21 has hundreds of different features that it can extract from midi files. All of these features, however, are either not applicable to the music I am working with, or not suitable features for processing by a neural network. I first skimmed down the list of features to

around 80 by only taking features that are 1-dimensional, as in they can be represented by a single number instead of a tuple or a vector of values. I took these 80 features at first and began processing the data using my script. The processing time for this was much greater than I expected, so I amortized this processing time by using multiple Ubuntu virtual private servers from DigitalOcean. This allowed me to run my script on 4 different machines. Even doing this, it took roughly 3 days of non-stop compute time to extract the 80 features from the >1000 midi files. These features were extracted into a csv file for use in later stages of this experiment. I would later have to use this processing script again to extract features from the new music generated by the deep learning model, and eventually the Markov model. So I used a statistical technique called backwards elimination which cut down my number of features from 80 to 45 by removing features that were non-significant either because it had high collinearity to other features thus making it redundant, or because it was not predictive of what composer the music came from. This would help cut down on the processing time for future data, and would also decrease the training time on the classification models. Lastly, of the remaining 45 features, 5 of these were mostly null or 0 values for most of the midi files, so these features were removed leaving us with a total of 40 features. Two of these features are the name of the midi file, and the composer. The name is purely for organizational purposes, and the composer will be used as the output vector, so this leaves us with 38 variables to be the columns for our matrix of features. The final remaining 38 features are listed in the appendix, and a description of all of these features can be found in music21s documentation.

Creating and Training the Classifier Models:

In various fields of computer science, but prominently in machine learning, there is a theorem passed around that is called the no free lunch theorem. What this theorem states is that

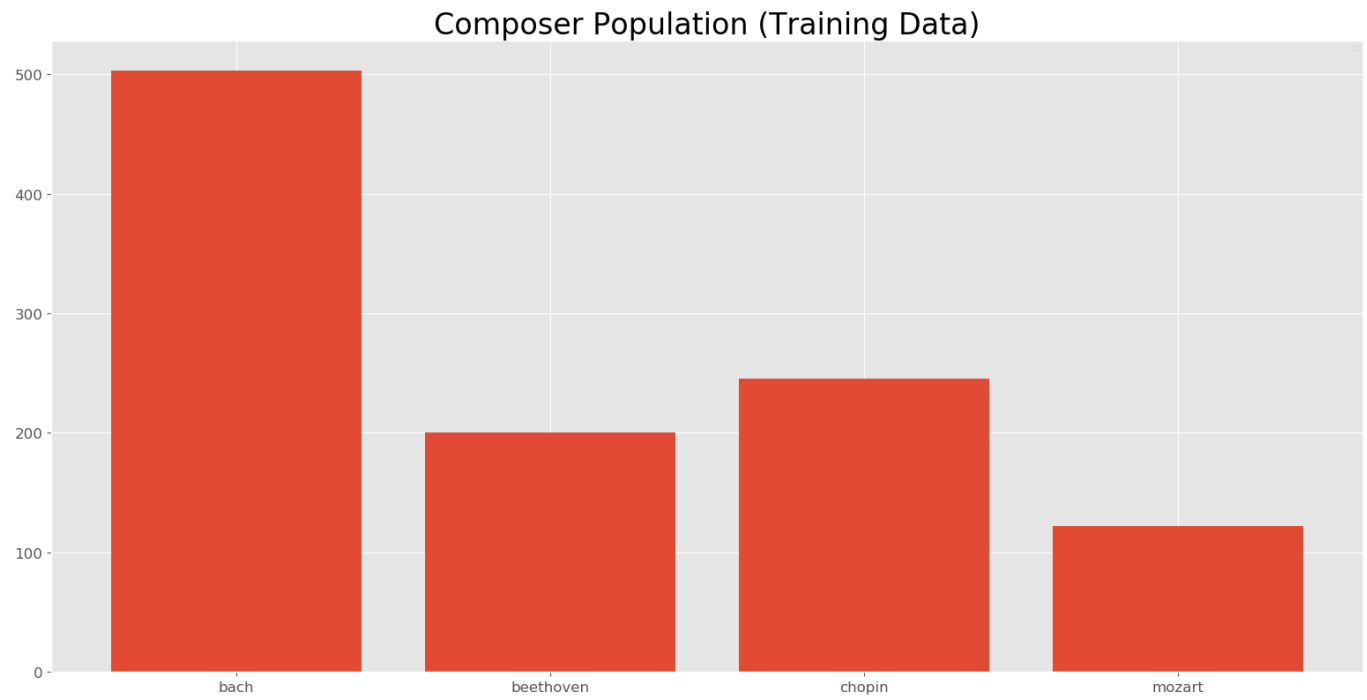
usually is not sufficient to depend solely on one model when tackling a problem and that it is usually more beneficial to use an ensemble of models. I abided by this theorem and decided to try seven different machine learning algorithms for use as a classifier in my experiment. These seven algorithms are:

1. XGBoost Classification
2. LightGBM Classification
3. Naive Bayes Classification
4. K-Nearest Neighbors Classification
5. Random Forest Classification
6. Linear Discriminant Analysis
7. Logistic Regression

Fortunately, many of these algorithms are already implemented in the scikit-learn python package. LightGBM, and XGBoost, also have their own dedicated python package that includes the implementation of these algorithms available for download.

I performed the pre-processing that was explained earlier on all of the training data (the same data that the music generative models were trained on). The output was a 1070 x 40 csv file. 1070 being the number of midi files that I had in total for the four different composers, and 40 being the number of features I extracted for each midi file (including the ID and Composer). The following is a graph showing the distribution of composers in the training set:

Figure 1.1: Composer Population (Training Data)



As shown in this graph, and as was stated earlier, most of the data that I was able to find was music composed by bach. While I did not expect this to have any adverse effects when it comes to the music generative models (because there was a separate model for each composer), when it comes to the classifier this may lead to each classifier being able to recognize bach better than it can recognize the other composers. It is also possible for the model to overfit to the training data, and thus predict mostly bach since the data it was trained on was mostly music by Bach. I take steps however to circumvent and avoid this possible pitfall and prevent overfitting.

When creating a classification model, one of the most important aspects is choosing the hyperparameters. A hyperparameter is a parameter of the model itself, for example, in K-Nearest Neighbors the only hyperparameter is K (the number of neighbors used for comparison when performing classification). In order to choose my hyperparameters, I performed grid search on the seven models. Grid search is implemented in scikit learn. It goes through various different combinations of hyperparameters for each model, fits the model with those parameters to a given

training set, and returns the set of hyperparameters that performed best on a given test set. While fitting the model it also does K-Fold Cross Validation to prevent overfitting. What K-Fold Cross Validation does is that for each iteration of grid search, it cuts the data into a training and validation set. Meaning that the models are only trained on only a subset of the entire data, and this subset changes with each iteration. This prevents the model from growing stagnant and merely "memorizing" the dataset instead of learning, thereby preventing overfitting. The time it takes for grid search grows exponentially with each new possible hyperparameter value I want to try, so I limited the range of hyperparameters for each model in order to decrease on the time grid search would take. The following image is a snapshot of the source code for this experiment that shows what the hyperparameters used for each model was:

Figure 1.2 Classifier HyperParameters

```
xgb = XGBClassifier(n_estimators=800,learning_rate=0.05,max_depth=15)
xgb.fit(X_train,Y_train)
light = LGBMClassifier(n_estimators=800,learning_rate=0.05,max_depth=13)
light.fit(X_train,Y_train)
forest = RandomForestClassifier(n_estimators=300)
forest.fit(X_train,Y_train)
lda = LinearDiscriminantAnalysis(solver='lsqr')
lda.fit(X_train,Y_train)
knn =  KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train,Y_train)
gauss = GaussianNB(var_smoothing=1e-09)
gauss.fit(X_train,Y_train)
logr = LogisticRegression(solver='liblinear',penalty='l1',multi_class='auto')
logr.fit(X_train,Y_train)
```

Next, I wanted to see how well each classification model performs on the training set. I did this by doing an 80:20 split of the training data, using 80% of it as training data, and 20% as a validation set for the purposes of seeing how well each model performs. I then did K-Fold Cross Validation with 10 folds in order to prevent overfitting. The following graphs show the

mean accuracy of each classification model after cross-validation, and the standard deviation for those accuracies.

Figure 1.3: Classifier 10-Fold Cross Validation Results (Training)

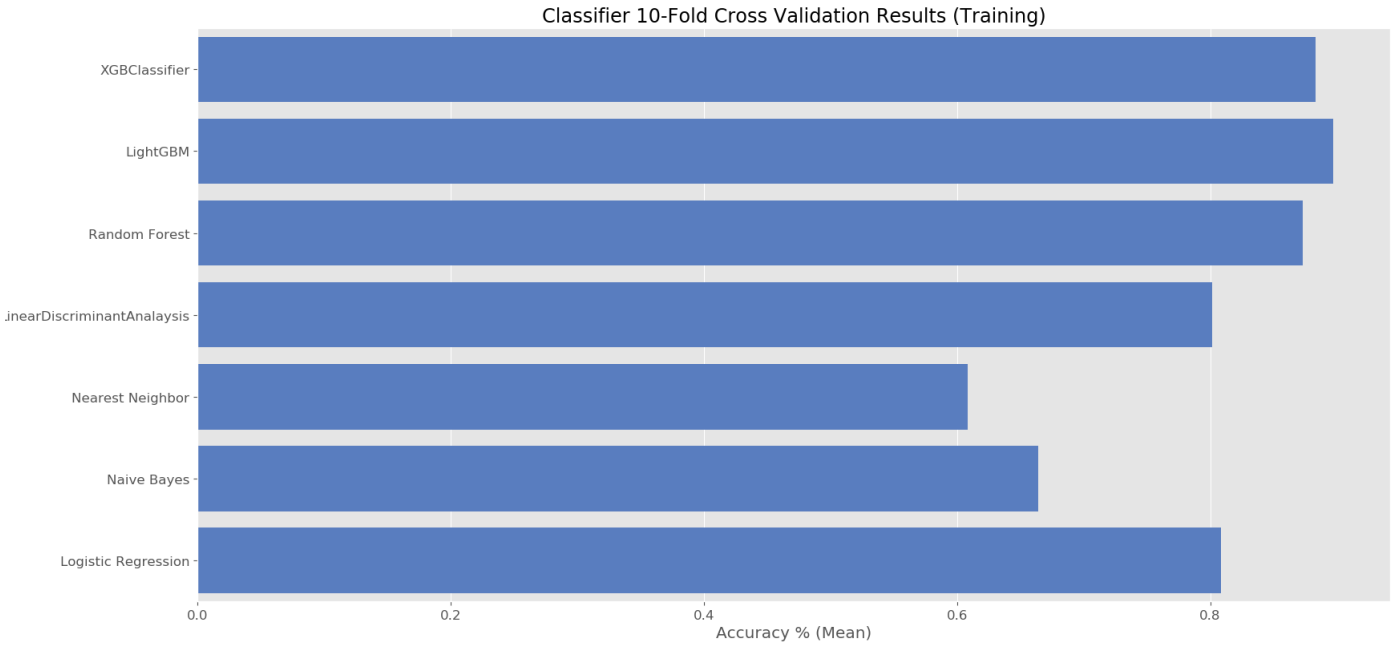
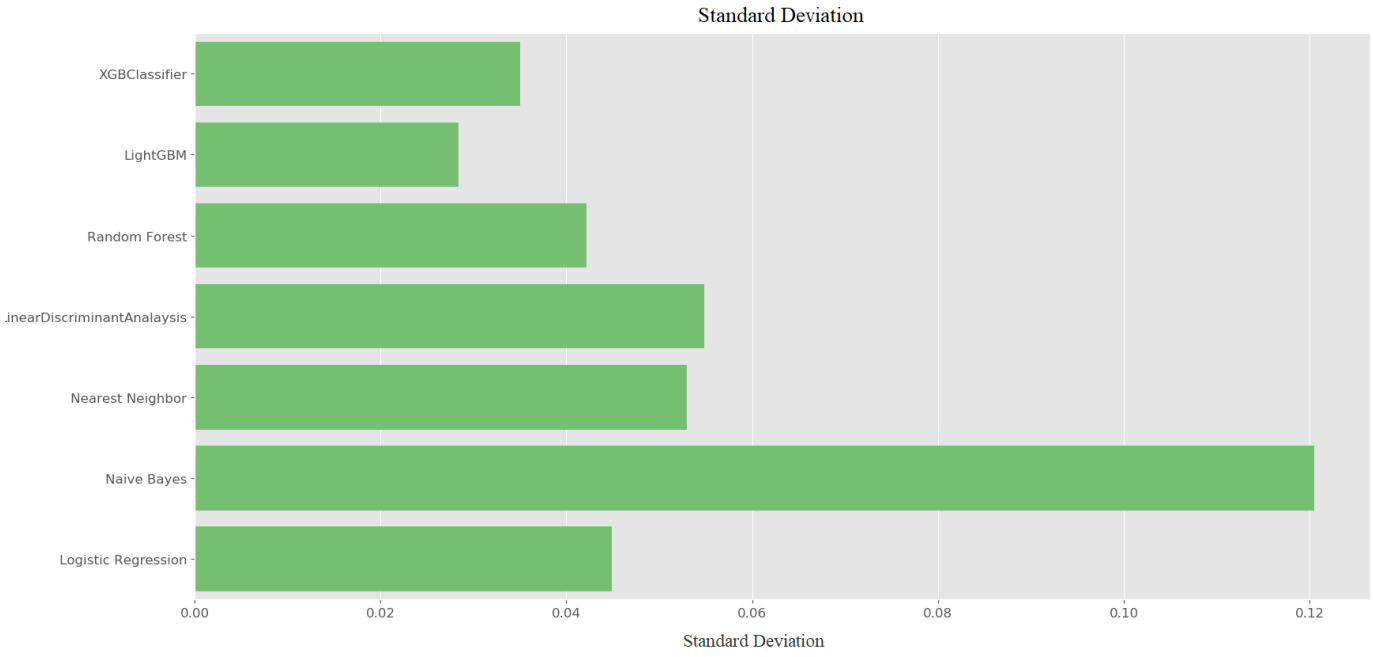
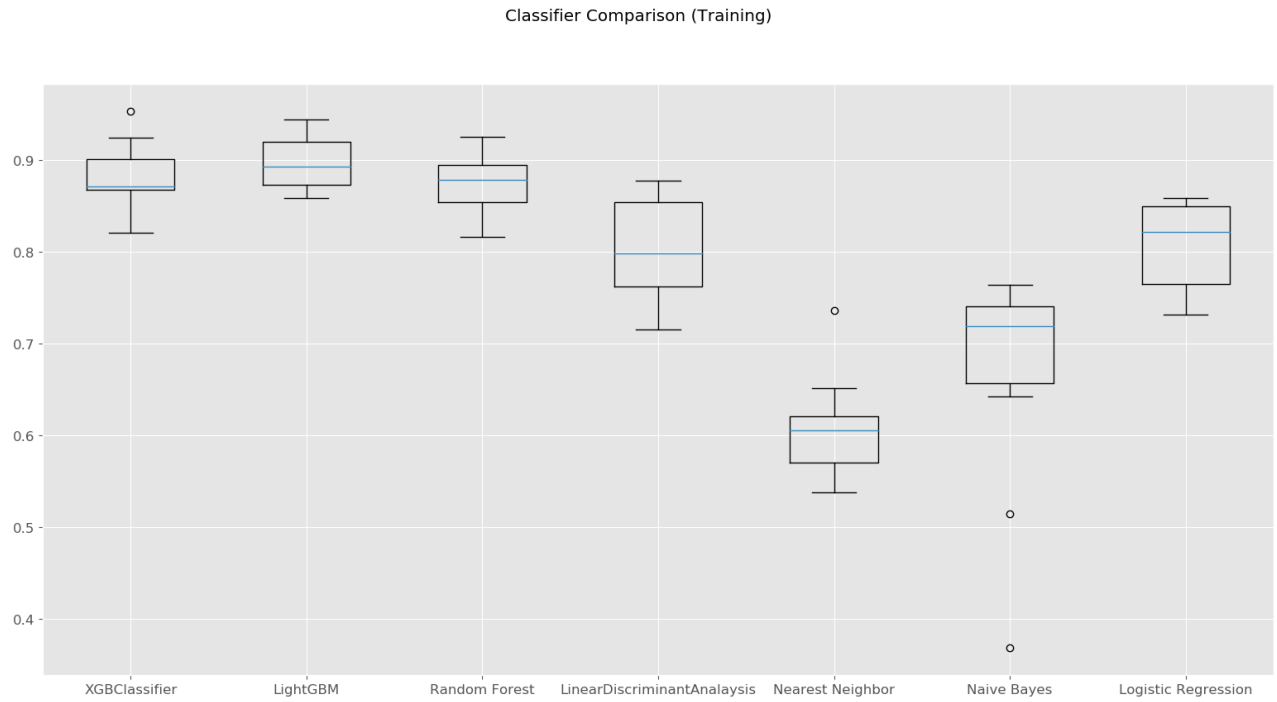


Figure 1.4: Classifier 10-Fold Cross Validation Standard Deviation Percentages (Training)



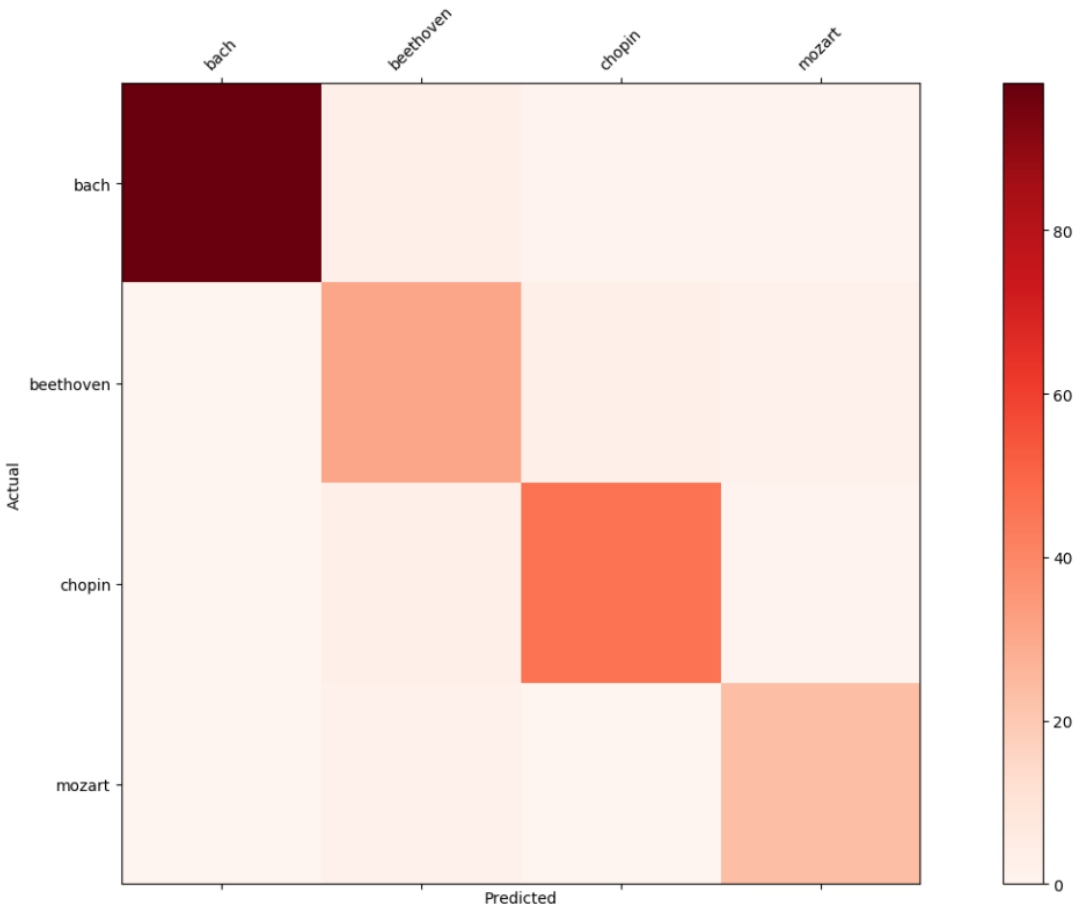
As we can see from the graphs the best performing model was LightGBM with around a 90% accuracy, roughly 3% standard deviation. XGBoost and Random Forest were the next best performing models with roughly 88% accuracy and roughly a 3.7% and 4.2% standard deviation respectively. Logistic Regression and Linear Discriminant Analysis had roughly an 81% and 80% accuracy respectively, with a 4.3% and 5.4% standard deviation respectively. Naive Bayes scored roughly 66% accuracy and a 12.1% standard deviation. K-Nearest Neighbors was the worst performing model with 61% accuracy and 5% standard deviation. The following is a box plot representation of the models' results on the training data composed using the 10 accuracy scores from each fold in K-Fold Cross Validation for each model:

Figure 1.5: Boxplot of K-Fold Cross Validation Results



Lastly, here is a heatmap for the training results. The heatmap, otherwise known as a confusion matrix, shows how accurately the classifier is for each composer, and how likely the model will confuse one composer for another. The following heatmap only shows the results for the model that performed best in training (which is the LightGBM Classifier):

Figure 1.6: Confusion Matrix (Heatmap) For LightGBM Classifier (Training Data)



A version of this confusion matrix with the numbers for how often each kind of prediction occurred is shown below:

Figure 1.7: Numerical Confusion Matrix (Training)

Index	bach	beethoven	chopin	mozart
bach	89	2	1	3
beethoven	4	26	1	1
chopin	1	0	52	1
mozart	2	4	1	26

Classification on Generated Data:

Now that the models are trained, it's time to conduct the experiment itself. Using the same pre-processing script used for the training data, I processed the music generated by the deep learning models, and the music generated by the Markov model. The result being two 400 x 40 csv files which contain the 40 extracted features for each of the 400 generated songs in each file (800 total). These two sets are the two test sets for this experiment, 400 songs from each composer generated by Markov models as test set 1, and 400 songs from each composer generated by deep learning as test set 2.

For each of the seven classifiers that were trained earlier, I went back and trained them one last time, but this time using 100% of the training data rather than the 80:20 split used for validation. I then took these seven models, and used them to predict the composer for the 400 songs in the Markov test set, and then for the 400 songs in the deep learning test set. This concludes the methodology.

Results, Discussion, and Limitations:

The seven classifiers made earlier were used on the two test sets (the Markov test set, and the deep learning test set). As a reminder, there were 8 different generative models made throughout this experiment. The first group of 4 models was made using deep learning techniques, with each model being trained solely on the music of one composer. The same goes for the four Markov models. This was done so that the music generated by those models would be made in the same style as the composer the model was trained on. This is in hope that the classifier would accurately predict the music generated by these models as being composed by the composer they were trained on. It was my prediction that the music generated by the deep

learning model would be of higher quality both qualitatively and quantitatively. While it would be hard to logically prove that the music of the deep learning model is qualitatively better, it was my prediction that the music generated by the deep learning model would have a higher classification rate than the music generated by the Markov model. This would effectively show quantitatively the superiority of deep learning over Markov modeling in this area.

The results of the classification are shown in the figures below:

Figure 2.1: Classification Results (Markov Test Set)

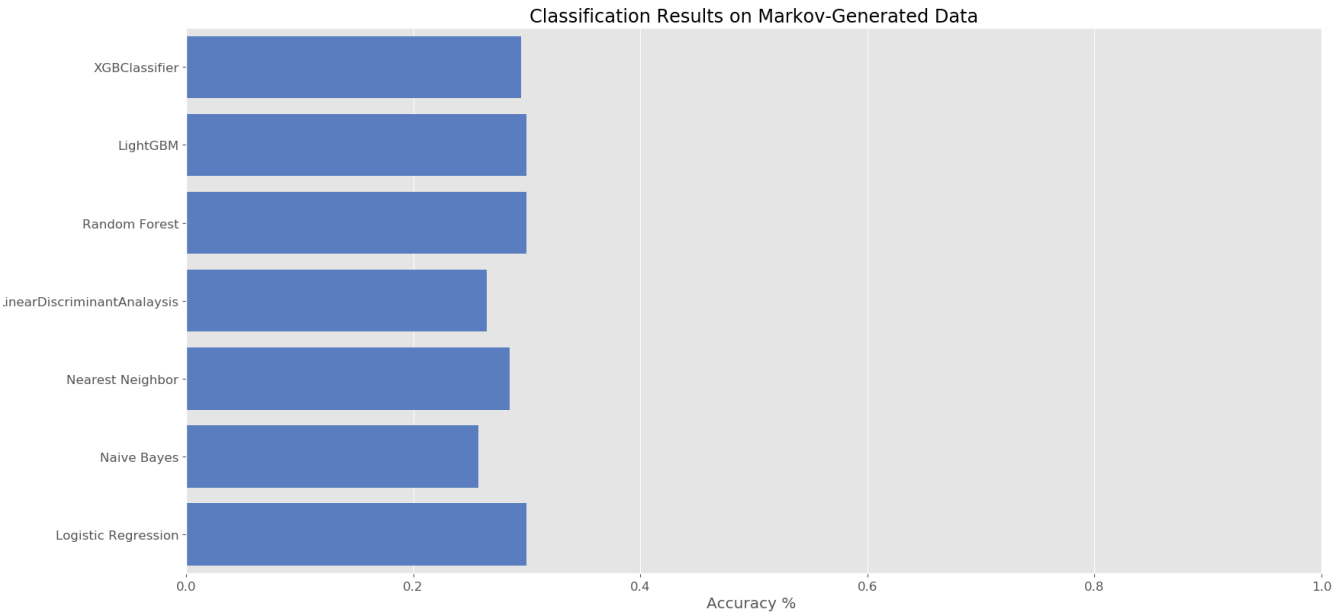


Figure 2.2 Classification Results (Deep Learning Test Set)

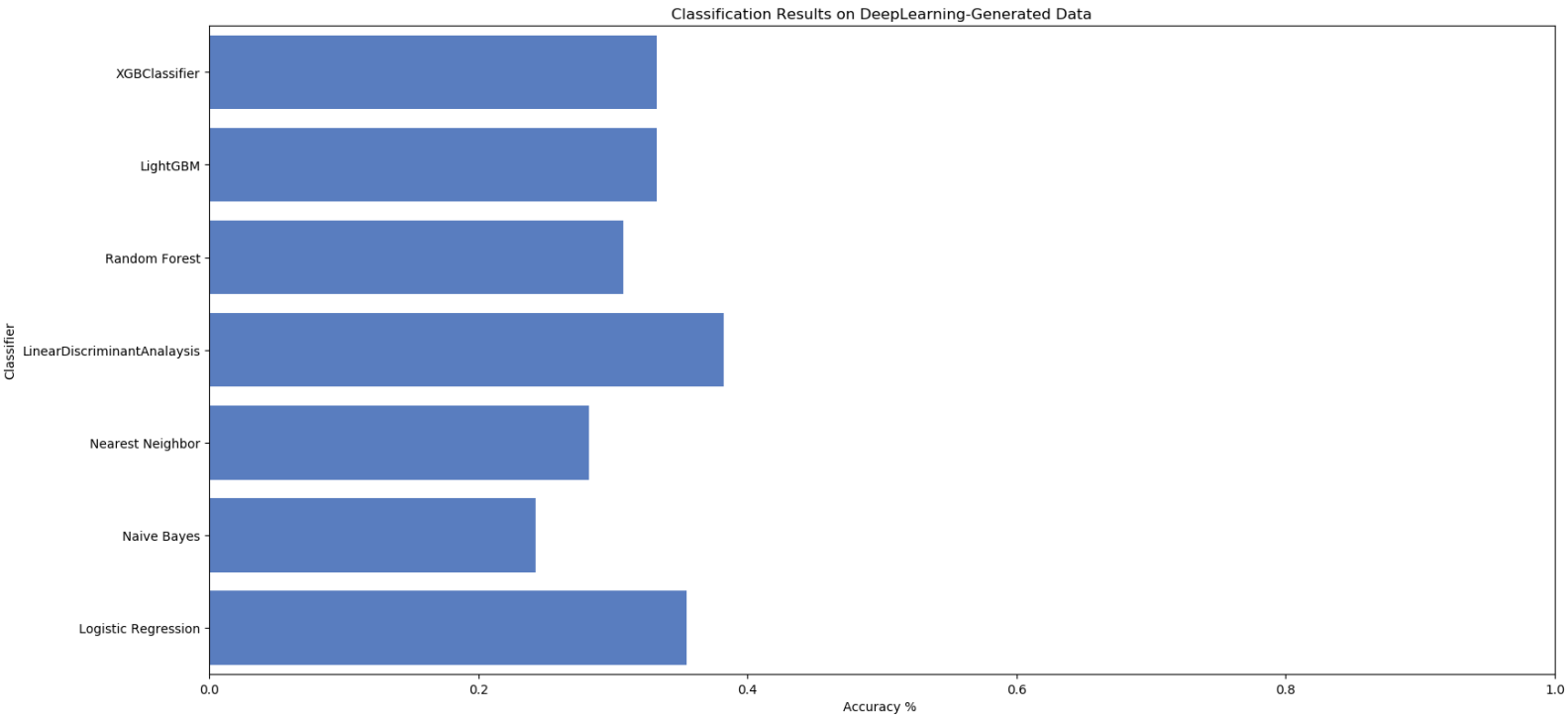


Figure 2.3 Classification Accuracies (Top:Deep Learning, Bottom:Markov)

```
In [110]: dl_accuracy
     ...:
Out[110]:
{'XGBClassifier': 0.3325,
 'LightGBM': 0.3325,
 'Random Forest': 0.3075,
 'LinearDiscriminantAnalaysis': 0.3825,
 'Nearest Neighbor': 0.2825,
 'Naive Bayes': 0.2425,
 'Logistic Regression': 0.355}

In [111]: mk_accuracy
Out[111]:
{'XGBClassifier': 0.295,
 'LightGBM': 0.3,
 'Random Forest': 0.3,
 'LinearDiscriminantAnalaysis': 0.265,
 'Nearest Neighbor': 0.285,
 'Naive Bayes': 0.2575,
 'Logistic Regression': 0.3}
```

While none of these accuracies are by themselves remarkable, these results show that there is a higher accuracy rate for the music generated by deep learning. Each and every model performed better at classifying the deep learning generated music, with the sole exception of Naive Bayes which performed roughly 1% worse for the deep learning test set. Every other model either performed equally as well or up to 12% better. The best performing model when it came to predicting the music generated by the Markov model was LightGBM with a 30% accuracy rate (which it shared with Logistic Regression and Random Forest). This is consistent with the training set results which also had LightGBM as the best performing model. The following confusion matrix gives a more detailed view on the performance of LightGBM for classifying the Markov generated data:

Figure 2.4: Confusion Matrix (Markov Test Set)

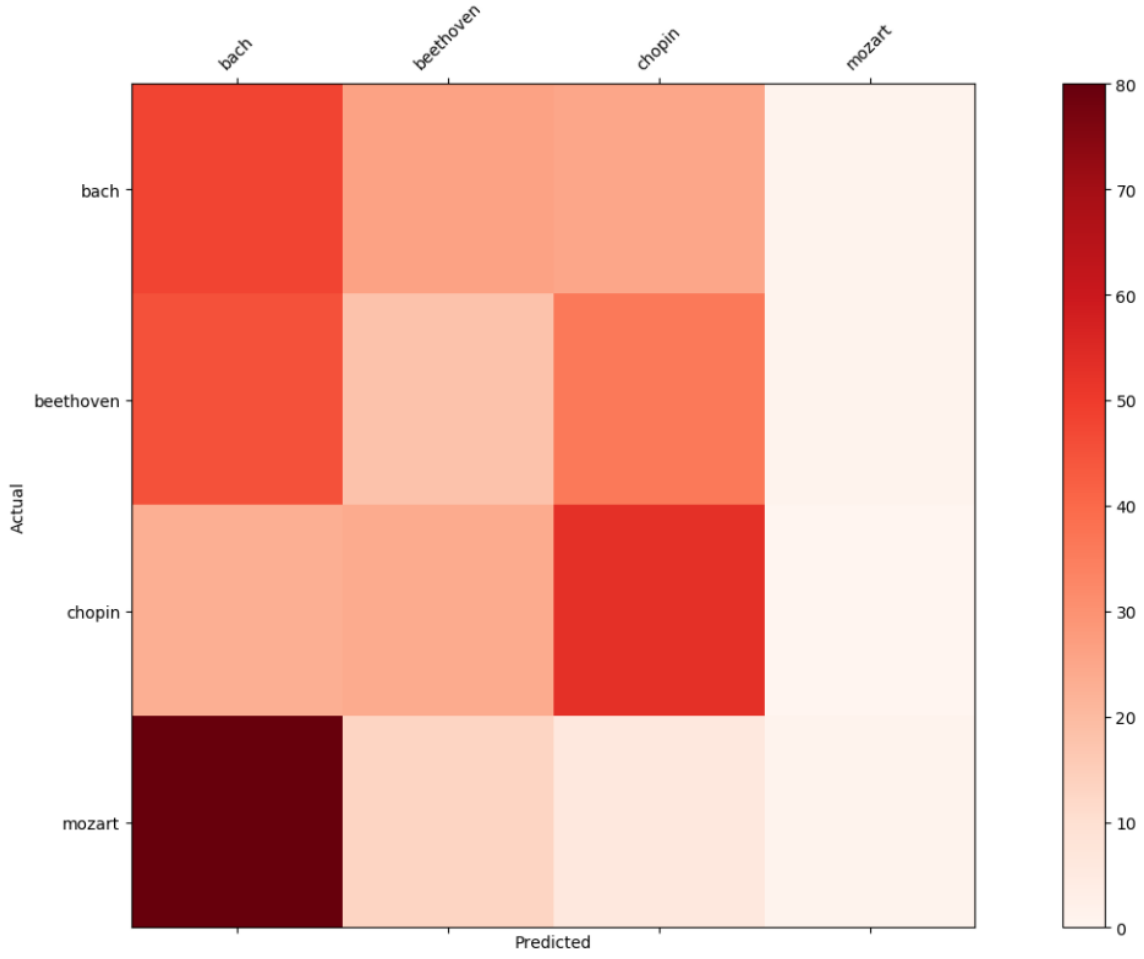


Figure 2.5: Numeric Confusion Matrix (Markov Test Set)

Index	bach	beethoven	chopin	mozart
bach	48	26	25	1
beethoven	45	18	36	1
chopin	23	24	53	0
mozart	80	13	6	1

These results show that when it came to the Markov model, it is clear to see through the confusion matrices that there was a lot of confusion. The in-class accuracies (the accuracy within one label) were 48%, 18%, 53%, and 1%, respectively for Bach, Beethoven, Chopin, and Mozart. The only label that was accurately classified more than half the time was Chopin. The worst performance came when trying to classify music generated in Mozart's style as being from Mozart, with only 1 of those 100 songs generated in Mozart's style being classified as being from Mozart. 80 of them were classified as being from Bach by this classifier. This shows that the Markov Model did not perform well when generating music that would be classified as being from the same composer that the model was trained on

The best performing model when it came to the deep learning data was Linear Discriminant Analysis, with an accuracy of 38%. It outperformed LightGBM for this test set by roughly 5%. This was also surprising as LightGBM was the best classifier in both the training set, and the Markov test set. The following graphs give a more detailed view of the performance of the Linear Discriminant Analysis classifier on the deep learning test set:

Figure 2.6: Confusion Matrix (Deep Learning Test Set)

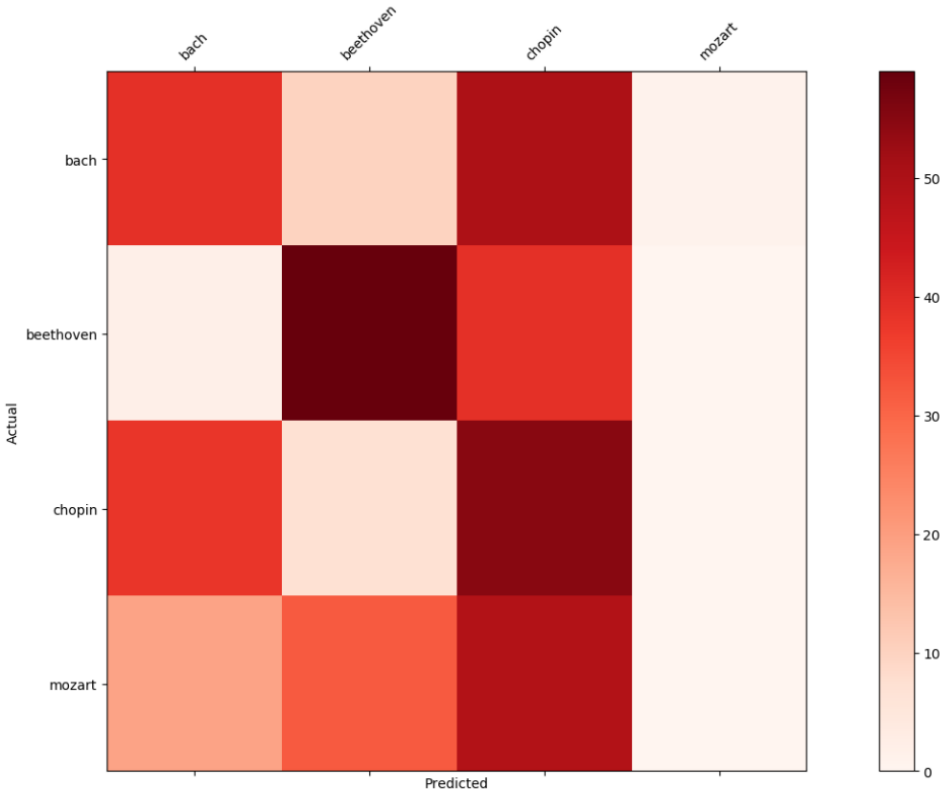


Figure 2.7: Numeric Confusion Matrix (Deep Learning Test Set)

Index	bach	beethoven	chopin	mozart
bach	39	10	50	1
beethoven	2	59	39	0
chopin	38	7	55	0
mozart	19	32	49	0

These results of the best performing model for the deep learning set differ in quite a few ways from the results of the best performing model for the Markov set. The results in the Markov set show more general confusion, while for the deep learning set there is more specific

confusion. Bach is mostly mislabeled as Chopin for the deep learning set. Similarly, Beethoven is mostly mislabeled for Chopin as well. Chopin is mostly mislabeled as Bach. Where the two share results, however, is in Mozart, this time with Mozart having a 0% in-class accuracy. This means that this model did not classify any of the songs generated by deep learning in Mozart's style as being composed by Mozart. Beethoven and Chopin have higher in-class accuracy rates than they did in the Markov results, which Bach has a slightly lower in-class accuracy rate. These results show that there was significant confusion mainly centered around Chopin. It would also appear that neither model was able to capture Mozart's style well. It may seem that the problem might be with the classifier itself, as for the deep learning set, the Linear Discriminant Analysis classifier only predicted Mozart once. This one prediction wasn't even correct. However, the total accuracy of this model, and of almost every other classifier model saw an improvement when it came to classifying the deep learning test set. In addition, there was a great improvement to the in-class accuracy of Chopin and Beethoven. I am confident in saying that when considering the in-class accuracies and the total accuracy, that the classifier performed significantly better on the deep learning test set than the Markov model test set.

When it comes to the limitations of this experiment, there are two main limitations that I would attribute as the main limiting constraints. The first being the quality of training data. My main priority when it came to gathering training data was to gather a well-sized dataset of midi files for each composer. Since I was limited to free and public repositories, the quality of these midi files was not my concern. When training the generative models, some of these midi files were unable to be processed in its entirety by music21 due to their quality. Some that were able to be processed also had issues when it came to feature extraction (Which resulted in the dropping of some features as mentioned earlier in the feature extraction portion of the methodology). If given

access to a better source of midi files, it is possible that the generative models would have produced music of a higher quality.

The second limitation would be the time constraint when it came to building the classifier models. As mentioned earlier, in order to optimize the classifiers to a reasonable degree, I performed grid search in order to fine-tune each classifiers hyperparameters. However, I limited the scope of which hyperparameters I fine-tuned in order to make sure that the grid search would finish in a reasonable amount of time. Each new hyperparameter I wanted to try out would increase the computational time exponentially since grid search tries each and every possible combination of hyperparameters specified. If given enough time, I could perform a vastly larger grid search, which would possibly improve the performance of the classifiers.

Conclusion:

The best performing models based on the classification results on the Markov test set were 33% accurate. The best performing model based on the classification results on the deep learning test set was 38% accurate. Mostly every classification model tried in this experiment saw an improvement in accuracy for the deep learning test set. The in-class accuracies for Chopin and Beethoven were significantly better in the deep learning test set than in the Markov test set, albeit with Bach having a slightly lower in-class accuracy than in the Markov set. This supports my prediction, that the music generated by the deep learning test would have a higher classification rate. While the Markov test had more general confusion, it had significantly better in-class accuracy results. The deep learning test gave more specific and targeted confusion, which led to a higher total accuracy rate. These results differ from the result of Kaysers

experiment, in which he saw drastically higher accuracy rates. I would attribute the discrepancy between our results mainly to the difference in quality and quantity of training data. If given higher quality training data, I would expect to garner similar results. In addition, Kaysers experiments involved a different and greater set of composers than I used, which could also explain the discrepancy in our results. A more extensive and higher quality data search would have also allowed me to expand on the number of composers used for this experiment.

However, I am personally very satisfied with the results of the generative models. The music generated by the deep learning model is aesthetically pleasing to me, and some specific songs are of very high quality in my opinion. The songs generated by the Markov model sound more rigid and less pleasing to the ear than the music of the deep learning model. While one of the main goals of this experiment was to show quantitatively the superiority of deep learning when it comes to music generation, the other main goal was to simply create good music. I believe that the results of my experiment are sufficient enough and make a solid case for the argument set by my first goal. While whether or not the second goal was reached may vary depending on who you ask, but I am of the opinion that this experiment was a fun, innovative, and interesting experience that ultimately resulted in the algorithmic creation of well-made music.

Appendix:

Feature List:

1. MostCommonMelodicIntervalFeature
2. DistanceBetweenMostCommonMelodicIntervalsFeature
3. AmountOfArpeggiationFeature
4. RepeatedNotesFeature
5. ChromaticMotionFeature
6. StepwiseMotionFeature
7. MelodicThirdsFeature
8. MelodicFifthsFeature

9. MelodicTritonesFeature
10. MelodicOctavesFeature
11. DurationOfMelodicArcsFeature
12. SizeOfMelodicArcsFeature
13. NoteDensityFeature
14. AverageNoteDurationFeature
15. VariabilityOfNoteDurationFeature
16. MaximumNoteDurationFeature
17. MinimumNoteDurationFeature
18. StaccatoIncidenceFeature
19. AverageTimeBetweenAttacksFeature
20. VariabilityOfTimeBetweenAttacksFeature
21. InitialTempoFeature
22. DurationFeature
23. MostCommonPitchPrevalenceFeature
24. MostCommonPitchClassPrevalenceFeature
25. RelativeStrengthOfTopPitchesFeature
26. RelativeStrengthOfTopPitchClassesFeature
27. IntervalBetweenStrongestPitchesFeature
28. IntervalBetweenStrongestPitchClassesFeature
29. NumberOfCommonPitchesFeature
30. PitchVarietyFeature
31. PitchClassVarietyFeature
32. RangeFeature
33. MostCommonPitchFeature
34. PrimaryRegisterFeature
35. ImportanceOfBassRegisterFeature
36. ImportanceOfMiddleRegisterFeature
37. ImportanceOfHighRegisterFeature
38. MostCommonPitchClassFeature

Music Repositories + Code Repository:

http://www.midiworld.com/

http://www.kunstderfuge.com/

http://www.jsbach.net/midi/

https://www.classicalarchives.com

http://www.piano-midi.de

https://github.com/CruzJeff/Honors_Thesis

References:

- Mike Kayser, “Generative Models of Music”, 2013, http://cs229.stanford.edu/proj2013/Kayser-GenerativeModelsOfMusic.pdf

- Li-Chia Yang, et al, “MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation”, arXiv:1703.10847 [cs.SD], Mar. 2017.

- music21: a Toolkit for Computer-Aided Musicology, MIT, https://web.mit.edu/music21/.

- Johnson, Daniel. “Composing Music With Recurrent Neural Networks.” Hexahedria, Daniel Johnson, 3 Aug. 2015, www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/.

- Theano 1.0.0 Documentation, University of Montreal, 3 Aug. 2018, http://deeplearning.net/software/theano/.

- XGBoost Documentation, The XGBoost Contributors, https://xgboost.readthedocs.io/.

- LightGBM 2.2.4 Documentation, Microsoft, 3 Aug. 2018, https://lightgbm.readthedocs.io

- Scikit-Learn 0.16.1 Documentation, 3 Aug. 2018, https://scikit-learn.org/.