**Computer Systems Technology CST 4702 (Machine Learning)**

**Classifying Classical Music Compositions Using Machine Learning Models**

|  |  |
| --- | --- |
| **Ace Shah**  Department of Computer Science  NYC College of Technology  arjun.shah@mail.citytech.cuny.edu | **Haowei Chen**  Department of Computer Science  NYC College of Technology haowei.chen80@login.cuny.edu |

**Abstract**

Classical music classification, particularly associating compositions with their composers, presents an intricate challenge due to the complex structures and unique stylistic elements of classical music. This research focuses on leveraging machine learning techniques to classify compositions using features extracted from MIDI (Musical Instrument Digital Interface) files. A dataset comprising works from six prominent composers is utilized. Features such as pitch histograms, average note durations, and key signatures were extracted using the music21 library, emphasizing melodic and harmonic attributes. Machine learning models including Random Forest, Gradient Boosting, SVM, MLP, Logistic Regression, and k-NN were trained and optimized using hyperparameter tuning. The Random Forest and Stacked Ensemble models yielded superior results, achieving classification accuracies above 90%. This paper details the dataset, feature extraction process, machine learning methodologies, and model performance comparisons, providing insights into computational musicology and scalable music classification frameworks.

1. **Introduction**

Classical music is a cornerstone of cultural heritage, representing centuries of artistic evolution. Composers like Bach, Beethoven, and Mozart have distinct styles, yet their works often share overlapping musical elements, making classification a non-trivial task. Advances in computational methods, particularly machine learning, offer new opportunities to analyze and classify these compositions objectively and efficiently.

MIDI files serve as a compact yet detailed representation of musical compositions, encoding data on pitch, tempo, duration, and dynamics. This research focuses on using these files to classify compositions from six composers: Alkan, Lully, Scarlatti, Schubert, Tchaikovsky, and Victoria. The goal is to automate the classification process by extracting relevant features from MIDI files, training machine learning models, and evaluating their performance. In addition to achieving high accuracy, the project aims to analyze the strengths and weaknesses of different models and understand the importance of feature engineering in music classification.

Challenges such as imbalanced data distribution, feature redundancy, and computational complexity were addressed through careful preprocessing and model selection. The project emphasizes not only the accuracy of the results but also the interpretability of the models and their scalability to larger datasets.

1. **Dataset**

The dataset consists of 1,791 MIDI files from six composers, with varying numbers of compositions per composer:

|  |  |
| --- | --- |
| Composer | Number of Files |
| Scarlatti | 597 |
| Victoria | 333 |
| Schubert | 270 |
| Tchaikovsky | 239 |
| Alkan | 237 |
| Lully | 115 |

This uneven distribution posed challenges for model training, particularly in avoiding bias toward composers with more compositions. To address this, the dataset was preprocessed to balance class representations and enhance feature extraction.

1. **Parsing MIDI Files**: The music21 library was used to extract features from raw MIDI files. Features included pitch histograms, rhythmic patterns, and harmonic intervals.
2. **Handling Missing or Corrupted Files**: Several MIDI files were incomplete or invalid. These files were either repaired or removed from the dataset.
3. **Feature Normalization**: To ensure comparability across compositions, features such as note durations and pitch frequencies were scaled using standardization techniques.
4. **Machine Learning Models**

This section delves into the machine learning models implemented, their mathematical underpinnings, and their performance in the context of this project. Each model was carefully selected based on its strengths and evaluated using metrics like accuracy, precision, recall, and F1-score.

***3.1 Random Forest***

Random Forest is a robust ensemble learning method that constructs multiple decision trees during training. It aggregates predictions from individual trees to improve accuracy and reduce overfitting. The key strength of Random Forest lies in its ability to handle high-dimensional datasets and provide insights into feature importance.

**Mathematical Representation**:  
The Gini impurity is used to split nodes in the decision trees:

A mathematical equation with numbers and symbols

Description automatically generated

where pi​ represents the proportion of samples belonging to class iii in a node.

**Implementation Details**:

* Hyperparameters such as the number of trees (n\_estimators) and tree depth (max\_depth) were tuned using GridSearchCV. The best parameters were:
  + n\_estimators = 300
  + max\_depth = 20
* The Random Forest model achieved an accuracy of **92.19%**, making it one of the best-performing models.

|  |  |
| --- | --- |
| Metric | Random Forest |
| Accuracy (%) | **92.19** |
| Precision | **0.92** |
| Recall | **0.92** |
| F1-Score | **0.92** |

***3.2 Gradient Boosting***

Gradient Boosting builds an ensemble of decision trees by iteratively optimizing a loss function. Unlike Random Forest, Gradient Boosting focuses on minimizing errors by learning from the residuals of previous iterations. This makes it highly effective for datasets with complex patterns.

**Mathematical Representation**:  
Gradient Boosting minimizes the following loss function:

A black and white image of a mathematical equation

Description automatically generated

where Fm(x) is the model at iteration m, η is the learning rate, and L is the loss function.

**Implementation Details**:

* Key hyperparameters such as learning rate (learning\_rate) and the number of trees (n\_estimators) were optimized. The best configuration was:
  + learning\_rate = 0.1
  + n\_estimators = 200
* Gradient Boosting achieved an accuracy of **89.54%**.

|  |  |
| --- | --- |
| Metric | Gradient Boosting |
| Accuracy (%) | 89.54 |
| Precision | 0.90 |
| Recall | 0.90 |
| F1-Score | 0.90 |

***3.3 Support Vector Machine (SVM)***

SVM is a classification algorithm that identifies the hyperplane that best separates classes in a feature space. For non-linear separability, kernels such as RBF (Radial Basis Function) were used.

**Mathematical Representation**:  
SVM maximizes the margin between the hyperplane and support vectors:

A black text on a white background

Description automatically generated

**Implementation Details**:

* The model was tuned for parameters like C (regularization) and gamma (kernel coefficient). The best parameters were:
  + C = 100
  + gamma = 0.01
* SVM achieved an accuracy of **79.08%**, with performance varying significantly based on feature scaling.

|  |  |
| --- | --- |
| Metric | SVM |
| Accuracy (%) | 79.08 |
| Precision | 0.79 |
| Recall | 0.79 |
| F1-Score | 0.79 |

***3.4 Multi-Layer Perceptron (MLP)***

MLP is a feedforward neural network that uses hidden layers and activation functions to capture complex patterns. The backpropagation algorithm adjusts weights to minimize error.

**Mathematical Representation**:  
The output of a neuron is defined as:

A math symbols on a white background

Description automatically generated

**Implementation Details**:

* The model used a single hidden layer with 100 neurons. Despite careful tuning, convergence was not always achieved, leading to computational inefficiencies.
* MLP achieved an accuracy of **80.61%**.

|  |  |
| --- | --- |
| Metric | MLP |
| Accuracy (%) | 80.61 |
| Precision | 0.80 |
| Recall | 0.81 |
| F1-Score | 0.81 |

***3.5 Logistic Regression***

Logistic Regression models the relationship between features and class probabilities using the sigmoid function. It serves as a baseline for comparison.

**Mathematical Representation**:

A mathematical equation with numbers and symbols

Description automatically generated

**Implementation Details**:

* Logistic Regression was tuned for regularization (C) and class weighting (class\_weight). The final configuration achieved an accuracy of **75.17%**.

|  |  |
| --- | --- |
| Metric | Logistic Regression |
| Accuracy (%) | 75.17 |
| Precision | 0.74 |
| Recall | 0.75 |
| F1-Score | 0.74 |

***3.6 k-Nearest Neighbors (k-NN)***

k-NN classifies a sample based on the majority vote of its kkk-nearest neighbors. It is a simple yet effective algorithm for small datasets.

**Mathematical Representation**:

A black square with square and square symbols

Description automatically generated with medium confidence

**Implementation Details**:

* The optimal number of neighbors (k) was determined to be 3, with weighted distance as the metric. The model achieved an accuracy of **73.08%**.

|  |  |
| --- | --- |
| Metric | k-NN |
| Accuracy (%) | 73.08 |
| Precision | 0.72 |
| Recall | 0.73 |
| F1-Score | 0.72 |

***3.7 Ensemble Methods***

Voting and Stacking ensembles combined predictions from multiple models to improve overall performance. These methods utilized the strengths of individual models and achieved robust results:

* **Voting Ensemble**: Combined probabilities from Random Forest, Gradient Boosting, and MLP. Accuracy: **88.01%**.
* **Stacked Ensemble**: Used Logistic Regression as a meta-classifier to aggregate predictions. Accuracy: **92.05%**.

|  |  |
| --- | --- |
| Metric | Ensemble |
| Accuracy (%) | 88.01 |
| Precision | 0.88 |
| Recall | 0.88 |
| F1-Score | 0.88 |

1. **Feature Extraction**

Feature extraction is a critical step in machine learning projects, particularly in domains like music classification, where raw data is unstructured and lacks direct numerical representations. In this project, MIDI (Musical Instrument Digital Interface) files were analyzed using the music21 library to derive structured features. MIDI files contain detailed information about musical notes, chords, dynamics, and timing, making them ideal for computational analysis.

***4.1 Definition and Importance***

Feature extraction involves transforming raw data into numerical attributes that can be used as inputs for machine learning models. In the context of this project, features were engineered to capture melodic, harmonic, and rhythmic characteristics of compositions. This process was essential for bridging the gap between the symbolic representation of music and the mathematical requirements of machine learning algorithms.

***4.2 Extracted Features***

The features extracted from the MIDI files were selected based on their ability to encapsulate the stylistic nuances of the composers:

1. **Pitch Histogram**:  
   This feature represents the frequency distribution of pitches (notes) in a composition, normalized across 12 semitones in an octave. It provides insights into the tonal preferences of composers.  
   Example: A higher frequency of pitches in the C major scale suggests a preference for compositions in that key.
2. **Average Note Duration**:  
   This feature calculates the mean duration of all notes in a composition, capturing the rhythmic style. Composers with slower tempos often exhibit longer note durations.
3. **Key Signature**:  
   The tonic and mode (major or minor) of a composition were identified using key analysis. This feature highlights the harmonic structure and is particularly useful for distinguishing between composers with contrasting styles.
4. **Number of Chords**:  
   Chords were detected and counted to measure harmonic complexity. For instance, compositions with frequent chord changes tend to exhibit higher harmonic variability.
5. **Melodic Intervals**:  
   This feature quantifies the changes in pitch between successive notes. It provides insights into the melodic tendencies of composers, such as their preference for stepwise motion or leaps.
6. **Tempo and Velocity**:  
   The average tempo (beats per minute) and velocity (intensity of notes) were calculated to understand the dynamic range and pace of compositions.
7. **Chord Progressions**:  
   N-gram analysis of chord sequences was performed to identify recurring harmonic patterns unique to each composer.
8. **Challenges and Time Requirements**

Feature extraction was the most time-intensive part of the project, taking approximately 4–5 hours. Several challenges were encountered:

* **Parsing Errors**: Many MIDI files were corrupted or incomplete, requiring manual inspection and correction.
* **Redundant Features**: Some extracted features were highly correlated, necessitating feature selection to avoid multicollinearity.
* **Computational Load**: Processing large MIDI files with complex polyphonic structures led to increased memory usage and longer execution times.

Despite these challenges, the extracted features effectively captured the essential characteristics of the compositions, enabling the models to achieve high classification accuracy.

1. **Results**

**Performance Metrics**

The performance of the models was evaluated using accuracy, precision, recall, and F1-score. These metrics provide a comprehensive view of model effectiveness:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Accuracy (%) | Precision | Recall | F1-Score |
| Random Forest | **92.19** | **0.92** | **0.92** | **0.92** |
| Stacked Ensemble | **92.05** | **0.92** | **0.92** | **0.92** |
| Gradient Boosting | **89.54** | **0.90** | **0.90** | **0.90** |
| Voting Ensemble | **88.01** | **0.88** | **0.88** | **0.88** |
| MLP | **80.61** | **0.80** | **0.81** | **0.81** |
| SVM | **79.08** | **0.79** | **0.79** | **0.79** |
| Logistic Regression | **75.17** | **0.74** | **0.75** | **0.74** |
| k-NN | **73.08** | **0.72** | **0.73** | **0.72** |

1. **Confusion Matrices**

Confusion matrices provided insights into model-specific misclassifications. For example:

* **Random Forest**: Exhibited minimal misclassifications, with the majority occurring between Tchaikovsky and Schubert due to overlapping harmonic structures.
* **SVM**: Struggled with distinguishing between Alkan and Lully, likely due to their unique pitch distributions.

These matrices highlighted areas for improvement, such as incorporating additional features to resolve ambiguities.

**Time and Space Complexity**

The computational efficiency of the models varied significantly:

* **Random Forest**: Moderate training time with high memory efficiency due to parallel tree construction.
* **Gradient Boosting**: High training time due to sequential tree optimization but offered competitive performance.
* **MLP**: Computationally expensive, requiring significant memory for backpropagation and weight updates.
* **k-NN**: Minimal training time but high inference time due to the distance calculation for all samples.

1. **Comparison of Models**

**Performance Analysis**

Random Forest and Stacked Ensemble models outperformed others in terms of accuracy, recall, and F1-score. Gradient Boosting provided competitive results, albeit with longer training times. Ensemble methods demonstrated the value of combining diverse model strengths, achieving robust and consistent performance.

**Trade-Offs**

* **Accuracy vs. Complexity**: While Random Forest offered high accuracy with moderate complexity, models like Gradient Boosting required more computational resources.
* **Speed vs. Scalability**: Logistic Regression and k-NN were computationally efficient but struggled with scalability and accuracy.

**Justification for Final Model**

The Random Forest model was selected as the primary classifier due to its high accuracy, interpretability, and computational efficiency. The Stacked Ensemble was identified as an alternative for tasks requiring marginally higher accuracy.

1. **Conclusion**

This project successfully demonstrated the effectiveness of machine learning in classifying classical music compositions by composers. By leveraging MIDI files and extracting features such as pitch histograms, average note duration, and harmonic progressions, various machine learning models were trained and evaluated. Among the models tested, the **Random Forest** and **Stacked Ensemble** models emerged as the most effective, achieving accuracy rates exceeding 90%.

**Random Forest** was particularly noteworthy for its ability to handle high-dimensional data and its interpretability, making it an excellent choice for capturing complex relationships between features. Its use of ensemble techniques and feature importance analysis provided robust results with minimal overfitting. On the other hand, the **Stacked Ensemble** model offered marginally improved generalization by combining the strengths of multiple base models, including Gradient Boosting, MLP, and Random Forest. This approach effectively leveraged the diversity of the base models, resulting in a high-performing meta-classifier.

In comparison:

* **Gradient Boosting** also performed well, achieving 89.54% accuracy, but required longer training times due to its iterative optimization process.
* **Support Vector Machines (SVM)** and **Logistic Regression** served as reliable baseline models, offering moderate accuracy but falling short in capturing the complexities of musical compositions.
* **k-Nearest Neighbors (k-NN)** demonstrated simplicity and computational efficiency but struggled with scalability and accuracy for larger datasets.
* **Multi-Layer Perceptron (MLP)**, while powerful, required extensive tuning and computational resources to achieve competitive results.

The **Random Forest** model emerged as the best model in terms of interpretability and computational efficiency, while the **Stacked Ensemble** model offered the most robust performance overall, balancing accuracy and generalization effectively.

The most time-intensive component of the project was feature extraction, which took 4–5 hours due to challenges in parsing MIDI files and addressing inconsistencies. Several features required repeated adjustments to accurately represent the compositions' melodic, harmonic, and rhythmic structures. Despite these challenges, the detailed feature engineering process played a pivotal role in the models’ success.

Looking ahead, future work could explore expanding the dataset to include more composers and compositions, further enhancing the models' diversity and generalizability. Additionally, deep learning approaches, such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), could provide more advanced tools for music classification. Incorporating complex features like tempo variations, polyphonic textures, and dynamic intensity could also enhance the models' classification capabilities.

In conclusion, this project highlights the importance of ensemble techniques and meticulous feature engineering in achieving reliable and robust results. The **Random Forest** model stands out for its efficiency and interpretability, while the **Stacked Ensemble** model demonstrates the potential of leveraging multiple models to achieve superior performance. The insights gained from this study offer valuable contributions to computational musicology and set the stage for future advancements in automated music analysis.

1. **References**
2. Lebar, Justin & Chang, Gary & Yu, David. (2012). *Classifying Musical Scores by Composer: A Machine Learning Approach*.
3. Shi, Sander. (2018). GitHub Repository. *MIDI Classification Tutorial*. Retrieved from <https://github.com/sandershihacker/midi-classification-tutorial>.
4. Cataltepe, Zehra, Yusuf Yaslan, and Abdullah Sonmez. *Music Genre Classification Using MIDI and Audio Features.* EURASIP Journal on Advances in Signal Processing 2007, no. 1 (2007): 036409.
5. Huang, Allen, and Raymond Wu. *Deep Learning for Music.* arXiv preprint arXiv:1606.04930 (2016).
6. Kalingeri, Vasanth, and Srikanth Grandhe. *Music Generation with Deep Learning.* arXiv preprint arXiv:1612.04928 (2016).
7. Lee, Jongpil et al. *Sample-level Deep Convolutional Neural Networks for Music Auto-tagging Using Raw Waveforms.* CoRR abs/1703.01789 (2017).
8. Ruotsi, Ruoho. (2017). GitHub Repository. *LSTM-Music-Genre-Classification*. Retrieved from <https://github.com/ruohoruotsi/LSTM-Music-Genre-Classification/blob/master/lstm_genre_classifier_keras.py>.
9. Dorsey, Brannon. (2017). GitHub Repository. *Midi-RNN*. Retrieved from <https://github.com/brannondorsey/midi-rnn/blob/master/train.py>.
10. Oord, A.V.D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). *Wavenet: A Generative Model for Raw Audio.* arXiv preprint arXiv:1609.03499.
11. music21 Documentation: Retrieved from <https://web.mit.edu/music21>.
12. Scikit-learn Documentation: Retrieved from <https://scikit-learn.org>.
13. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning.* Springer.
14. Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python.* *Journal of Machine Learning Research*, 12, 2825-2830.
15. Stanford CS230 Final Report. (2018). *Classifying Classical Music Compositions Using Machine Learning.* Retrieved from <https://cs230.stanford.edu/projects_fall_2018/reports/12441334.pdf>.