MIDI Classification Using Support Vector Machines (SVM) and Music21

Wilson Roldan

CST 4702

Fall 2024

Abstract

This project focused on classifying MIDI files from six composers—Albeniz, Anglebert, Dowland, Pachelbel, Palestrina, and Telemann—using Support Vector Machines (SVM). The dataset, comprising 378 MIDI files, was processed to extract key musical features such as note count, pitch, density, key and time signatures, and tempo, which were then used to train four SVM models with different kernel functions: linear, polynomial, radial basis function (RBF), and sigmoid. The models were evaluated using metrics such as test accuracy, precision, recall, F1 score, and cross-validation score. The RBF kernel outperformed the others with the highest test accuracy (92%) and the fewest misclassifications, while the linear kernel also performed well. Polynomial and sigmoid kernels exhibited higher misclassification rates. The findings demonstrate the effectiveness of SVMs, particularly with the RBF kernel, for music classification tasks.

Introduction

This project investigates the use of machine learning techniques, specifically Support Vector Machines (SVM) [1], for classifying MIDI music files based on their features. The dataset used consists of MIDI files from six classical composers—Albeniz, Anglebert, Dowland, Pachelbel, Palestrina, and Telemann—chosen for their balanced number of files. The total dataset initially contained 378 MIDI files, which were then preprocessed to remove duplicates and split into training and testing sets. The primary goal of this project is to explore the effectiveness of different SVM kernel functions in classifying these MIDI files based on various extracted musical features.

Feature extraction was carried out using Music21 [2], a Python library designed for handling and analyzing music data. Key features such as the number of notes, average pitch, note density, key signature, time signature, and tempo were extracted from each MIDI file. These features serve as the input for the SVM models, with the composers' names acting as the target labels for classification. The dataset was further preprocessed by encoding categorical features such as key signatures and time signatures, ensuring the data was ready for machine learning tasks [2]. The dataset was then split into a training set (90%) and a test set (10%) using stratified sampling to maintain balanced class distributions [3].

To evaluate the performance of the models, four different SVM models were trained using different kernel functions: linear, polynomial, radial basis function (RBF), and sigmoid [4]. The models were assessed based on several performance metrics, including test accuracy, precision, recall, F1 score, and cross-validation scores [5]. The results showed that the RBF kernel produced the best performance, achieving a test accuracy of 92% and the least number of misclassifications. The linear kernel also performed well, closely following the RBF model in accuracy. In contrast, the polynomial and sigmoid models demonstrated more significant variability in their performance and exhibited higher misclassification rates. This project demonstrates the potential of SVMs for classifying music based on MIDI data, highlighting the importance of kernel choice in optimizing classification results.

Dataset

Six composer files were chosen from the ‘k_collecction’ dataset provided for this project. The dataset contained folders full of MIDI files by a specific composer, as well as loose MIDI files from assorted composers. The six composer files were added to a new folder called

‘Composer_Files’. This new folder contained MIDI files in folders named after the respective composer, for the composers Albeniz, Anglebert, Dowland, Pachelbel, Palestrina, and Telemann.

Each composer file contained roughly 60 files, with a total of 378 files. After removing 2 files that are duplicates, the dataset contains 376 files before separating into training and test set. These composers were chosen because they had a similar number of MIDI files, which would make the distribution of composers more balanced across the dataset. Imbalances in the dataset can strongly affect the SVM model training.

Machine Learning Models

Supervised machine algorithms known as SVM are used to classify MIDI files based on features extracted from the data [1]. They work by finding the optimal hyperplane that separates data points of different classes into higher dimensional space, which is commonly referred to as the kernel trick [3]. A hyperplane is a decision boundary used to separate two different classes [1]. The goal is to maximize the margin, or the distance between the hyperplane and the nearest support vectors. SVMs are commonly used for classification and regression tasks because they are resistant to noisy data, while also working well with linear and non-linear data [1].

Four SVM models were created to test how different kernel types would affect classification results. The four kernel functions used are linear, polynomial, radial basis function (RBF), and sigmoid [3]. These different SVM models will be used to fit or train the model using provided training data and corresponding target labels. During this process, the model learns patterns and relationships in the data that are used to make predictions about data outside of the training set.

Each of the SVM models is created using the SVC function from Sklearn [6]. The parameters of the function used in this project are kernel, ‘C’, gamma, and random_state. The kernel represents the kernel functions like linear or RBF. The ‘C’ parameter is the regulation parameter used to control the tradeoff between maximizing the margin or minimizing classification error. A higher ‘C’ value means the model will try harder to classify all training data points correctly, while a lower value allows for more misclassifications. A higher value may lead to overfitting because the model is too complex. A lower value may lead to underfitting by prioritizing generalization by sacrificing margin size. The gamma is the parameter that controls the influence of a single training example on the decision boundary. The random_state parameter is a seed for the random generator used for reproducibility.

Feature Extraction

Feature extraction is carried out in two steps, by two helper functions. The first helper function takes a composer’s name as a parameter, as well as a base file path to the ‘Composer_Files’ folder in the working directory. The method then checks if that file name exists in the base file path folder. If the file exists, it iterates through all MIDI files in the folder using Music21’s converter.parse function to load the file into a score object [2]. The function stores filename, parsed score, and composer’s name in a list. An error message prints if any file cannot be processed.

The second helper function takes the list output from the first helper function as a parameter, and extracts features from the score object. The function begins by initializing an empty list to store feature dictionaries and iterates through parsed scores to retrieve MIDI file metadata like filename, score object, and composer. It then uses the score object to extract features like number of notes, average pitch, note density, key signature, time signature (tonic

and mode), and tempo [2]. Note density is measured by the number of notes per quarter-length of the score. The default tempo of 120 bpm (beats per minute) is used for any score where tempo is unavailable. The composer's name is treated as a feature and is stored for use later as a class label.

If an error occurs while processing the score objects, the function prints an error message indicating the problematic file. After processing all files, the function converts the list of feature dictionaries into a Pandas dataframe to make it easier for analysis. Together, these two helper functions provide a structured way to quantify musical characteristics from MIDI files in a way suitable for machine learning tasks. The two functions work on the subfolders of 'Composer_Files' to reduce RAM usage. This means that each composer folder is parsed into a score, then has its features extracted and stored in a dataframe. The dataframes for each composer are then concatenated into a single dataframe that is used to split into training and test sets. The figure below shows the first 5 rows of the dataframe that holds all feature data.

	Num_Notes	Avg_Pitch	Note_Density	Key_Signature	Time_Signature	Tempo	Composer
0	1125	55.558222	2.710843	D major	3/4	150.0	albeniz
1	2104	44.264734	3.053701	E- major	2/4	132.0	albeniz
2	832	52.564904	2.390805	F# minor	3/4	112.0	albeniz
3	3288	40.896290	7.506849	E- major	3/4	84.0	albeniz
4	3141	40.155683	4.060763	B major	2/4	140.0	albeniz

Before splitting the dataset into training and test sets using Sklearn libraries, the ColumnTransfer [7] function from Sklearn is used for one hot encoding for non-numerical features like key signatures and time signatures. Class labels, composer names, are also converted into numerical values using LabelEncoder from Sklearn [8]. The train_test_split

function is used to split the dataset in a way that 90% of the dataset is used for training and 10% is used for testing, while the distribution of classes is stratified or evenly distributed.

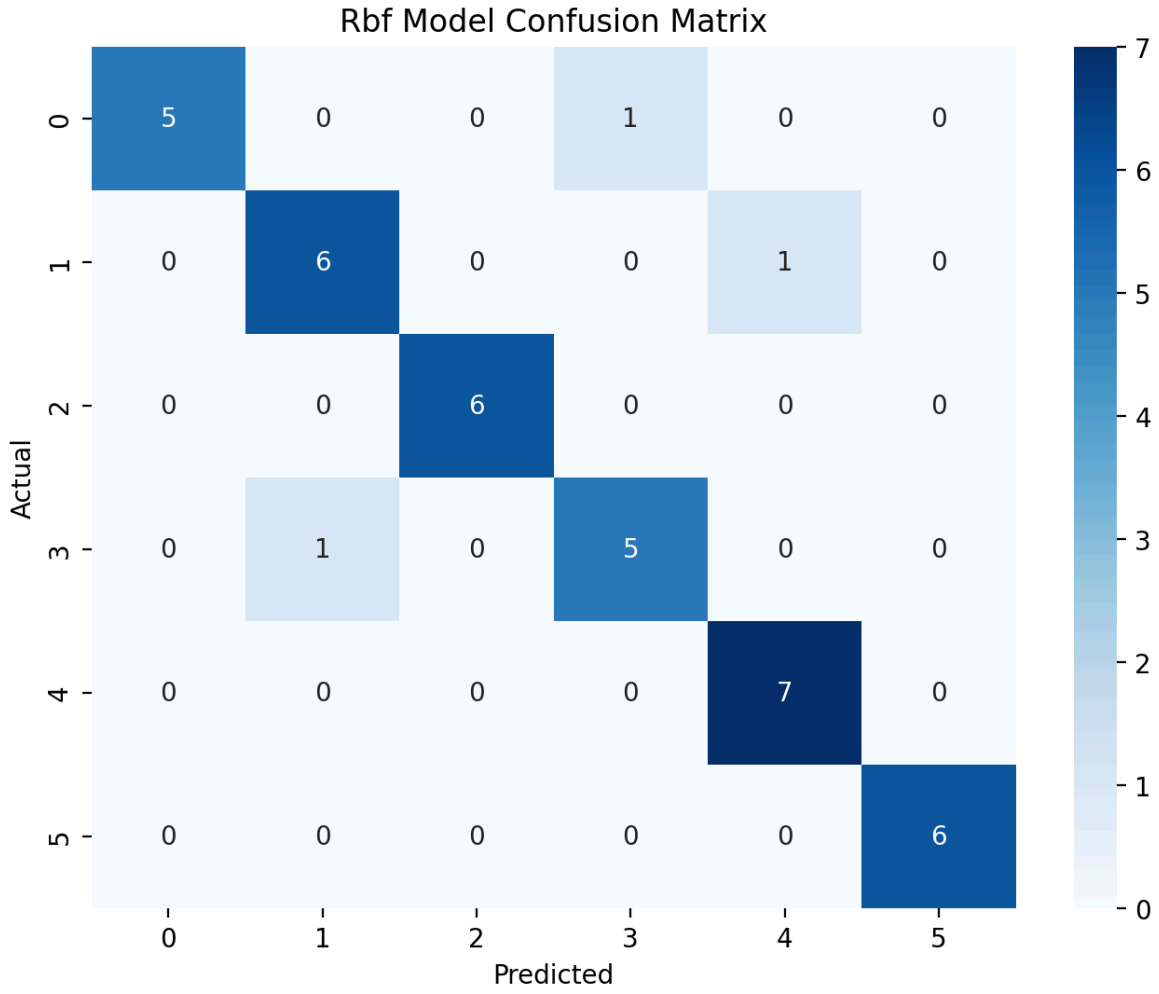
Results

Sklearn libraries are used for evaluating the performance of each model. The evaluation metrics we are interested in are test accuracy, precision, recall, F1 score, and cross-validation score [9]. Cross-validation is used to evaluate how the model generalizes based on how the model performs on the equally split folds or subsets of the dataset. This is valuable because it offers a more dependable evaluation of the model's performance on unseen data compared to a single train-test split.

	Model	Accuracy	Precision	Recall	F1-Score
0	RBF	0.921053	0.924342	0.921053	0.920734
1	Linear	0.910192	0.894737	0.894737	0.892579
2	Sigmoid	0.684211	0.716599	0.684211	0.671053
3	Polynomial	0.710526	0.840351	0.710526	0.697488

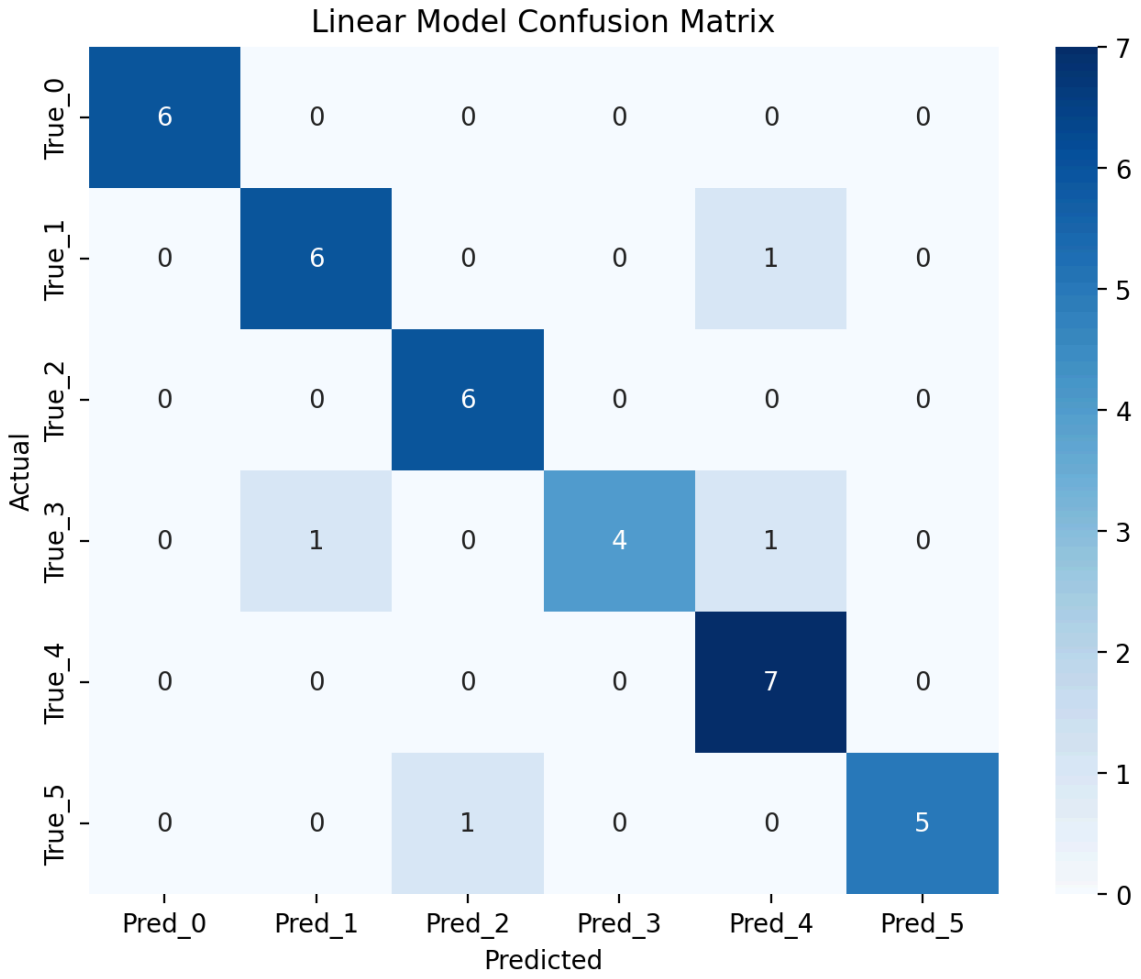
The RBF model achieved a test accuracy of 92%, average precision of 92%, average recall of 92%, and an average F1 score of 0.92. The average cross-validation accuracy for the training set is 0.7161. The parameters for the SVC function were kernel = 'rbf', 'C' = 1.0, gamma = 'scale', and random_state = 543. The RBF confusion matrix [10] illustrates that there are few misclassifications because there are high values on the diagonal, which indicate that the predicted classes match the actual classes at a high rate. There are 3 misclassifications for this model. The time complexity and space complexity are O (n^2 * d), where n represents the number of samples and d represents the number of features. It takes 11.7 ms ± 2.02 ms per loop (mean ±

std. dev. of 7 runs, 100 loops each) for the RBF section of code to run, while memory usage peaks at 1972.39 MiB or about 2GB of memory.



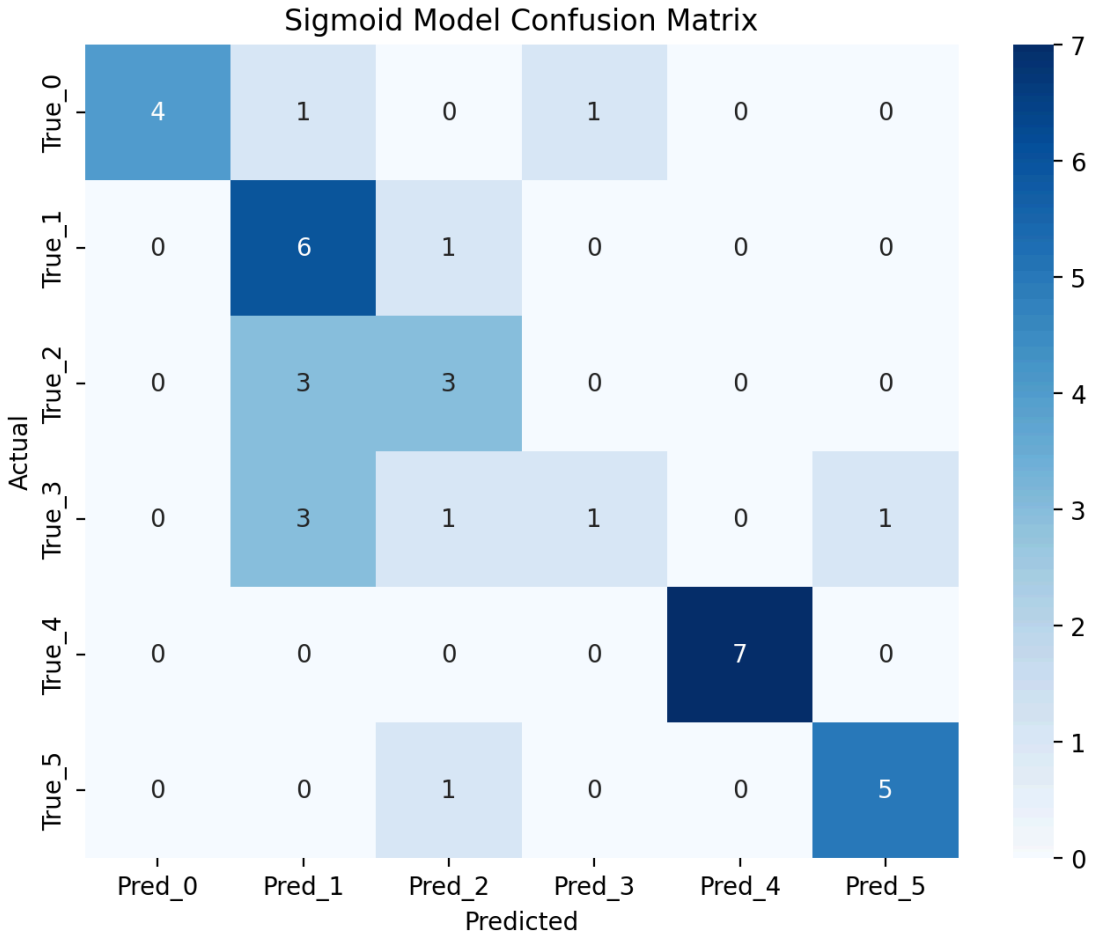
The linear model achieved a test accuracy of 91%, average precision of 89%, average recall of 89%, and an average F1 score of 0.89. The average cross-validation accuracy for the training set is 0.7132. The parameters for the SVC function were kernel = ‘linear’, ‘C’ = 1.0, gamma = ‘scale’, and random_state = 543. The linear confusion matrix illustrates that there are few misclassifications because there are high values on the diagonal, which indicate that the predicted classes match the actual classes at a high rate. There are 4 misclassifications for this

model. The time complexity and space complexity are O (n^2 * d), where n represents the number of samples and d represents the number of features. It takes 8.79 ms ± 125 µs per loop for the linear section of code to run, while memory usage peaks at 1972.39 MiB or about 2GB of memory.



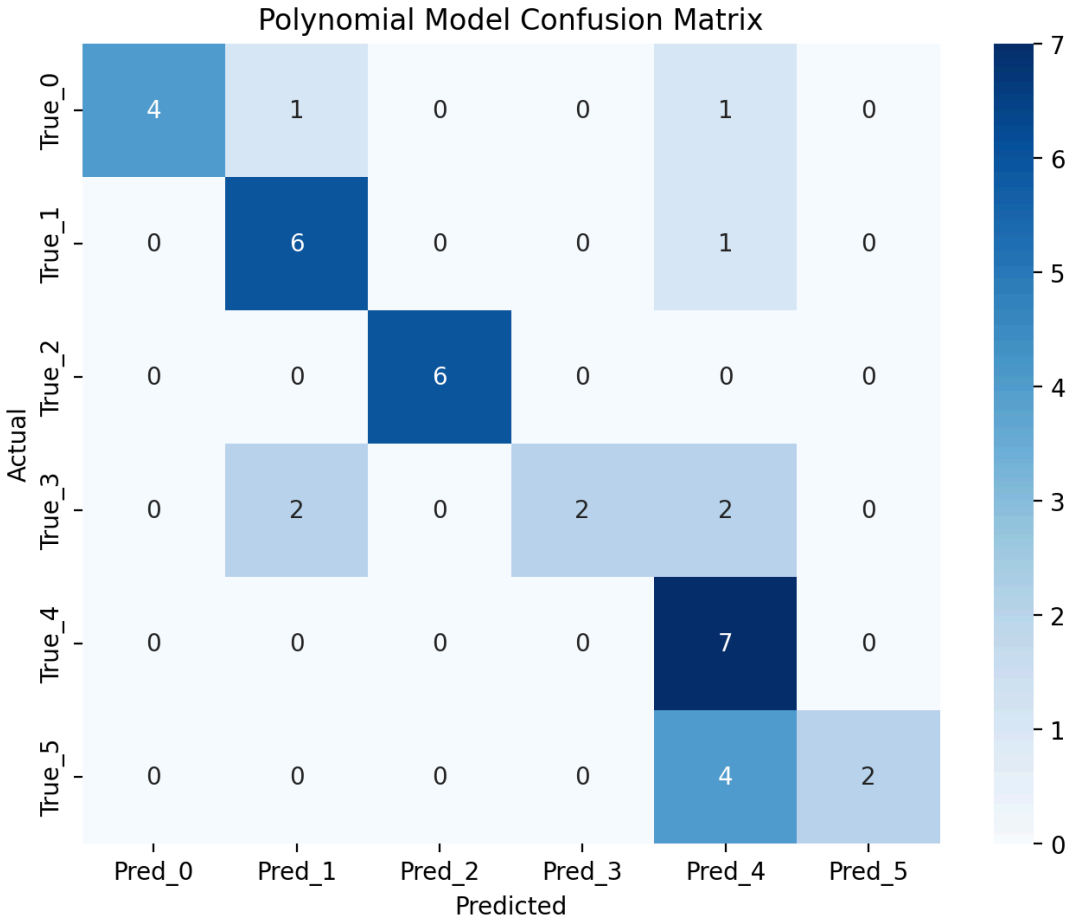
The sigmoid model achieved a test accuracy of 68%, average precision of 72%, average recall of 68%, and an average F1 score of 0.67. The average cross-validation accuracy for the training set is 0.5769. The parameters for the SVC function were kernel = ‘sigmoid’, ‘C’ = 1.0, gamma = ‘scale’, and random_state = 543. The sigmoid confusion matrix illustrates that there are many misclassifications, but 4 of the 6 composers are classified well. There are 12

misclassifications for this model. The time complexity and space complexity are O (n^2 * d), where n represents the number of samples and d represents the number of features. It takes 11.7 ms ± 2.39 ms per loop for the sigmoid section of code to run, while memory usage peaks at 1972.39 MiB or about 2GB of memory.



The polynomial model achieved a test accuracy of 71%, average precision of 84%, average recall of 71%, and an average F1 score of 0.69. The average cross-validation accuracy for the training set is 0.6063. The parameters for the SVC function were kernel = ‘poly’, ‘C’ = 1.0, gamma = ‘scale’, and random_state = 543. The polynomial confusion matrix illustrates that there are many misclassifications, but half of the composers are classified well. There are 11

misclassifications for this model. The time complexity and space complexity are O (n^2 * d), where n represents the number of samples and d represents the number of features. It takes 17.3 ms ± 4.67 ms per loop for the polynomial section of code to run, while memory usage peaks at 1972.39 MiB or about 2GB of memory.



Comparison of Models

Time complexity and space complexity are the same across models because they use the same machine learning techniques (SVMs), with the only difference being the kernel function as the varying factor. Across all models, reducing or increasing the ‘C’ parameter showed a decrease in test accuracy. This is also true for the gamma parameter, as the ‘scale’ option

outperformed the ‘auto’ option for each SVM model. Additionally, while it is typical to split the dataset into an 80% training set and a 20% test set, all models showed improved performance with a 90% training set and a 10% test set. I believe this is because the dataset is very small (less than 400 data points). Stratifying the dataset split to maintain the distribution among classes also increased the test accuracy across all models.

The RBF model performed the best based on the evaluation metrics. It achieved the highest test accuracy of 0.92 and the highest average cross-validation accuracy on the training set of 0.7161. This model also had a total of 3 misclassified data points, which is the least of all the SVM models. The linear performs relatively well and is closely comparable to the RBF model. The linear model achieves a test accuracy of 0.91 and the average cross-validation accuracy on the training set is 0.7132. This model has the second least misclassifications with a total of 4. The polynomial and sigmoid models exhibited the most variance when tuning hyperparameters and were clearly outperformed by the RBF and linear models.

Conclusion

In this project, MIDI music files were classified based on their features by using SVMs and Music21. Different SVM models were used to assess the performance and accuracy of classification using a dataset of 378 MIDI files by 6 different composers. Evaluation metrics like test accuracy, precision, recall, F1 score, and average cross-validation accuracy on the training set were used to measure the performance of each model. The results demonstrate SVM models, particularly when using different kernel functions, create hyperplanes and maximize margins to distinguish different classes of a dataset for classification based on features of the music file.

References

- [1] "Support vector machine," Wikipedia, The Free Encyclopedia, Dec. 21, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine. [Accessed: Dec. 22, 2024].
- [2] "Music21 Documentation," Music21, Dec. 21, 2024. [Online]. Available: https://www.music21.org/music21docs/index.html. [Accessed: Dec. 22, 2024].
- [3] "sklearn.model_selection.train_test_split — Scikit-learn 1.2.0 documentation," Scikit-learn, Dec. 19, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. [Accessed: Dec. 22, 2024].
- [4] "Kernel method," Wikipedia, Dec. 19, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Kernel_method#Mathematics:_the_kernel_trick. [Accessed: Dec. 22, 2024].
- [5] "sklearn.metrics — Scikit-learn 1.2.0 documentation," Scikit-learn, Dec. 19, 2024. [Online]. Available: https://scikit-learn.org/stable/api/sklearn.metrics.html. [Accessed: Dec. 22, 2024].
- [6] "Support Vector Machines — Scikit-learn 1.2.0 documentation," *Scikit-learn*, Dec. 19, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html. [Accessed: Dec. 22, 2024].
- [7] "sklearn.compose.ColumnTransformer — Scikit-learn 1.2.0 documentation," Scikit-learn, Dec. 19, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html. [Accessed: Dec. 22, 2024].

- [8] "sklearn.preprocessing.LabelEncoder — Scikit-learn 1.2.0 documentation," Scikit-learn, Dec. 19, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html. [Accessed: Dec. 22, 2024].
- [9] "sklearn.model_selection.cross_validate — Scikit-learn 1.2.0 documentation," Scikit-learn, Dec. 19, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html. [Accessed: Dec. 22, 2024].
- [10] "sklearn.metrics.confusion_matrix — Scikit-learn 1.2.0 documentation," Scikit-learn, Dec. 19, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html. [Accessed: Dec. 22, 2024].