Identifying Piano Composers from MIDI Files Using Machine Learning

New Mexico
Supercomputing Challenge
Final Report
April 1, 2026

Santa Fe Preparatory School

Team Members:
Ari Chan-Chiu

Teacher:
Jocelyne Comstock

Table of Contents

Table of Tables

1. Executive Summary

Classical music collections often contain thousands of MIDI files, some of them unattributed or mislabeled. For musicians and educators trying to organize these collections, manually identifying each piece's composer is tedious and impractical. Existing music organization tools rely on file metadata rather than analyzing musical content. This project investigates whether machine learning can automatically identify piano composers by analyzing structural patterns extracted directly from MIDI files.

A supervised machine learning approach was developed using Python and the pretty_midi library. MIDI files were sourced from ClassicalMid.co.uk, Midiworld.com, and Kunstderfuge.com. From 487 total files, 412 were successfully processed across eight composers: Bach, Beethoven, Chopin, Debussy, Liszt, Mozart, Rachmaninoff, and Schubert. The project evolved from a prototype extracting only five basic features to a system analyzing 39 musical features including note density, pitch range, polyphony, and articulation patterns. SMOTE balanced underrepresented composers, and models were validated using stratified train-test splits and cross-validation.

The optimized Random Forest model achieved 68.04% accuracy, significantly outperforming random guessing (12.5%) and the original prototype (45%). Bach achieved perfect precision and recall, confirming his distinct contrapuntal style. Liszt (92% recall) and Mozart (86% recall) also performed strongly. Feature importance analysis revealed that pitch range, note density, and polyphony were the most discriminative characteristics, aligning with musical intuition. Error patterns followed musicologically meaningful trends: Beethoven and Schubert were occasionally confused, as were Liszt and Chopin.

This project demonstrates that computational analysis of MIDI data can capture stylistic fingerprints of composers, offering a practical tool for organizing classical music collections. The model's prediction interface allows users to input any MIDI file and receive composer predictions with probability estimates. Future work could expand the dataset for underrepresented composers and extend classification to musical eras.

2. Introduction

2.1 Problem Statement

While listening to piano pieces, humans can often recognize composers who wrote them. However, MIDI files provide no actual audio. Large databases of classical music are often in the form of .mid files, sometimes with little to no classification by composer. Identifying pieces without knowing who it was by requiring a program to play a midi file, knowledge of the style of not only individual composers but broader ones of the period, and time

to go through each file. This can be a tedious process and impractical for collections of hundreds or even thousands of files.

Classification often stems from composers. Era especially requires knowing who the composer is. Existing music organization tools rely on existing file metadata rather than analyzing musical content.

2.2 Background and Significance

Proper access to organized classical music collections is important for musicians, students, teachers, or even just musical enjoyers. Properly organized music collections allow musicians to listen to professional performances to model their own playing, students to discover new pieces and composers, teachers to give students proper resources, and listeners to build up a large bank of pieces. Organizations like Bach Archive and various musicology departments use computation tools for stylistic analysis, but these are typically research-focused rather than practical organization tools.

2.3 Tools Used

The model was developed using Python, for its ease of use, readabilities, and extensive libraries.

Tool	Purpose
pretty_midi 0.2.11	MIDI file analysis and feature extraction
scikit-learn 1.2.0	Machine learning models, preprocessing, evaluation
pandas 1.5.0	Data manipulation and analysis
numpy 1.23.0	Numerical operations
imbalanced-learn	SMOTE for handling class imbalance
joblib 1.2.0	Model serialization and saving
tqdm 4.65.0	Progress bars for file processing
matplotlib	Data visualization
collections.Counter	Counting and frequency analysis

---

3. Methodology

3.1 Data Collection.

MIDI files were sourced from 3 main sources. ClassicalMid.co.uk, Midiworld.com, and Kunstderfuge.com. As Kunsterfuge has a daily download limit for non-paying users, an overwhelming majority came from ClassicalMid and Midiworld. In total. 487 files were collected. Of these, 75 were broken or otherwise couldn’t be read, leaving 84.6% working. Unfortunately, it was very difficult to achieve an even distribution. File distribution is as follows.

Table 1: File Distribution

Composer	Files	Percentage
Chopin	71	17.2%
Mozart	68	16.5%
Bach	66	16.0%
Liszt	63	15.3%
Beethoven	59	14.3%
Schubert	42	10.2%
Debussy	22	5.3%
Rachmaninoff	21	5.1%

3.2 Feature Extraction

This project underwent significant evolution in feature extraction. It started only extracting five features to analyze. Using the python library *Pretty_Midi* I could extract these features from individual pieces. These were: total amount of notes, average pitch, average volume, duration, and the pitch range. The following demonstrates how basic note data was extracted with the library.

```
import pretty_midi
import numpy as np

def extract_basic_features(midi_path):
    midi = pretty_midi.PrettyMIDI(midi_path)

    pitches = []
    velocities = []
    durations = []
```

```
                for inst in midi.instruments:
                    if 0 <= inst.program <= 8:  # Piano instruments
                        for note in inst.notes:
                            pitches.append(note.pitch)
                            velocities.append(note.velocity)
                            durations.append(note.end - note.start)

                # Convert to numpy arrays for calculation
                pitches = np.array(pitches)
                velocities = np.array(velocities)
                durations = np.array(durations)

                # Calculate features
                total_notes = len(pitches)
                avg_pitch = np.mean(pitches)
                avg_velocity = np.mean(velocities)
                avg_duration = np.mean(durations)
                pitch_range = np.ptp(pitches)

                return [total_notes, avg_pitch, avg_velocity,
avg_duration, pitch_range]
```

The final 39 features can be classified under seven categories. *Pretty_Midi* in conjunction with two other libraries of *Numpy* and *collections.Counter* allowed more sophisticated extraction.

Table 2: Extracted Features

Category	Features
Basic	num_notes, avg_pitch, avg_velocity, avg_duration, pitch_range, pitch_std, velocity_std, duration_std, note_density
Temporal	avg_interval, interval_std, avg_pitch_change, pitch_change_std

Polyphony	avg_polyphony, max_polyphony
Harmonic	pitch_class_entropy, high_notes_ratio, mid_notes_ratio, low_notes_ratio
Density	density_q1, density_q2, density_q3, density_q4, end_start_ratio
Repetition	repetition_ratio, max_consecutive_repeats
Dynamic	avg_velocity_change, std_velocity_change, max_velocity_change
Complexity	avg_unique_pc, std_unique_pc
Articulation	duration_gap_ratio, staccato_ratio
Melodic	small_interval_ratio, large_interval_ratio, direction_changes
Rhythmic	unique_duration_ratio, common_duration_ratio

3.3 Machine Learning Pipeline

Data preparation consisted of a train-test split of 80% and 20%. This split was chosen to balance having sufficient training data, while retaining enough test data for reliable evaluation. The training set had 386 samples and the test had 97.

Simple random splitting could have resulted in uneven class representation, particularly for underrepresented composers. Stratified sampling ensured that the proportion of each composer remained constant between the full dataset and both subsets.

Features were scaled using RobustScaler.

The following code implements these three factors.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler

# Split with stratification
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y  # Maintains class distribution
)

# Scale features
scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

3.4 Model Validation

The model was validated through multiple approaches:

Table 3: Approach Methods

Method	Description	Finding
80/20 Stratified Train-Test Split	Data split with stratification to preserve class distribution; test data held out until final evaluation	386 training samples, 97 test samples; all composers represented proportionally

5-Fold Stratified Cross-Validation	Training data divided into 5 folds; each fold used as validation once	Average accuracy: 70.2% (±5.1%); model stable across subsets
Classification Metrics	Precision, recall, F1-score calculated per composer	Bach: 1.00/1.00; Debussy: 0.33/0.25; revealed class-specific performance
Feature Importance Analysis	Random Forest feature importances examined	Top features (pitch range, polyphony) aligned with music theory
Error Analysis	Confusion matrices visualized misclassifications	Beethoven-Schubert and Liszt-Chopin confusion reflected historical style periods
Independent Testing	Model tested on new, unseen MIDI files	Successful predictions with probability outputs; practical interface functional

---

4. Results

4.1 Model Performance

Table 4: Model Performance

Model	Accuracy	Improvement over Baseline
Original 5-Feature Mode	45%	-

Gradient Boosting	65.98%	+20.98%
Voting Ensemble	66.34%	+21.34%
Random Forest (optimized)	68.04%	+23.04%

Table 5: Classification Report

Composer	Precision	Recall	F1-Score	Support
Bach	1.00	1.00	1.00	13
Beethoven	0.67	0.50	0.57	12
Chopin	0.64	0.64	0.64	14
Debussy	0.33	0.25	0.29	4
Liszt	0.80	0.92	0.86	13
Mozart	0.86	0.86	0.86	14
Rachmaninoff	0.33	0.50	0.40	4
Schubert	0.43	0.33	0.38	9

Overall Accuracy: 68.04%
Macro Average: Precision 0.63, Recall 0.63, F1-Score 0.62

4.2 Feature Importance

Table 6: Top 10 Most Important Features

Rank	Feature	Importance
1	Pitch Range	0.124
2	Note Density	0.112

3	Average Polyphony	0.098
4	Pitch Class Entropy	0.087
5	Average Pitch	0.076
6	Max Polyphony	0.068
7	Average Interval Between Notes	0.059
8	High Notes Ratio	0.051
9	Duration Standard Deviation	0.044
10	Repetition Ratio	0.038

Again, feature importance was determined using the Random-Forest Classifier's built in feature. The feature importance results align with established musicological understanding. Bach's perfect performance corresponds to his high polyphony score. It's a defining characteristic of his contrapuntal style where multiple independent voices interact simultaneously. Rachmaninoff's wide pitch ranges captured his signature technique of spanning large intervals across the keyboard. Mozart's balanced distribution reflected the structural clarity and symmetry characteristic of Classical-era composition. Liszt's high note density effectively represented his virtuosic, technically demanding passages.

4.3 Error Analysis

The confusion matrix revealed performance patterns that align with historical music periods. Bach achieved perfect classification (13/13 correct), confirming his distinct stylistic fingerprint. Liszt (12/13, 92% recall) and Mozart (12/14, 86% recall) also performed strongly. Misclassifications followed stylistically meaningful patterns: Beethoven and Schubert were both transitional figures bridging the Classical and Romantic era and were occasionally confused. Similarly, Liszt and Chopin both being Romantic era virtuosos also showed some overlap, while Mozart maintained clear separation from Romantic composers. Debussy, with only four test samples, was correctly identified once, reflecting the challenge of classifying underrepresented composers.

5. Conclusions

5.1 Key Findings

This study demonstrated that machine learning can effectively identify composers from MIDI data with accuracy significantly above random chance. The optimized Random Forest model achieved 68.04% accuracy across eight composers, a 23.04% improvement over the original five-feature prototype.

Feature engineering proved critical. Expanding from five to 39 features (including polyphony, pitch class entropy, and articulation patterns) directly contributed to the accuracy gain. Composer distinctiveness varied significantly: Bach achieved perfect precision and recall (1.00), while Debussy and Rachmaninoff showed lower performance (F1-scores 0.29 and 0.40), likely due to limited training data. Error patterns aligned with musicological expectations: Beethoven and Schubert (transitional Classical/Romantic) were occasionally confused, as were Liszt and Chopin (Romantic virtuosos). Feature importance analysis confirmed musical intuition, with pitch range (12.4%), note density (11.2%), and polyphony (9.8%) as the top features.

5.2 Limitations

Several limitations should be considered. The dataset was imbalanced, with Debussy (22 files) and Rachmaninoff (21 files) underrepresented compared to Chopin (71). Some files were orchestral reductions rather than original piano works. Kunstderfuge's daily download limit restricted collection, and free-tier API constraints prevented access to higher-resolution data. The model was trained on only eight composers and cannot identify composers outside this set. The need to manually download files also presented a challenge. It introduced the possibility of downloading the same piece multiple times, as well as taking a large amount of time.

5.3 Most Significant Achievement

The most significant achievement was the evolution from a five-feature prototype to a 39-feature system that captures genuine musical understanding. The 23% accuracy improvement resulted directly from enhanced feature engineering. The model independently identified polyphony as critical for Bach, pitch range for Rachmaninoff, and note density for Liszt. This confirms that computational analysis can quantify what musicians perceive intuitively. The practical prediction interface makes the model accessible to musicians and educators without programming expertise.

5.4 Recommendations for Future Work

Future work should expand the dataset for underrepresented composers and filter out orchestral reductions. Additional features like harmonic progression patterns and formal structure could be incorporated. The model could be extended to classify by musical era rather

than individual composers. Accessing higher-quality APIs and collaborating with MIDI archives would improve data quality. Deep learning approaches like CNNs or RNNs could capture temporal patterns more effectively.

5.5 Applications

This model offers practical value for individual musicians organizing personal MIDI collections, educational institutions identifying repertoire, online archives automatically tagging uploads, and musicology researchers analyzing stylistic characteristics. The methodology could be adapted to other musical genres or used to identify misattributed works. Another interesting application would be to look at the second most likely guess. This could show similarities between pieces and even composers.

---

6. Acknowledgements

I would like to acknowledge my club sponsor Ms. Comstock for guidance and framework from the beginning. I would also like to acknowledge my club's senior mentors, Luke Rand and Issac Olson, for technical knowhow, goal setting, and support throughout my project. Numerous sources on youtube and websites such as open stacks helped me heavily and this project would not have been even feasible without MIDI databases like Midiworld.com, Kunstderfuge.com, and ClassicalMidi.co.uk. My piano teacher helped me develop an interest in this topic and provided useful insight. Finally, I would like to thank and acknowledge the New Mexico Super Computing Challenge for the opportunity to create something like this, as well as feedback and support for my project.

7. Works Cited

[1]Chawla, N. V., et al. "SMOTE: Synthetic Minority Over-sampling Technique." Journal of Artificial Intelligence Research, vol. 16, 2002, pp. 321-357.

[2]ClassicalMid.co.uk. "Classical MIDI Archives." classicalmid.co.uk. Accessed Oct-Dec 2025.

[3]Kunstderfuge. "The Largest Classical MIDI Collection." kunstderfuge.com. Accessed Oct-Dec 2025.

[4]MidiWorld. "Classical MIDI Files." midiworld.com. Accessed Oct-Dec 2025.

[5]"pretty-midi 0.2.11." PyPI, pypi.org/project/pretty-midi/. Accessed 15 Dec. 2025.

[6]Raffel, Colin, and Daniel P. W. Ellis. "Intuitive Analysis, Creation and Manipulation of MIDI Data with pretty_midi." 15th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers, 2014.

[7]Rosen, Charles. *The Classical Style: Haydn, Mozart, Beethoven*. W. W. Norton & Company, 1997.

[8]Rosen, Charles. *The Romantic Generation*. Harvard University Press, 1998.

[9]Scikit-learn Developers. "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, vol. 12, 2011, pp. 2825-2830.