Stacking Classifier Approach for Music Genre Classification

Momoe Nomoto, Yuechen Wang

Abstract

Automatic music genre classification is essential for efficient and reliable labeling of a large corpus of digitized music and for music recommendation systems, which rely on genre information to suggest similar music to listeners. This paper proposes an approach that utilizes multiple machine-learning classification algorithms (KNN, SVM, and Random Forest) and an ensemble learning technique to create an optimal meta-model that provides more accurate classifications. Our meta-model achieves an accuracy score of 0.69 in the test set, which exceeds the average accuracy of the models we have reviewed so far.

Introduction

The management of a large corpus of digitized music requires accurate genre labels. When a musician uploads new music, they must choose the correct genre label, which can be a time-consuming and error-prone task. Automatic music genre classification eliminates the need for manual categorization, resulting in more efficient and reliable labeling. Moreover, automatic genre classification is crucial for music recommendation systems, which rely on genre information to suggest similar music to listeners. In cases where older music lacks genre labels, machine learning techniques can be used to classify these pieces, providing faster and more accurate results. With all these benefits, automatic music genre classification can greatly improve the management and accessibility of music collections.

Our approach involves using multiple machine learning classification algorithms, including K-Nearest Neighbor (KNN), Support Vector Machine (SVM), and Random Forest (RF), and applying an ensemble learning technique such as stacking to create an optimal meta model that provides more accurate classifications. While we considered using other models like CNN, a comparison paper [1] demonstrates that deep learning is no more effective than SVM, so we plan to use models other than deep learning models. Although individual base learners may not effectively fit the data, stacking addresses this issue by learning the relationship between the prediction results of each of the ensembled models on out-of-sample data and the ground truth. Stacking then combines the base models in a way that reduces variance and improves the overall accuracy of the ensemble model. We will also be creating our own dataset to extract a more

comprehensive set of spectral and rhythmic audio features of each of our predefined genres: blues, classical, disco, electronic, hiphop, jazz, pop, and rock. Our goal is to leverage a dataset with more comprehensive features and ensembling to improve the accuracy and reliability of music genre classification.

Dataset

Dataset Creation

Current research in music information retrieval often utilizes the GTZAN dataset [2], the Million Song Dataset [3], and audio features provided by Spotify API. Upon conducting a thorough examination of the datasets, we discovered that the available audio features were limited, raising concerns that a minimal set of audio features would not be adequate for predicting a class among a large set of classes. We opted to generate our own dataset by extracting audio features from songs using a pre-existing Python package called Librosa [4]. Since the Spotify API can provide a wealth of information including 30 second song previews, we downloaded the .wav files of songs from playlists of the predefined genres. We then extracted the following features for each song: mel-frequency cepstral coefficients (MFCCs), chroma energy normalized values (CENS), roll-off frequency, mel-scaled spectrogram values, rms values of power spectrogram for harmonic and percussive parts, spectral centroid, p-th order spectral bandwidth, spectral contrast, coefficients of fitting an nth-order polynomial to the columns of the spectrogram, tonal centroid features, zero crossing rate, onset strength, bpm, and beats information. We first decompose the audio into percussive and harmonic spectrograms (see Figure 1), and then calculate the average and standard deviation for each type of feature based on root-mean-square (RMS) per frame from the spectrograms, which results in a total of 79 features. Since many of these features are likely to be correlated, we plan to employ several preprocessing techniques to mitigate this issue. For instance, we intend to use principal component analysis (PCA) and feature selection to reduce the dimensionality of the dataset and remove redundant features.

Dataset Overview

Our dataset comprises a total of 1120 data points, with a balanced distribution of 140 songs from each of the eight genres: blues, classical, disco, electronic, hip hop, jazz, pop, and rock. To evaluate the performance of the stacked model, we reserved 30 songs from each genre as the test dataset. The remaining data points were partitioned into subsets for training the individual models and the meta model via cross-validation.

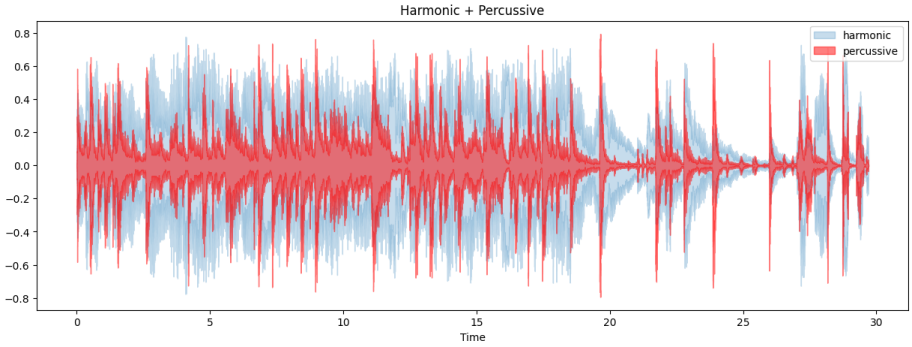

Figure 1: Percussive & Harmonic Signals Separation Before Feature Extraction

Solution

Pipeline Overview

Figure 2 illustrates the pipeline we utilized for our stacked ensemble, which involves three base models and a higher-level meta model. We initially applied our training data to determine the best preprocessing techniques and parameters for our base models before training our stacked classifier using these optimized parameters. As our dataset is well-balanced across all classes, we have relied on this attribute to guide many of our decisions, such as the selection of accuracy as the strategy for evaluating the performance of cross-validated models. The "FunctionTransformer" shown in the diagram refers to a self-defined feature dropping function, and we will describe the choice of best parameterization in more detail in the subsequent subsections.

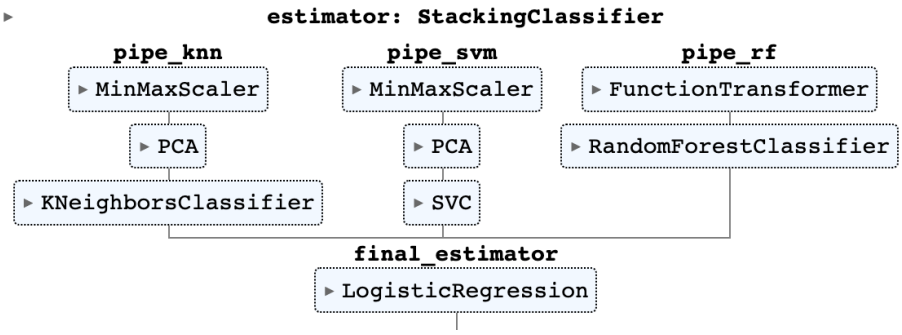

Figure 2: Stacking Pipeline

Individual Estimators

K-Nearest Neighbor

As we are dealing with a large number of features compared to the number of samples, we have included feature scaling and PCA as preprocessing steps to mitigate redundancy, multicollinearity, and dimensionality issues. The KNN classifier has various hyperparameters such as leaf size, number of neighbors, and power parameter for the Minkowski metric. To determine the optimal hyperparameters, we explored the parameter space by trying values from 1 to 30 for the number of neighbors, 1 to 50 for the leaf size, and either Manhattan distance or Euclidean distance for distance calculation. After performing stratified 10-fold cross-validation, we found that the best hyperparameters for the KNN classifier were a leaf size of 28, 20 neighbors, and the Manhattan distance metric, as shown in Figure 4. Moreover, we also experimented with different numbers of principal components to identify the number that accounts for the optimal amount of variance and yields the best testing accuracy. We found the optimal number of principal components to be 25 for KNN classifier, accounting for 88% of the variance, as shown in Figure 3. The combination of these parameters resulted in an accuracy of 0.632 on the testing set. Furthermore, prior to performing PCA, we mean normalized the data to ensure that all features were on the same scale.

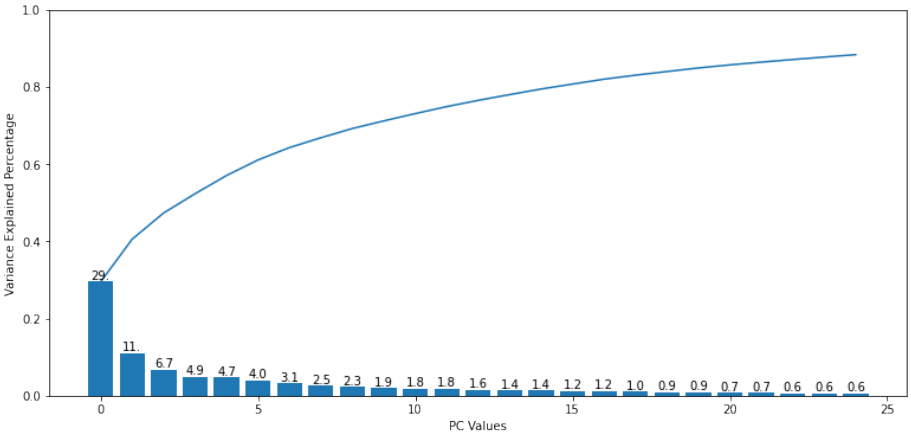

Figure 3: Scree Plot for Training Data

Support Vector Classifier

The soft margin support vector classifier relies on hyperparameters such as regularization parameter C, kernel coefficient gamma, and the type of kernel. Figure 5 displays the testing and validation scores for variation in each of the hyperparameters. Clearly, we see that the radial basis function (RBF) kernel

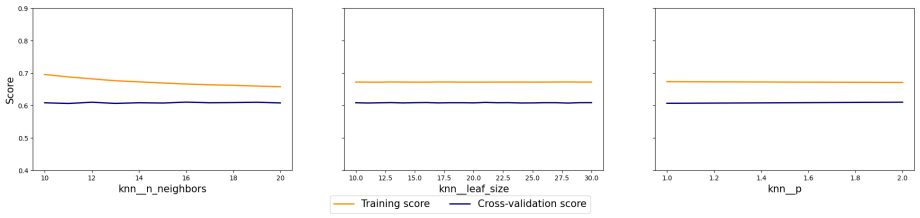

Figure 4: KNN Training and Validation Curves for Hyperparameters

is the best fit for our non-linear dataset. The parameter space also consists of values from 0.1 to 100 for C and values from 0.001 to 1 for gamma. We found that a gamma value of 0.1 and a C value of 10 yielded optimal testing results after conducting a 10-fold stratified cross validation. These hyperparameter values remained consistent as we adjusted other parameters such as the number of principal components, which differed from the KNN classifier's high sensitivity to preprocessing steps and sample size. Overall, the SVC performed better on the testing data than KNN, with an accuracy of 0.743.

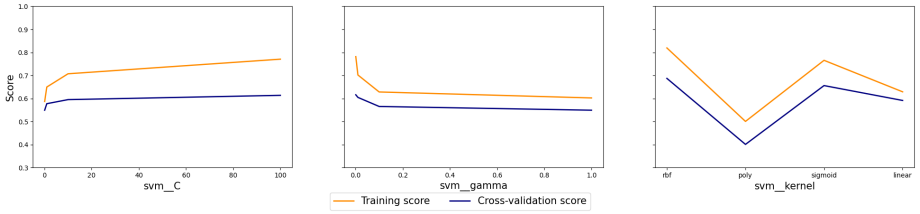

Figure 5: SVC Training and Validation Curves for Hyperparameters

Random Forest

Before directly applying random forest, we first began by training individual decision tree classifiers as a comparison with other tree models we applied later. For the individual decision tree classifier, we used cross-validation to select the best hyperparameters including alpha, max depth of trees, and the minimal number of samples per leaf node. The best testing accuracy score is 0.477 when the hyperparameters are 0.01, 7, and 2. In order to improve the performance of individual decision tree classifiers, we decided to try two ensemble methods - bagging & boosting. Decision tree classifiers with bagging applied achieved the best accuracy score of 0.599 when n_estimators = 50, alpha = 0.01, max_depth = 8, and min_samples leaf = 1, which indicates a significant improvement compared with individual decision tree classifiers. As for boosting, we tried both Adaboost and gradient boosting while the former has rather poor performance since its testing accuracy remains the same disregarding values of hyperparameters. We suspect this happened because the Adaboost method fits the binary

classification problems better. Gradient boosting yields a slight improvement in the performance of the test set. Cross-validation results show that the testing accuracy hits 0.639 with learning rate = 0.15, the number of trees = 100, max_depth = 4. In the end, we applied the random forest model whose performance is very similar to the decision tree classifier + gradient boosting as n_estimators = 250, min_samples_leaf = 1, max_depth = 15.

To further reduce model variance and improve testing accuracy, we implemented feature selection based on two criteria separately. We first select features by the feature importance score given by the random forest classifier. We set different thresholds within the range from 0.005 to 0.025 and applied cross-validation to find the best hyperparameters. However, the first result we get indicates that an underfitting problem might be involved in some thresholds, we think this problem is caused by a small test size, therefore, we adjust our test size from 0.33 to 0.4, which solves the problem. The graph (Figure 6) after tuning the test size shows when we filter out features whose importance is less than 0.0075, the testing accuracy would be 0.656 (max_depth = 10, min_samples_leaf = 1, n_estimators = 200), which is as good as the validation score. Next, we applied feature selection based on the feature correlation score, but we do not consider the result reliable enough because its testing accuracy appears to be more fluctuating (Figure 7). In addition, we also tried to combine these two criteria, but no significant improvement is presented compared with the model applied feature selection solely based on feature importance.

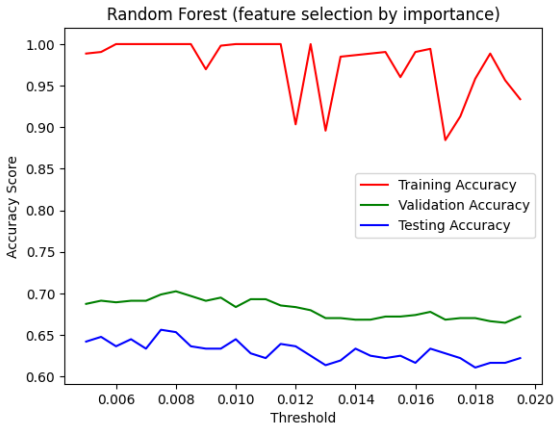

Figure 6: Random Forest: Feature Selection by Importance Score

Stacking Classifier

Upon tuning the hyperparameters for each individual estimator and identifying the optimal settings, we will use these hyperparameters to train the individual estimators. Additionally, we will use the predictions generated by the individual

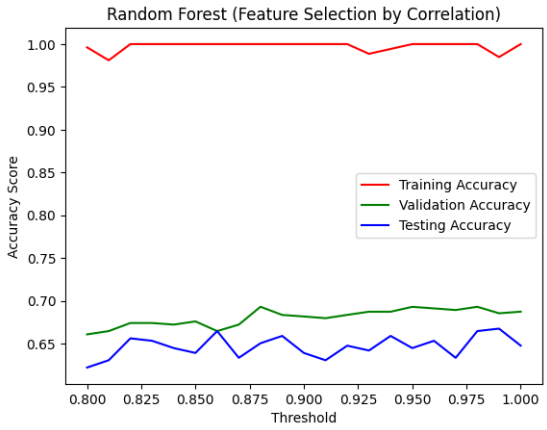

Figure 7: Random Forest: Feature Selection by Correlation

models to train the meta estimator. A stratified 5-fold cross-validation to find the optimal value for the regularization parameter C for the final estimator logistic regression over the space of np.logspace(-2, 2, 20) resulted in Figure 8 where the dark lines are the mean values and the shaded areas are the standard deviations. The optimal value for C was found to be 1.27. We used logistic regression as the final estimator to blend the individual estimators as it is simple and best for classification. Training accuracy was on average 0.92 and cross validation accuracy was on average 0.72.

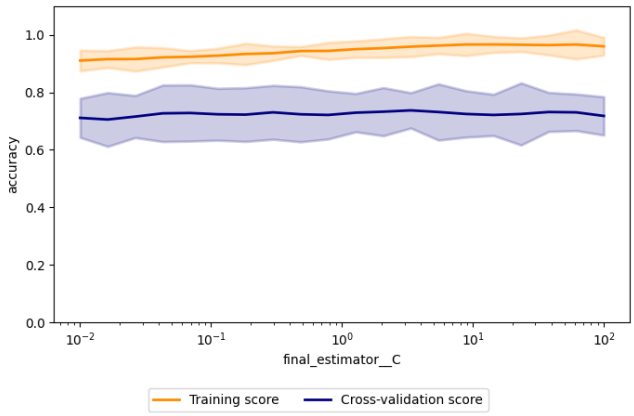

Figure 8: Accuracy Plot for Training and Validation Scores for Stacking Classifier

Results and Discussion

After determining all the hyperparameters and training the estimators, we evaluated the performance of the stacking classifier on a previously unseen dataset of 30 songs from each genre. Table 1 contains the precision, recall, and f1 scores of each of our class and the overall average. The recall is highest among the indicators, which is the percentage of a certain class correctly identified, and we value this indicator more for our application as opposed to precision. The accuracy score from the testing data is 0.69. Figure 9 displays the confusion matrix of the testing data, and we can see that blues was often mistaken for classical and jazz. The connection between blues and jazz becomes clear when considering their shared characteristics, such as the prominent use of "blue" notes, swung notes, and syncopated rhythms. Blues and classical have similar chord progressions and a walking bass. In fact, the genre of classical blues emerged from the fusion of these two styles. Low recall is also prominent in pop music because pop songs typically incorporate various features from other genres, resulting in a more diverse and less distinguishable set of characteristics. On the other hand, classical and rock almost always had the correct prediction. Rock has distinctive audio features, which are characterized by the presence of electric guitars, bass guitars, and drums played by bands. Classical puts great emphasis on tonal harmony and single-line melodies. The presence of these distinct features eases the task of predicting the correct class for new samples by the model. Furthermore, jazz has the highest precision, producing the least false positives. In comparison, when using this same set of testing data to predict using the base models, KNN gives an accuracy of 0.629, SVM gives an accuracy of 0.692, and random forest gives an accuracy of 0.671.

	Precision	Recall	F1-Score
Blues	0.41	0.30	0.35
Classical	0.68	0.95	0.79
Disco	0.53	0.70	0.60
Electronic	0.77	0.75	0.76
Hiphop	0.69	0.63	0.66
Jazz	0.88	0.76	0.81
Pop	0.54	0.44	0.48
Rock	0.66	0.83	0.74
Accuracy			0.67
Macro Avg	0.65	0.67	0.65
Weighted Avg	0.67	0.67	0.66

Table 1: Classification Report for Testing Data on Stacking Classifier

We are aware that our dataset is limited in size, which can result in overfitting, where the model fits too closely to the training data and does not generalize well to new data. Our full dataset resides in our Github respository [5]. With a larger dataset, the ensembling technique will perform better. However, there

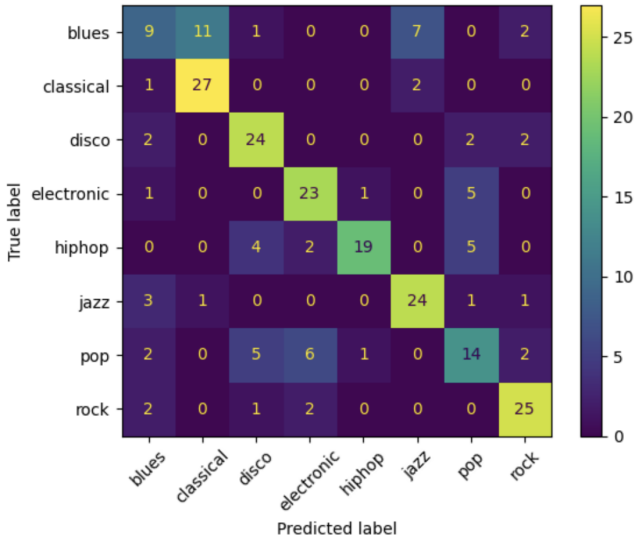

Figure 9: Confusion Matrix for Stacking Classifier

is no guarantee that stacking will always lead to better performance, as it only tends to improve performance on average. If you have several different models that produce different predictions, the accurate predictions from certain models can compensate for the errors from other models, leading to the average of the predictions gradually converging towards the ground truth.

Another ensemble method we experimented with is the majority voting classifier, which led to an accuracy score of 0.683. Therefore, the majority voting ensembling technique is no better than the stacking method with the extent of our dataset.

To enhance the accuracy of our models, we will need a much larger set of data from each genre. Additionally, we can improve the quality of our data by selecting songs that are more homogeneous within a genre and have a higher production quality. Although we have taken measures to select playlists that are predominantly of one genre, it is still possible that some songs within the playlists may contain elements of multiple genres. As a result, a next step in genre prediction could be to predict multiple genre labels for each song to provide a more accurate representation of its genre. This approach could potentially improve the accuracy of our models and help capture the complexities and nuances of music genres that cannot be captured by a single label.

References

[1] A. Elbir, H. Bilal Çam, M. Emre Iyican, B. Öztürk, and N. Aydin, "Music genre classification and recommendation by using machine learning techniques," in *2018 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2018, pp. 1–5.

[2] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.

[3] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[4] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, vol. 8, 2015.

[5] M. Nomoto and Y. Wang, "Ml-final-project," https://github.com/momoenomoto/ML-final-project/, 2023, accessed on: May 12, 2023.