Analysis and Classification of Symbolic Western Classical Music by Composer

Saahil Jain [1] **Akshay Smit** [1] **Tim Yngesjö** [2]

Abstract

Our aim was to experiment with various methods of classifying pieces of Western classical music by composer and better understand the most distinctive features of these composers. We chose to use MIDI encodings of musical scores, which contain no sound. When considering 15 different composers spanning distinct stylistic eras, decision tree boosting achieved the highest accuracy among all the models (79% test accuracy). Vertical intervals and texture proved considerably more useful for classification than rhythmic features. Additionally, we converted musical pieces into an image representation that was fed into a deep CNN, effectively creating a vision problem. Surprisingly, this achieved 70% test accuracy among 6 composers.

1. Introduction

Western classical music is accompanied by rigorous musical notation and a compendium of theory. Nevertheless, even experts find it difficult to differentiate between similar classical composers given a musical piece. For example, due to the stylistic similarities between the contemporaries Mozart and Haydn, humans achieved only 57% accuracy in differentiating these two on a study with over 40,000 instances of participation, while the majority classifier would achieve 72% (Yust et al., 2013). Musical theory provides a large number of intuitive measures of various properties of a musical piece. For example, we can measure the prevalence of intervals between two or more simultaneously played notes, known as "vertical intervals," in a particular piece. We can thus convert each musical piece into a vector of numbers, where each number corresponds to a specific feature informed by music theory, and feed this vector into our classifiers.

[1]Department of Computer Science, School of Engineering, Stanford University, Stanford, USA [2]Department of Electrical Engineering, School of Engineering, Stanford University, Stanford, USA. Correspondence to: Akshay Smit <akshaysm@stanford.edu>.

The inputs to our classifiers were 1,494-dimensional numerical feature vectors, and the output was a class prediction. For this case, the classifiers we consider are SVMs (using a one vs. one scheme), softmax regression, linear discriminant analysis, k-nearest neighbors, gradient boosting using decision stumps, and a fully-connected feed-forward neural network. For the SVM classifier, we also performed dimensionality reduction using PCA to check the accuracy with varying number of principal components. We also used PCA for visualizing our high-dimensional dataset.

An alternative is to convert each MIDI file into a visual score (a 64×1536 image), and then feed these images into a deep CNN with a softmax output layer. MIDI files contain sequences of notes for each voice in the piece, and we can extract these notes and plot them, thus creating a graph.

We are very grateful to the Classical Archives (Schwob) for donating their large MIDI database, comprising over 10,000 MIDI files, for this project.

2. Related Work

Much attention has been devoted to music genre classification (Basili et al., 2004), which is relatively easy for humans, who can achieve 70% accuracy (Dong, 2018). Our problem of composer classification is significantly more difficult. Even human experts struggle to classify similar composers, achieving 48% accuracy on 4 composers (Abeler, 2015) and 57% on 2 composers (Yust et al., 2013).

Work within composer classification itself has focused on a small number of composers and smaller datasets. For example, Pollastri et al. use hidden markov models to classify 5 composers using 121 themes per composer (Pollastri & Simoncelli, 2001). We classify 15 composers spanning over 5,000 musical pieces. For classification via computer vision, CNNs have been widely used to classify music into different genres (e.g., rock, pop) (Choi et al., 2017).

3. Data Processing

3.1. Generation of Numerical Features

We looked at a compendium of 15 composers representing the baroque, classical and romantic eras of classical music. The composers were Bach, Handel, Vivaldi, Telemann

(baroque), Mozart, Haydn (classical), Brahms, Chopin, Debussy, Mandelssohn, Liszt, and Dvorak (romantic). In addition, there were three "transitional" composers spanning two eras: Beethoven, Schubert, and Hummel. Using a powerful open-source tool jSymbolic (jsy), we converted each MIDI file into a 1,494-dimensional numerical feature vector, where each feature encoded a numerical property of the music. The resulting features represented the rhythm, melody, texture and tempo of the piece. The feature vectors were normalized so that each dimension had zero mean and unit variance.

We dropped the "Note Density per Quarter Note Variability" feature provided by jSymbolic due to the presence of a large number of NaNs. After the feature extraction was complete, we removed duplicate files using the names of the compositions included in a catalog provided by Classical Archives. This left us with 5,291 examples from which we created a 60-20-20 train-test-dev split.

3.2. Visualizing Composers using PCA

We performed dimensionality reduction on the 1,494-dimensional feature vectors using PCA, and plotted each composer against the other using the first two principal components. Composers from distinct eras (especially romantic vs. baroque) appeared far more separable than composers from the same era. Figure 1 shows Bach (baroque) vs. Beethoven (classical/romantic). In contrast, composers from similar eras like Haydn and Mozart could not be meaningfully separated at all using 2 principal components.

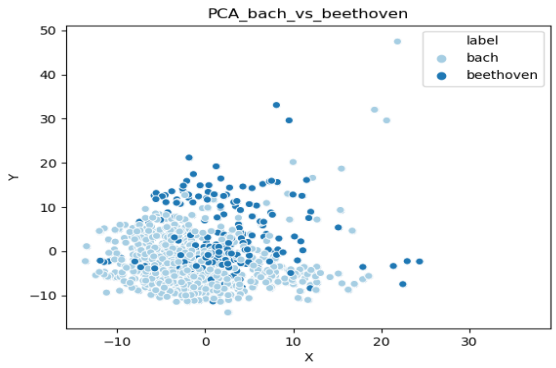

Figure 1. First two principal components, Bach vs. Beethoven

3.3. Image Generation

For the experiment, when using convolutional neural networks (CNNs) each MIDI file was transformed into an image (M) of size 64x1536 pixels: $M \in \mathbb{Z}_{0,255}^{64\times1536}$ where $\mathbb{Z}_{a,b}$ is the set of integers between a and b including a and b. The vertical axis of the image corresponds to 64 distinct MIDI notes (notes 32-96) because most of the notes in our pieces fall within this range. The horizontal axis is time. The first 153.6 s of the MIDI file was sampled with a frequency of 10 Hz to obtain a 1536 pixels wide image. Each pixel of the single-channel image M is given by

$$M_{i,j} = \begin{cases} \text{velocity of note } i \text{ at time } j/F_s & \text{if note } i \text{ is on} \\ 0 & \text{else} \end{cases}$$

where $F_s = 10$ Hz is the sample frequency. A MIDI file specifies the velocity of each note, where velocity is a measure of the force with which the note is played.

Figure 2. Small segment of one of the scores generated from a MIDI file

Augmentation of the images was used to make the training set larger. This was done by applying a random translation transformation by (dx, dy) to the image where $dx \in \mathbb{Z}_{-150,150}$ and $dy \in \mathbb{Z}_{-10,10}$. A 100 pixel wide section in a random position near the middle of the image was also set to be black ($M_{i,j} = 0$ if $|j - z| < 50$ where $z \in \mathbb{Z}_{718,818}$).

4. Experiments

4.1. Classification of 15 composers

In the most general setting, we considered all 15 of our composers, the results for which can be seen in Table 1. Our primary metrics were accuracy, precision, recall, and F1-score. The latter three are computed per-class, but we provide averages of these over all classes. Accuracy is computed across all classes (simply the number of pieces classified correctly divided by the total number of pieces).

Many of our other classifiers achieved near-perfect train accuracy, but couldn't exceed 80% test accuracy. Trying to address this apparent lack of generalizability with stronger regularization, however, only served to decrease both the dev and the train accuracy. One explanation for this phenomenon is that some composers were not adequately represented in our dataset. The baroque composer Telemann was the most poorly represented with only 99 examples in the entire dataset, and softmax only achieved a class-specific accuracy of 36% for Telemann. In comparison, Bach was the most represented, comprising 25% of the entire dataset. With softmax, the class-specific accuracy for Bach was 92%.

Model	Train Acc.	Dev Acc.	Test Acc.	Avg. Precision	Avg. Recall	Avg. F1 Score
Gradient Boosting	1.000	0.794	0.791	0.830	0.706	0.742
Softmax	0.990	0.742	0.743	0.720	0.658	0.682
SVM	0.999	0.741	0.734	0.765	0.641	0.686
LDA	0.885	0.704	0.707	0.675	0.623	0.639
k-NN	0.648	0.564	0.575	0.617	0.433	0.463
NN	0.989	0.752	0.752	0.720	0.684	0.698

Table 1. Metrics for classification of all 15 composers.

4.1.1. SVM + PCA

Multi-class support in scikit-learn's implementation of SVM is enabled by a one-vs-one scheme. First, we used all the 1,494 features. While we initially tried using a linear kernel, we were able to improve the dev accuracy by a few percentage points using an rbf kernel. We used L^2 regularization, with the amount of regularization controlled by the C parameter. We achieved the highest dev accuracy at $C = 5.5$ using a grid search. The γ parameter in the rbf kernel was set to $\gamma = 1/1494$ since the number of features is $1,494$. The classification accuracy of SVM was very similar to that of softmax regression.

We also attempted to use SVM's on data transformed with PCA. The results are in Table 2, and the final column indicates no PCA. Surprisingly, the train data is linearly separable even with 10 principal components (but with poor generalizability), which suggests that we may be able to get reasonable accuracy in limited-feature settings. We consider this in Section 4.3 where we restrict ourselves to a far smaller number of features grouped into rhythm, vertical intervals, melodic intervals, and texture.

# Components	2	10	50	100	500	1494
Train	0.486	1.000	1.000	1.000	1.000	1.000
Dev	0.304	0.421	0.522	0.570	0.681	0.741

Table 2. Accuracies for SVM with varying number of principal components

4.1.2. Gradient Boosting

We used gradient boosting with decision stumps using the powerful LightGBM library (lig). This uses the following initial state and update rule:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, \gamma\right)$$

$$F_m(x) = F_{m-1}(x) +$$

$$\arg\min_{h_m \in \mathcal{H}} \left[\sum_{i=1}^{n} L\left(y_i, F_{m-1}\left(x_i\right) + h_m\left(x_i\right)\right) \right]$$

where $h_m \in \mathcal{H}$ is the newly added decision stump and L is the cross-entropy loss function. F_{m-1} is a linear combination of decision stumps obtained in previous iterations.

This was by far our best classifier, achieving over 79% test accuracy. We used 700 estimators with an L^2-penalty, and the parameter λ_{L^2} controlling the amount of regularization was set to $\lambda_{L^2} = 10.0$ ($\lambda_{L^2} = 0$ means no regularization). We used sample bagging with a fraction of 0.6, so that at each iteration, only 60% of the train data would be considered. We found that this greatly increased the accuracy. Attempts at using greater L^2/L^1 regularization, and limiting the depth of the tree only decreased the accuracy.

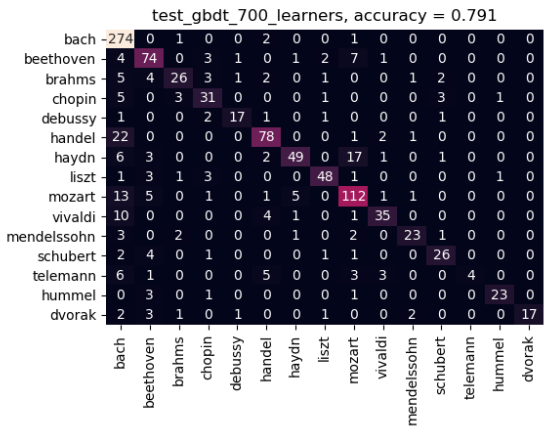

Figure 3. Confusion matrix on test data for LightGBM, all 15 composers

From Figure 3 we see that the classifier seems to often confuse composers from the same era, such as Handel and Bach (both baroque), and Haydn and Mozart (both classical). The baroque composer Telemann fares particularly poorly, with most Telemann pieces being wrongly attributed to other baroque composers like Bach, Handel and Vivaldi.

LightGBM also tracks feature importance. Whenever a split is performed on a feature, the resulting information gain is added to a running sum. This allows us to plot the top 50 features by information gain in Figure 4.

From Figure 4 we see that the most important features correspond to melodic or vertical intervals rather than rhythm.

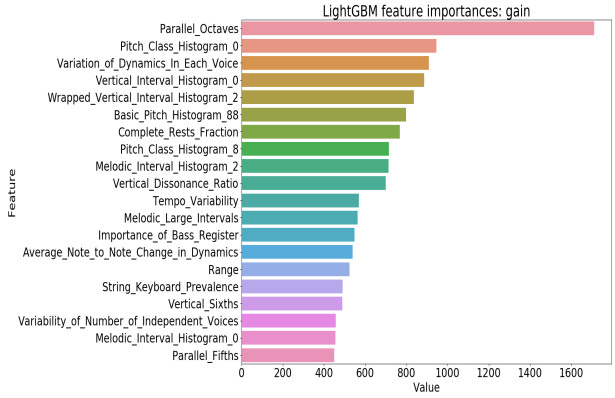

Figure 4. LightGBM feature importances by information gain

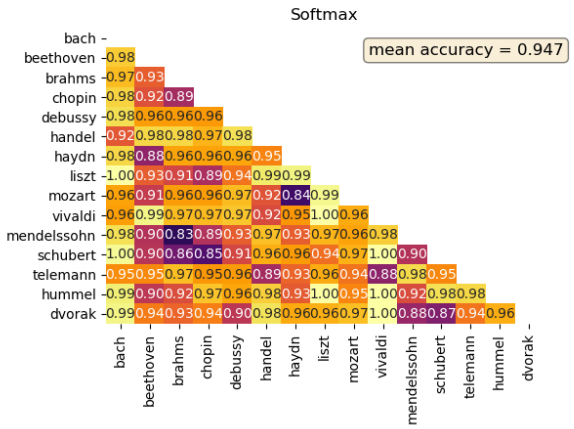

Figure 5. Binary classification on 15 composers using logistic regression

If the notes in an interval are played simultaneously, the interval is said to be vertical. If the notes are played one after another, the interval is melodic. In particular, dissonant intervals appear to be very important among the top features. jSymbolic measures the prevalence of every type of interval (both melodic and vertical) in a piece and creates a histogram. For a complete description of all jSymbolic features, see the jSymbolic feature explanations. The feature "Melodic Interval Histogram 2" measures the prevalence of melodic intervals of 2 semitones, which is very dissonant. The "Wrapped Vertical Interval Histogram 2" is similar, but considers vertical intervals instead of melodic intervals. The "Vertical Dissonance Ratio" is the ratio of the number of dissonant vertical intervals (seconds, sevenths, tritones) to the number of consonant vertical intervals. Such dissonant intervals were not prevalent during the baroque period due to clerical interference, but romantic composers like Liszt deliberately made heavy use of dissonant intervals to create feelings of tension. We will revisit this theme in Section 4.2 on binary classification.

4.1.3. Neural Nets

We used a fully connected neural net with a softmax output layer and ReLU activation functions, implemented in PyTorch (Paszke et al., 2019). We experimented with the architecture of the net, trying different numbers of hidden layers and varying the number of hidden units. We used a small amount of L2-regularization ($\lambda = 0.001$). We tried using dropout, but this failed to increase the dev accuracy. It turned out that a simple architecture with only one hidden layer (with 150 neurons) performed the best. The net was trained using SGD with batch size 50, momentum parameter 0.9, and learning rate 0.04. This learning rate allowed for quick training without causing divergence. The neural net achieved lower accuracy compared to gradient boosting, likely because our data was structured (numerical features usually denoting frequencies).

4.2. Binary Classification and Feature Analysis

With 1,494 numerical features, we have more than enough power to perform binary classification on each pair of composers and achieve a mean accuracy of 94.7% using logistic regression. The results of this are in Figure 5. The lower scores tend to correspond to composers from the same era, like Mendelssohn vs. Brahms. Beethoven, whose style changed significantly during his life, is confused for both Haydn (classical) and Mendelssohn (romantic). By looking at the magnitude and sign of the coefficients, we can determine the most important features in the logistic regression model.

When comparing romantic vs. baroque composers, instrumentation is an immediate giveaway. The top features for romantic composers include prevalence of piano, whereas for baroque composers, the top features include prevalence of harpsichord. Indeed, the harpsichord was predominant during the baroque period, and the piano only gained popularity after the baroque period. Instrumentation also immediately differentiates composers who predominantly wrote for different instruments, such as Vivaldi (string orchestra) vs. Chopin (piano). Mozart, who wrote extensively for French Horn, often had prevalence of French Horn among the top features.

Features corresponding to pitch variability and melodic/vertical intervals dominated the top features for most binary classification problems. Rhythmic features were barely represented. The most indicative features for romantic composers included a high prevalence of dissonant/non-standard melodic and vertical intervals as well as greater variance in pitch and dynamics. Historically, this expansion of dynamic range was due to larger orchestras as well as prevalence of the piano during the romantic era.

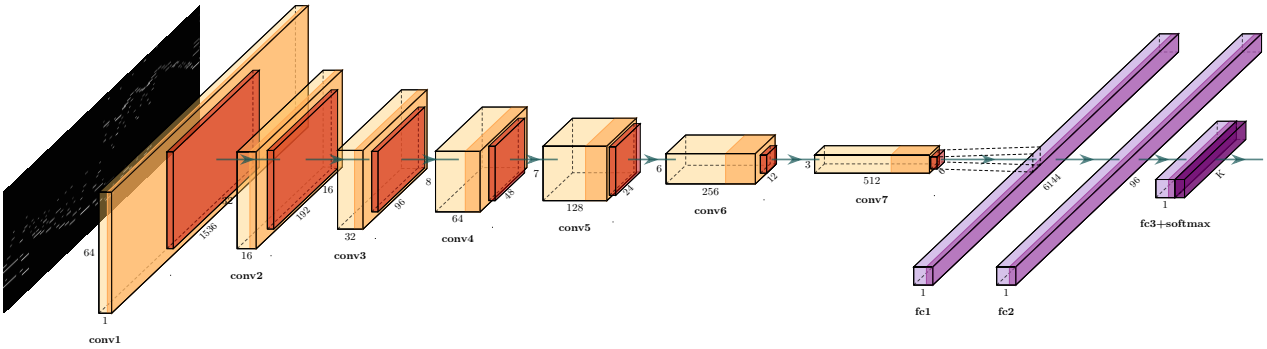

Figure 6. Architecture of the CNN

4.3. Classification by Era with Limited Features

We grouped composers by era (baroque, classical, and romantic), dropping Beethoven, Schubert and Hummel as their musical styles spanned two eras. We also created (disjoint) groups of features corresponding to rhythm (255 features), melodic intervals (146), vertical intervals (46), and texture (24). We did classification of era with LightGBM. Surprisingly, rhythm had the lowest overall accuracy at 73.2% whereas a mere 24 texture-related features gave an overall accuracy of 78.2%. Surprisingly, the highest overall accuracy of 85.3% was achieved by the vertical features. These results again suggest that rhythm is less distinctive than melodic or vertical intervals.

4.4. Image Classification with CNN

A separate experiment was conducted where we classified 6 different composers by feeding images of the MIDI files into a convolutional neural network. The architecture of the CNN can be seen in Figure 6. It consists of 7 convolutional layers with max-pooling and 3 fully connected layers. The accuracy of the classifier on the test set was 70%. We did some experimentation by varying the amount of augmentation that was used. The classifer only obtained an accuracy of 55% without augmentation compared to 70% when the size of the training data had been increased by a factor of 15. The effect of varying the amount of augmentation can be seen in Figure 7. Augmenting the data by a factor greater than 10 does not further improve the accuracy.

The accuracy of the CNN classifier is remarkably good considering the fact that much information is left out when the MIDI-files are converted into images, such as tempo, time signature, and dynamics.

5. Conclusion and Future Work

On the 15 composer classification experiment, gradient boosting with decision stumps gave the highest accuracy at

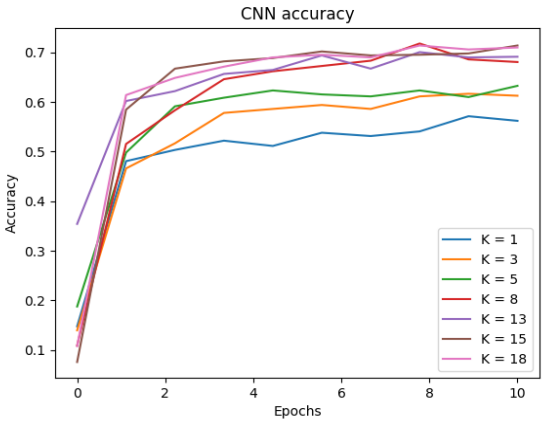

Figure 7. Accuracy during training with different amount of training data. The training data is augmented by a factor of K.

79.1%, outperforming the simple fully-connected NN. Composers from similar eras were considerably harder to differentiate than composers from clearly distinct eras, which was also observed in the binary classification experiment with logistic regression. Melodic and vertical intervals were found to be much more distinctive than rhythm, and instrumentation was an immediate giveaway when comparing composers who wrote for different instruments. Classification of images using the CNN gave an accuracy of 70% on 6 composers which, while surpassed by the numerical feature vector algorithms, is surprising due to the significant amount of information that is lost in converting from a MIDI file to the score representation.

Future work could involve further exploring image classification by extracting more information from our MIDI files, essentially creating multichannel images instead of grayscale images. We could also try generating 1 dimensional vectors from the notes to preserve the tracks instead of creating images. Another approach might be to use CRNN's to generate the sorts of scores that we fed into our CNN.

References

jsymbolic. URL http://jmir.sourceforge.net/jSymbolic.html.

Lightgbm. URL https://lightgbm.readthedocs.io/en/latest/.

Abeler, N. Musical composer identification mus-15. 2015.

Basili, R., Serafini, A., and Stellato, A. Classification of musical genre: a machine learning approach. In *ISMIR*, 2004.

Choi, K., Fazekas, G., Sandler, M., and Cho, K. Convolutional recurrent neural networks for music classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2392–2396, March 2017. doi: 10.1109/ICASSP.2017.7952585.

Dong, M. Convolutional neural network achieves human-level accuracy in music genre classification, 2018.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., dAlché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Pollastri, E. and Simoncelli, G. Classification of melodies by composer with hidden markov models. In *Proceedings First International Conference on WEB Delivering of Music. WEDELMUSIC 2001*, pp. 88–95, Nov 2001. doi: 10.1109/WDM.2001.990162.

Schwob, P. R. Classical archives midi files.

Yust, J., Wild, J., and Burgoyne, J. *Mathematics and Computation in Music: 4th International Conference, MCM 2013, Montreal, Canada, June 12-14, 2013, Proceedings*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013. ISBN 9783642393570. URL https://books.google.com/books?id=XZW6BQAAQBAJ.

6. Contribution

All authors contributed to the discussion and direction of the project.

Tim worked on separating composer MIDI files, performing PCA and producing the corresponding plots, the binary classification experiment, the fully-connected NN and the CNN. He also generated all the images and did data augmentation for the CNN, and was the joint poster lead.

Akshay ran the experiments for all 15 composers (SVM, gradient boosting, LDA, softmax), the era experiment, and did the analysis of feature importances. He also did a lot of the data cleaning and helped extract feature vectors from the MIDI files using jSymbolic. Also the report lead.

Saahil ran the experiments for all 15 composers for the milestone models, helped with data cleaning, extraction of feature vectors from MIDI files using jSymbolic, analysis of feature importances, and was the joint poster lead.

7. Code

The code for our project can be found at the following link: Code. Note that we have not included the data donated to us for this project by the Classical Archives to respect their wishes.