Music Genre Classification

Name:	**H. Dinesh Adhitya, K. Rohit Taeja**
Registration No./Roll No.:	18097, 18364
Institute/University Name:	IISER Bhopal
Program/Stream:	EECS
Problem Release date:	February 02, 2022
Date of Submission:	24^{th} April 2022

1 Introduction

There have been attempts to try and understand sound and what differentiates one song from another[1]. How to visualize sound. What makes a tone different from another? The aim is to identify potential features of individual genres to identify the types of music genres using existing techniques to identify salient features of the given data. The performance of different classification techniques on the training data is demonstrated, justifying that the proposed feature selection scheme is working well. The best framework is then run on the test data and the class labels are given in a text file.

The dataset contains 900 audio samples of music and each song corresponds to one of 10 different genres. There are 52 different features corresponding to each data point. These features correspond to the audio features of the signal (length, tempo, spectrogram features, zero crossing rate, etc.), as shown in 1.

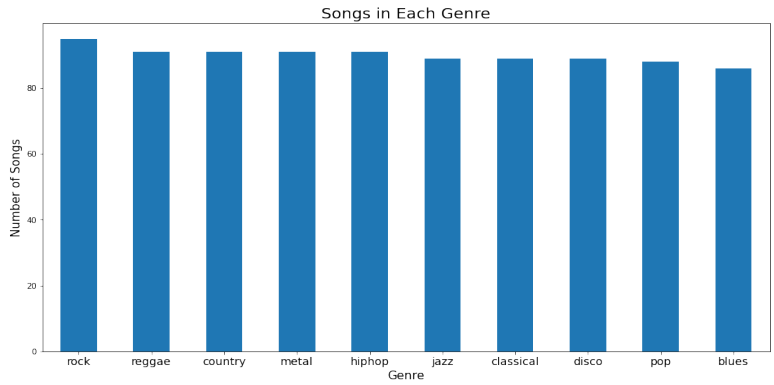


Figure 1: Overview of Corpus

Here, various classifiers are tested to come up with a model that can accurately classify the data, as measured by the f1 score. It is shown that the Random Forest classifier gives the best accuracy score of 0.80. The rationale behind this being that decision trees are ensemble learning methods, which give the edge needed to solve this problem, since a diverse range of features exist.

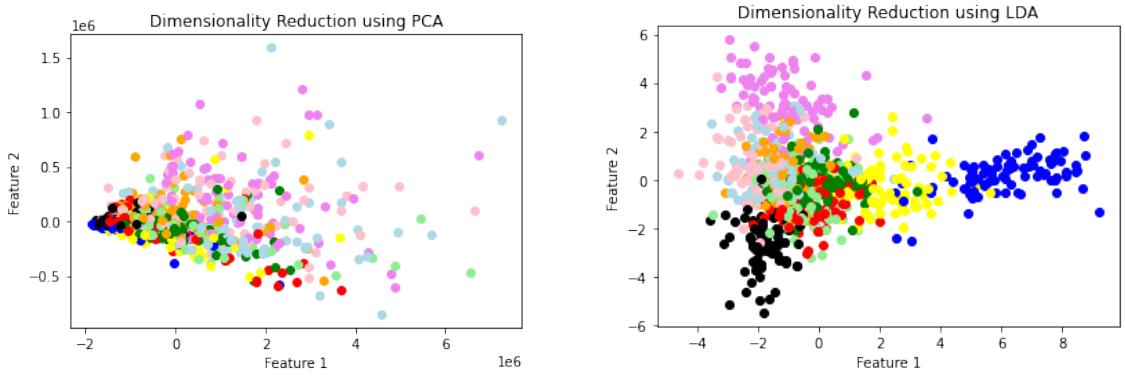
2 Methods

Being a classification task, all classification algorithms were run on the data. Namely, Logistic Regression, Decision Trees, Random Forests, SVMs, Multinomial Naive Bayes, KNNs and AdaBoost. A deep learning model was also run on the data. All hyper-parameter tuning was done using grid search

(with f1 score as the accuracy metric), which is present in python[2] as an inbuilt function[1] in the scikit library.[3]

2.1 Pre-Processing

By looking through the scatter plots of the various columns of the data, it is observed that majority of them follow a gaussian distribution. So, we scale the models first using the StandardScaler function. Given the multidimensional nature of the data, dimensionality reduction was hypothesised to be useful. PCA and LDA were thus performed to visualise possible groups in the data. However, as seen in 2, no clear separation was visible.



((a)) Principal Component Analysis

((b)) Linear Discriminant Analysis

Figure 2: Dimensionality Reduction

2.2 Algorithms

Multinomial Naive Bayes was the first classifier tested. It was implemented using the inbuilt python function[2] provided in scikit. The hyper-parameter tuning was done over α values 0.1, 0.5 and 1. The resulting model had a α value of 0.5.

Logistic Regression was implemented using the inbuilt python function[3] provided in scikit. The hyper-parameter tuning was done over the c values 0.01, 0.1 and 1; and the various algorithms to use in the optimisation, namely newton-cg, lbfgs and liblinear. The parameters of the resulting model were $c = 0.1$, and the algorithm was 'newton-cg'.

A certain kind of SVM algorithm, namely the SVC is used here. It can be found as an inbuilt python function[4] provided in scikit. The hyper-parameter tuning was done over the c (regularization parameter) values 0.1, 1, 2, 10, 50, 100; and the various kernels that are to be used, namely linear, rbf, poly and sigmoid. The parameter value after the tuning was $c = 2$.

The KNeighborsClassifier is an inbuilt python function in the scikit library, which implements learning based on the k nearest neighbors of each query point. We use this function, that can be found inbuilt [5] in python, provided in the scikit library. The hyper-parameter tuning was done over the k (no. of neighbors) values 5, 10, 15, 20; the weight functions used, uniform and distance; the p values 1 (Manhattan distance) and 2 (Euclidean distance); and finally, the algorithm used to compute the nearest neighbors, namely auto, ball_tree and kd_tree. After hyper-parameter tuning, the resulting weight function was 'distance' and the p value was 1.

Decision Tree was implemented using the inbuilt python function[6] provided in scikit. The hyper-parameter tuning was done over the splitting measures, namely gini and entropy; the function of

[1] GridSearchCV function
[2] MultinomialNB function
[3] LogisticRegression function
[4] SVC function
[5] KNeighborsClassifier function
[6] DecisionTreeClassifier function

number of maximum features to consider when splitting the data, namely auto, sqrt and log2; the values for the maximum depth of the tree, 10, 40, 45, 60; and finally the ccp_alpha value, which has to do with pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha value is chosen. The values were from 0.01 to 0.1 in steps of 0.01. After tuning, the parameters were 'entropy' for the splitting measure, 'auto' for the function of number of maximum features, maximum depth was 40 and the ccp_alpha value was 0.01.

An ensemble learning technique, the random forest is implemented using the inbuilt function[7] in python provided in the scikit library. The hyper-parameter tuning is done over the number of trees in the forest whose values were 30, 50, 100, 200; the splitting measures, namely gini and entropy; the maximum depth of the tree whose values were 10 and 20; and finally, the function of number of maximum features to consider when splitting the data, namely auto, sqrt and log2. The parameters after tuning were 'entropy' for the splitting measure, number of trees was 200, and the maximum depth was 10.

The AdaBoost technique is another ensemble learning technique, that is implemented using the inbuilt function[8] in python provided in the scikit library. The hyper-parameter tuning is done over RF, SVM and KNN, the three best performing models, in order to get the best output.

A multi-layer perceptron was implemented using an inbuilt function[9] in python's scikit library. The hyper-parameter tuning was run on the activation functions, namely tanh, logistic and relu. 'logistic' was the activation function that was the result of the hyper-parameter tuning.

2.3 Feature Extraction

Using the results of the PCA and LDA run initially as part of pre-processing, Random Forest was run. The accuracy did not show any significant improvement with it being 0.78 when run on PCA data, and 0.79 when run on the LDA data.

ANOVA was also used to extract the top 30 features from the data set. These 30 features were scaled using the StandardScaler, and Random Forest was used on this data. The resulting accuracy was 0.70.

3 Evaluation Criteria

To evaluate the models, the macro-averaged scores have been considered. Precision quantifies the number of positive class predictions that actually belong to the positive class. Recall quantifies the number of positive class predictions made out of all positive examples in the data set. F-Measure provides a single score that balances both the concerns of precision and recall in one number. The confidence Score is a number between 0 and 1 that represents the probability of an output belonging to the predicted class. The precision, recall and f-measure of the models run ave been summarised in 1.

Classifier	Precision	Recall	F-measure	Confidence Score
Multinomial Bayes	0.53	0.51	0.50	0.21
Logistic Regression	0.72	0.72	0.72	0.62
Support Vector Machine	0.76	0.75	0.75	0.81
K-Nearest Neighbor	0.73	0.72	0.72	0.99
Decision Tree	0.51	0.51	0.51	0.63
Random Forest	0.79	0.79	0.79	0.79
AdaBoost	0.75	0.76	0.76	0.77
Multi-Layer Perceptron	0.73	0.72	0.73	0.99

Table 1: Performance Of Different Classifiers Using All Terms

[7] RandomForestClassifier function
[8] AdaBoostClassifier function
[9] MLPClassifier function

4 Analysis of Results

Owing to the assumption it makes on the independence of the features, it comes with no surprise to the authors that it performs quite terribly. The macro f1 score comes out to be a mere 0.51, and the confidence score is barely even considerable at 0.22. The Decision Tree performs quite poorly on the data as well.

All other models work quite well in classifying the data, as seen from their accuracy scores. The Random Forest gives the best accuracy score of 0.80. However, the confidence scores of KNN and MLP are much better than the other models. Thus, they can be said to perform better than all the other models. While they both have almost the same accuracy scores, we rate MLP to be the best classifier owing to it's higher f-measure score.

We then proceed to see the features with the highest importance in the best models. However, feature importance cannot be retrieved from KNN and MLP. Hence, the top features from KNN and Random Forest are presented below in 3.

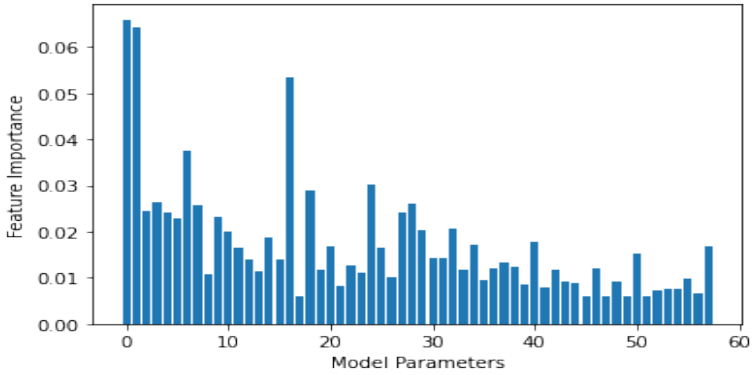


Figure 3: Feature Importance from Random Forest

5 Discussions and Conclusion

From the feature importance calculated before, the top four features are the 'length', 'chroma_stft_mean', 'spectral_centroid_var', 'perceptr_var'. These features can be said to be what distinctly characterise different genres of music. The length corresponds to the length of the wave, and the other features correspond to specific characteristics of the waveform of the sound file. A potential future study could include adding more data to the initial data set. This will give better clarity on the data and hence, make implementations more reliable. Other algorithms can also be tested. This study itself, could be up-scaled in order to create a recommendation system of music.

References

[1] Jaime Ramírez and M. Julia Flores. Machine learning for music genre: multifaceted review and experimentation with audioset. *CoRR*, abs/1911.12618, 2019.

[2] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

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