Music Genre Prediction from Audio, Metadata and Lyric Features Report

Ankita Dhar

1 Introduction

Music is one of the most cherished art forms and source of entertainment from the time immemorial. During times of pandemic, like the COVID-19, music has been one of the most sought-after entertainment; which is why classification of music has various practical implementations. Digital music services like Spotify, Youtube, Amazon music use numerous methods to classify songs into various categories, which includes classification by music genre. Such classifications provide users fluid access to the kind of music they like and thus enhancing the user experience. This paper will discuss how Machine Learning can prove to be effective in classification of music by genres. To have a deep insight on this topic, we will be considering the publicly available [1] **The million song dataset**; this dataset is available for researchers to perform MIR (music information retrieval) without licensing issues. In this paper, various machine learning techniques, like *Naïve Bayes, Multi-Layer Perceptron, Decision Trees, Random Forest* and *Support Vector Machines*, will be compared with respect to their ability to automatically classify music into respective genres. However, use of classification algorithms to classify songs is not enough; a proper pre-processing of data plays a vital role in achieving the desired accuracy of a models. For the list of songs that we consider here, we have a set of various features to use. I hypothesize that proper pre-processing of data can prove to be beneficial in achieving a well-trained model. In an effort to portray the effect of pre-processing of features, we will be discussing a few pre-processing techniques across the above-mentioned models.

2 Literature Review

A lot of work has been done in music information retrieval using Machine Learning methods till date. Among other tasks, classification of music by genre is a problem much delved about. It has been found that *MFCC* is one of the most effective features in speech recognition, and thus has been applied in this task in a hybrid model of Gaussian and k-NN by G. Tzanetakis and P. Cook [2] in 2002. In the following year, year 2003, C. Xu et al [3] had applied SVM to classify music in mainly four genres with as high accuracy as 90%.

Although pre-processing data is a part of each work done to solve this problem, but it is not talked about much. In this paper we will have a look at the importance and requirement of different types of pre-processing based on a model's way of working.

3 Implementation

Music classification using supervised machine learning is considered to be effective ways of classification. This can be justified with the fact that there are millions of songs online and from decades they have been stored and sorted manually. To make use of the age-old sorting information digitally stored is only obvious.

3.1 Dataset

3.1.1 Overview

Our dataset is segregated into 3 parts as:

1) train features with corresponding labels to train our model
2) validate features with corresponding labels to validate our model
3) test features, to which the trained model is to be applied to classify songs.

The three sets of data comprise of various features, namely – *trackID, title, tags, loudness, tempo, time_signature, key, mode, duration, vect_1 to vect_148*.

3.1.2 Pre-processing data

The features *trackID* and *title* are unique to each instance and provide no information about a song, therefore are discarded and not used in any model.
Rest of the features, except *tags*, store numerical value. The tags comprise of numerous words

from the lyrics and is an important attribute because generally genre of a music can be identified from its lyrics.
Pre-processing of *tags*, is one of the most important tasks that needs to be accomplished. Tags is a comma separated long string, which is to be converted into some kind of numerical form, so that it can be used in combination with rest of the features. There are two ways to do so. Firstly, we can convert the tags into vectors using *TfidfVectorizer* from *sklearn.feature_extraction.text* and then sum up the vector values, where *TfidfVectorizer* essentially gives relatively lesser weight frequently used words. Secondly, we can use *MultiLabelBinarizer* from *sklearn.preprocessing* to get set of words in tags as binary-features for all the songs.
Another pre-processing that becomes important for some algorithms is scaling dataset. We would use various scaling techniques; discuss and compare their effects.

3.2 Experimentation (Baseline Approach)

As a baseline experimentation algorithm, we can have performance of *Zero-R* model, which gives an accuracy of 12.22% to be the lower bound for other models.

3.3 Experimenting with different models

Let us consider *Naïve Bayes* approach first. For this experiment we will consider combination of all the features (except *trackID* and *title*. A summary of different pre-processing is listed in the table 3.3.1.
We find that selecting K best tags after application of *MultiLabelBinerizer* yields good results as the lyrics' wordings are used in the best way that could define closeness of a song with an existing song.

Preprocessing Applied	Accuracy
tags preprocessed with TfidfVectorizer	51.11 %
tags preprocessed with MultiLabelBinerizer	59.77 %
Normalize feature using normalize	41.77 %
Scaling features using StandardScaler	40.66 %
Scaling features using RobustScaler from sklearn	42.66 %

Table 3.3.1 - Accuracy of Naïve Bayes with different data pre-processing

However, upon scaling the data, a significant drop is seen in the accuracy. Upon investigating, we might see the issue to be lying in the huge dimensions provided in the datasets, which we often refer to as curse of dimensionality.
From the experiment done with the *Naïve Bayes Model*, we have collected information on the importance of pre-processing data. Let us now consider various models and effect of pre-processing lyrical tags on each model.

3.3.1 Pre-processing the feature "tags"

Pre-processing applied on tag for Decision Tree Algorithm	Accuracy
TfidfVectorizer	36.22 %
MultiLabelBinerizer	42.66 %

Table 3.3.2 - Accuracy of *Decision Tree* with different pre-processing on feature "tags"

In *Decision Trees* method, the data is continuously split, depending on certain parameters. Since such splitting decisions to be made depends upon provided features, good pre-processing makes a lot of difference here, as can be seen from the tabulated data.

Pre-processing applied on tag for Random Forest Algorithm	Accuracy
TfidfVectorizer	45.55 %
MultiLabelBinerizer	59.33 %

Table 3.3.3 - Accuracy of *Random Forest* with different pre-processing on feature "tags"

When we consider *Random Forests*, a similar kind of effect can be seen here. The logic behind this can be easily understood from the fact that *Random Forest* is an ensemble learning method which creates multitude of decision trees during training phase.

Pre-processing applied on tag for Multi-Layer-Perceptron Algorithm	Accuracy
TfidfVectorizer	12.22 %
MultiLabelBinerizer	12.22 %

Table 3.3.4 - Accuracy of *Multi-Layer-Perceptron* with different pre-processing on feature "tags"

Since *Multi-Layer-Perceptron* algorithm provides a non-linear mapping between input vectors and corresponding output vector, a lot of information gets hidden during the process.

Pre-processing applied on tag for SVM Algorithm	Accuracy
TfidfVectorizer	12.22 %
MultiLabelBinerizer	12.22 %

Table 3.3.5 - Accuracy of *SVM* with different pre-processing on feature "tags"

Support Vector Machine has a method called kernel trick which converts non-separable instance labels into separable instance labels, while doing so, the algorithm sometimes might under-perform if the data is not scaled well.

Let us see effect of scaling data on these machine learning methods.

3.3.2 Scaling data

From the previous segment we find pre-processing tags using *MultiLabelBinerizer* gives better output, hence in this section we will be using the same feature set as above with tags processed with *MultiLabelBinerizer*.

Scaling features for Decision Tree Algorithm	Accuracy
MaxAbsScaler, StandardScaler, RobustScaler, MinMaxScaler …	42.66 %
Normalizer, preprocessing.normalize	43.33 %

Table 3.3.6 - Accuracy of *Decision Tree* with different Scaling functions

Upon performing this experiment, we find that scaling has negligible impact on *Decision Tree*'s performance. This observation is in sync with our understanding of working of this algorithm. In this method, there are no serious implications because of rounding of numbers.

Scaling features for Random Forest Algorithm	Accuracy
MaxAbsScaler, MinMaxScaler, StandardScaler, RobustScaler	60.22 %
Normalizer, preprocessing.normalize	62 %

Table 3.3.7 - Accuracy of *Random Forest* with different Scaling functions

Decision Tree and *Random Forest* algorithm both just compare two objects and create branches based on certain criteria. Therefore, scaling is not important here.

Scaling features for Multi-Layer-Perceptron Algorithm	Accuracy
RobustScaler	39.33 %
StandardScaler	41.55 %
Normalizer, preprocessing.normalize	62 %

Table 3.3.8 - Accuracy of *Multi-Layer-Perceptron* with different Scaling functions

On the other hand, *MLP* and *SVM* methodology's performance has been enhanced by a huge margin. The reason behind this jump in accuracy of these two models is the way they work internally. In the broad sense, both of the methods can perform well if all the data are within a range that are comparable to other features.

Scaling features for SVM Algorithm	Accuracy
StandardScaler	65.33 %
MaxAbsScaler	60.66 %
Normalizer	66.22 %

Table 3.3.9 - Accuracy of *SVM* with different Scaling functions

4 Performance Analysis

To gain a better insight let us have a look at other evaluation metrics.

Model	Precision	Recall	F1-score
Naïve Bayes	61.04 %	57.61 %	54.60 %
Decision Tree	41.58 %	40.41 %	40.43 %
Random Forest	70.09 %	57.92 %	56.25 %
MLP	**61.49 %**	**59.41 %**	**59.62 %**
SVM	**73.31 %**	**61.93 %**	**61.00 %**

Table 4.8 – Showing Evaluation Metric for different Machine Learning models

From the above information we find that the F1-scores of *MLP* and *SVM* are highest. From high F1-score we can realize that the models have a high and balanced precision and recall attributes.

For *MLP* it has been found that the model performs the best with tanh activation function. The current parameters used for *MLP* are hidden_layer_sizes as (1000,50), solver as stochastic gradient descent, learning_rate_init as 0.01 and max_itr as 1000; comparing this to the cost involved in configuring *SVM*, which includes kernel parameters, *MLP* would be a better choice.

5 Error Analysis

Understanding learning curves is one of the best way to do error analysis. Below are the learning curves of *MLP* and *SVM* models.

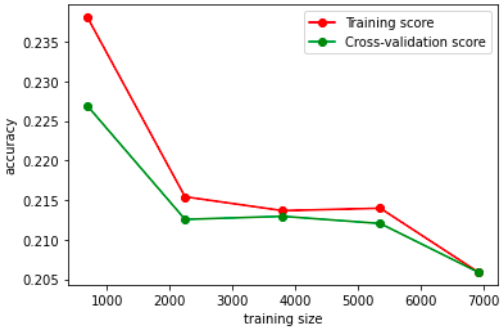

Figure 5.1- Learning curve for Multi-Layer Perceptron

In this stage, we find that the *MLP* model is over-fitting; i.e. it has a high variance and low bias.

In contrast, *SVM* has relatively high bias and low variance. *SVM* model can be said to have under-fitted, however we can say that the model is not too bad. Hence, after error analysis, we would suggest *SVM* as a better choice for the task.

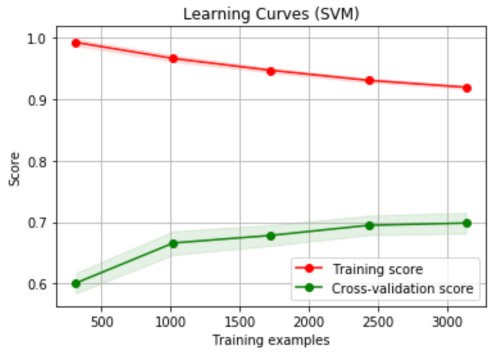

Figure 5.2- Learning curve for SVM

6 Improvements

I. Using *SelectKBest*, we select *130* best tags to do performance testing. We have arrived to this number by trial and error, keeping in mind the time taken to process the data.

II. To reduce variation in the results of *MLP*, a parameter *shuffle* is turned off.

III. *random_state* variable of *Random Forest* is set to *370* to reproduce same value of accuracy.

7 Conclusion and Future Work

Learnings from this experiment gives us enough data points and information to claim that using the right method of pre-processing is an indispensable task to get a well-trained model. We also realized the importance of different preprocessing techniques for different models.

Understanding other preprocessing methods and their importance in classification of nominal and ordinal data can be taken up as future work.

*** At the time when the paper was written, I had very little knowledge of SVM model. In future works, I would like to analyse why did SVM work so well in multi-class classification.*

References

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

A. Schindler and A. Rauber. Capturing the temporal domain in Echonest Features for improved classification effectiveness. In Proceedings of the 10th International Workshop on Adaptive Multimedia Retrieval (AMR) , 2012.

[2] Tzanetakis, G. and Cook, P. (2002). Musical genre classification of audio signals. IEEE Transactions on Speech and Audio Processing, 10(5), pp.293–302.

[3] Changsheng Xu, N. C. Maddage, Xi Shao, Fang Cao, and Qi Tian. Musical genre classification using support vector machines. In 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03)., volume 5, pages V–429, April 2003.