Composer Classification Using Symbolic Music Data: Progress Report 2

James Diotte
CUNY Graduate Center
(Dated: April 20, 2017)

Abstract: The purpose of this project is to extract features found in symbolic music data and train a classification model to discern between different classical composers, in this case Mozart and Handel. We used the built in feature extraction module found in the Python package music21 to extract our feature vectors. We chose an initial 177 features based on the work of several other researchers and added in an additional 7 features and dropped one of the original. With the extracted features we tested several different pre-processing and classification pipelines and found the best performance was achieved by a gradient boosting tree classifier which had an average accuracy score in the mid 90 percent range.

I. INTRODUCTION

Using machine learning to distinguish between the works of two composers is related to the broader task of automated music genre recognition. The task is to look for inherent characteristics in the music which allow us to make a classification. Music is typically stored either directly as an audio file or indirectly in a symbolic format such as MIDI file which contains instructions for a machine to reproduce the song (essentially like digital sheet music), The latter of which we will use in this project. Handel and Mozart were chosen for this project as both were very prominent composers during the Baroque and Classical period (resp.) of Western Music and as such there music is on some level similar, yet different enough. The challenge to distinguish between the two is made more difficult by the fact that they both wrote pieces for a large variety of different instruments and settings, e.g. piano, viola, violin music etc as well as symphonies, operas, concertos etc. Also their music spans a large period of time in their individual lives over which their style could change. We will use the music21 Python package [1] in this project to process and extract 183 features based on pitch, rhythm and chord characteristics from 526 and 574 works by Handel and Mozart respectively.

II. LITERATURE REVIEW

The features we chose are based on the work of Cory McKay in his doctoral dissertation on automatic music classification [2], furthermore McKay had previously implemented a Java package jSymbolic to process MIDI files and extract 111 of the features he describes in his dissertation [3]. The music21 developers have recreated 57 of the jSymbolic features as well as designed a few of their own. We reflected the work done by Cuthbert, Ariza and Friedland [4] in choosing our features our initial set of features which ended up totalling 177.

III. THE DATA

A. Extraction

The raw data used was 526 and 574 MIDI files of music from Handel and Mozart (resp.), the music selection spans a wide range of the works of both composers. We use the music21 DataSet object to process the data into feature vectors. The DataSet object has two containers one for parsed MIDI files and another for a set of feature extractor methods, once the MIDI files and the feature extractors are loaded into a DataSet object there is a process method which applies all the feature extractors to all the parsed MIDI files and then gives the option to save the results as a csv file for easier loading and processing. See [4] for more details.

B. The Features

The features extracted from the MIDI files are all high level statistics coming from the pitch and rhythm information. Some of the features are themselves are vectors. The largest vector feature is the Pitch Histogram which is an element of $\mathbb{R}^{128}$, each component of which is the relative frequency of the particular note taking into account its octave. The 128-dimensional Pitch Histogram Feature is related to the Fifths Pitch Histogram which is a transformation of the full histogram of notes onto a circle of fifths, the result of which is a vector in $\mathbb{R}^{12}$. These two features alone account for 140 of the 177 initial features, another 12-dimensional feature is the Pitch Class Distribution. On top of the 177 features used in [4] we added in 7 new features relating to prevalence of certain chord shapes and arpeggiation. The remaining 31 1-dimensional features will be listed here:

- Initial Time Signature 0
- Initial Time Signature 1
- Compound or Simple Meter
- Triple Meter
- Quintuple Meter

- Changes of Meter
- Most Common Pitch Prevalence
- Most Common Pitch Class Prevalence
- Relative Strength of Top Pitches
- Relative Strength of Top Pitch Classes
- Interval Between Strongest Pitches
- Interval Between Strongest Pitch Classes
- Number of Common Pitches
- Pitch Variety
- Pitch Class Variety
- Range
- Most Common Pitch
- Primary Register
- Importance of Bass Register
- Importance of Middle Register
- Importance of High Register
- Most Common Pitch Class
- Unique Note Quarter Lengths
- Most Common Note Quarter Length (Dropped)
- Most Common Note Quarter Length Prevalence
- Amount of Arpeggiation (new)
- Repeated Notes (new)
- Chromatic Motion (new)
- Melodic Thirds (new)
- Melodic fifths (new)
- Melodic Tritones (new)
- Melodic Octaves (new)

All of the features except the last nine, the three related to note quarter lengths and the new features are music21 implemented jSymbolic features based on McKay [2], the last nine are native features to music21. Most common note quarter length as implemented by music21 results in a categorical variable as such it was dropped.

IV. RESULTS

A. Experimental Design

The approach used here was to select a given *pipeline*, by which we mean a flow of operations from raw data to pre-processing and transformation to classification, and to test its performance using an independent training and test set with test set size in a ratio of 85% train to 15% test. Standard 10-fold cross validation showed high variability which is hypothesized to be a result of the varied nature of the works selected by each composer, to compensate Monte Carlo cross validation was used in place of k-fold. The idea behind Monte Carlo CV is to use N random samplings without replacement of fixed sized test subsets of the data-set to try and predict the average error rate over every possible training-test split. In our experiment we used 1000 random samples and recorded a pipeline's average classification error, standard deviation, and maximum and minimum errors. In terms of the pipeline, all pre-processing and data transformation routines were first fit on the training data only and then applied independently to the test data. All experimentation was carried out in Python using the Scikit-Learn package, and for the sake of reproducibility any randomized algorithm was executed with a fixed random state seed.

B. Best Pipelines

The best overall results came from a fine tuned gradient boosted tree classifier (GBC). The GBC model [5] is a non-linear model and like other decision tree algorithms works on splits in data independent from one another thus it was not necessary to use any pre-processing transforms. Other non-linear classifiers used were ridge classification (Ridge), k nearest neighbors (KNN) and a multi-layer perceptron (MLP). The best pipeline using a linear decision rule and one of the best overall was a pre-processing and transformations routine of standardization followed by kernel PCA with a degree five polynomial kernel projected onto the first 750 dimensions followed by a linear support vector classifier (SVC). The other linear classifier tested was logistic regression (LogReg). Standardization is a routine shifting and scaling of the training data to make the training distribution of each predictor have zero mean and unit variance. Kernel PCA is an application of the kernel trick where data is implicitly embedded into a much larger dimensional reproducing kernel Hilbert space in which the inner product can be calculated indirectly from the choice of kernel, the kernel trick can amplify small differences between classes as it may induce previously unseen separability. As PCA algorithms ultimately only depends on distances the kernel trick allows us to carry out the analysis of our embedded data in the larger space implicitly. For all of the classifiers tested, barring the GBC, 3 differ-

ent pipelines were tested: no pre-processing, standardization only (p1), standardization followed by kernel PCA (p2). Other dimensionality reduction and pre-processing strategies examined which showed significantly less performance were: standard PCA projecting on to dimension 10, 25, 100, and full 183; standardization without scaling to unit variance; radial basis and degree 2 through 6 polynomial kernels for kernel PCA; kernel PCA projected into dimensions 25 through 2000 in increments of 25 (750 was best); using chi-squared analysis of feature distributions and selecting only those with a significant difference; Wilcoxon signed rank test for feature significance; and selecting only those features with high enough Pearson correlation to class. It was surprising that even though some of the features were highly correlated with one another, or almost non-correlated with class any dimensionality reduction technique used before other transforms resulted in a loss of performance. All parameters of classifiers and transforms were fine tuned using a grid search.

C. Accuracy Metrics and Discussion

As this was a binary classification task the accuracy metric used was simply the accuracy score: # of correct predictions / # total predictions. The accuracy score was recorded over 1000 different train/test splits and the average, best and worst case was recorded as well as the standard deviation (FIG 1). Using the Monte Carlo CV method we expect our accuracy scores to be close to normally distributed which we do see (FIG 3 & 4). The average performance of the GBC model was 92.19 % compared to 90.51 % for the SVC pipeline, in fact the GBC model had the highest average, best and worst case performance and the lowest variance of all the pipelines tested, and moreover these differences were statistically significant (via a two sided t-test). In FIG 2 the estimated accuracy score probability density distributions of the top 5 pipelines (with distinct classifier) have been plotted via kernel density estimation, here the x-axis represents an accuracy score for train/test split and the y-axis represents the probability density of that particular score. The idea behind kernel density estimation is that from the Monte Carlo CV step we get a sample of 1000 accuracy scores $(a_1, \ldots, a_{1000})$ for a particular pipeline which we can assume are i.i.d., furthermore we assume the probability distribution from which they came is continuous and that our 1000 samples should be distributed closely to it, from here we create the estimated probability density function using a Gaussian kernel as follows: $\hat{p}(x) = \frac{1}{1000h} \sum_{i=1}^{1000} \phi(\frac{x-a_i}{h})$, where $\phi(x)$ is the standard Gaussian density function and h is the bandwidth parameter estimated from the data (similar to bin size in a histogram), thus the plots in FIG 2 (as well as 3 & 4) are the graphs of our so obtained $\hat{p}$'s. The statistical significance of the performance differences can be visualized from these estimated probability distributions, clearly the GBC and Logistic Regression p2 distributions are different whereas the MLP p2, SVC p2 and Ridge p2 distributions are more or less the same. Also shown are the confusion matrices for the best and worst cases for the GBC and SVC p2 pipelines. It's interesting to see that the GBC model both have opposite error types in the worst case, the GBC has more false Handel predictions versus the SVC model with more false Mozart.

Performance	Best Score	Worst Score	Average Score	Standard Dev.
GBC	98.18%	85.45%	92.19%	1.95%
MLP p2	96.36%	83.64%	90.78%	2.14%
SVC p2	98.18%	82.42%	90.51%	2.28%
Ridge p2	96.97%	83.03%	90.4%	2.2%
MLP p1	96.97%	83.64%	90.16%	2.27%
LogReg p1	94.55%	76.97%	85.08%	2.66%
SVC p1	93.33%	75.15%	84.97%	2.73%
Ridge	92.73%	76.97%	84.75%	2.6%
KNN p1	92.73%	76.36%	84.71%	2.68%
Ridge p1	93.33%	76.97%	84.38%	2.68%
SVC	92.12%	75.76%	83.86%	2.67%
LogReg	92.73%	75.76%	83.32%	2.68%
KNN p2	90.91%	72.73%	83.07%	2.8%
MLP	92.73%	64.24%	82.44%	4.06%
KNN	90.3%	72.73%	81.78%	2.7%
LogReg p2	96.36%	49.7%	81.29%	8.96%

FIG. 1. Pipeline Performances over Monte Carlo CV

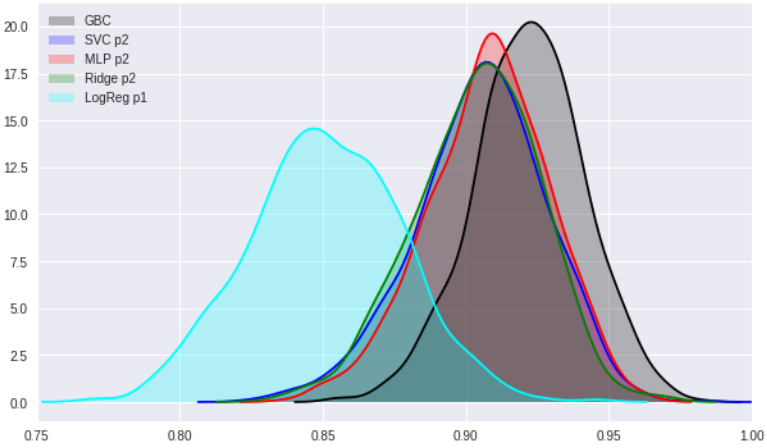

FIG. 2. Best Pipeline Estimated Score Distributions

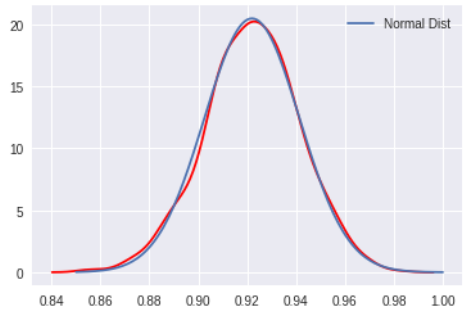

FIG. 3. Normal vs Estimated GBC Distributions

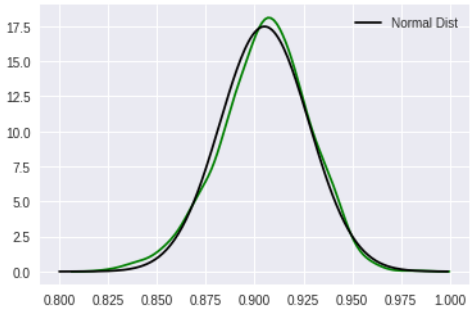

FIG. 4. Normal vs Estimated SVC Distributions

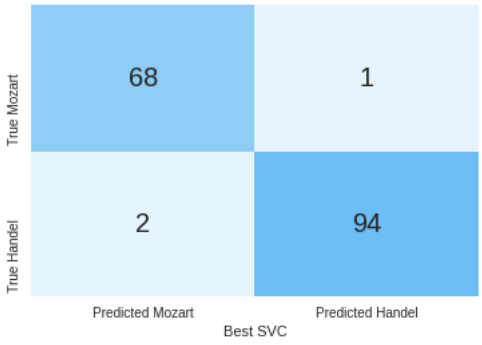

FIG. 5. Best SVC Confusion Matrix

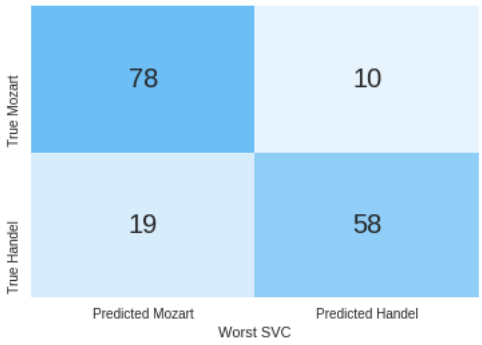

FIG. 6. Worst SVC Confusion Matrix

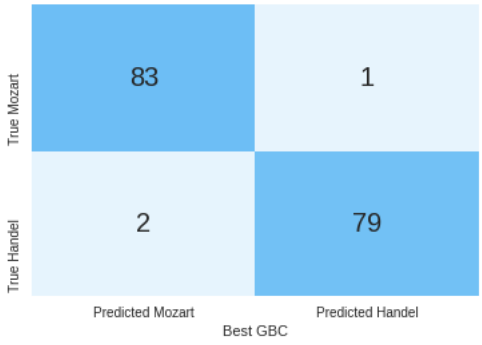

FIG. 7. Best GBC Confusion Matrix

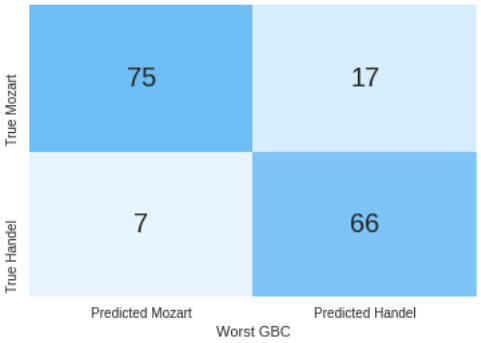

FIG. 8. Worst GBC Confusion Matrix

V. FUTURE WORK

This particular study focused only on two composers and was thus limited in scope, in the future it would be interesting to see how these techniques can be extended to distinguish between many different composers. It would also be interesting to investigate if these methods can be used for more modern music genre classification. The features used for this project were almost all global statistical features, while these showed good results in this case it would be worthwhile to also include more localized features as well.

[1] M. Cuthbert and C. Ariza: "music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data," *Proceedings of the International Symposium on Music Information Retrieval*, pp. 63742, 2010.

[2] C. McKay:"Automatic Music Classification with jMIR," Ph.D. Dissertation, McGill University, 2010.

[3] C. McKay and I. Fujinaga:"jSymbolic: A feature extractor for MIDI files" *Proceedings of the International Computer Music Conference*, pp. 3025, 2006.

[4] M. Cuthbert, C. Ariza and L. Friedland: "Feature Extraction and Machine Learning on Symbolic Music Using the music21 Toolkit," *12th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011

[5] Jerome H. Friedman: "Greedy Function Approximation: A Gradient Boosting Machine," *The Annals of Statistics*, Vol. 29, No. 5 (Oct., 2001), pp. 1189-1232