Comparison of learning models for music genre classification

Abstract
Music is categorized into subjective categories called genres. Humans have been the primary tool in attributing genre-tags to songs. Using a machine to automate this classification process is a more complex task. Machine learning excels at deciphering patterns from involuted data. We aimed to compare various machine learning models in context of music genre classification. The training dataset consisted of 4363 songs in total, unevenly distributed across genres (mostly pop-rock). Feature-space comprised of 3 categories, namely pitch, rhythm and timbre, with consideration of statistical subdivisions and bands under each category. Python is used for implementing this project specifically the scikit-learn library. Heat-maps, histograms and confusion-matrices are used to visualise data. In the data-analysis, results and discussions sections we try to emphasize on the aspects of model comparison and end-to-end design process as opposed to solving the songs classification optimally.

Keywords
scikit-learn[3] — heat-maps[4]

Contents

1. Introduction

The project revolves around the task of identifying the music genre of songs.This is useful as a way to group music into categories that can be useful for recommendation or discovery. The problem of music genre classification is difficult. Though some genres can be distinguished in a fairly straightforward procedure (e.g. heavy metal vs classical), others are fuzzier (*e.g.* rock vs blues).

In this analysis project, we constructed a predictor $h(x)$ for each genre Y, which takes the features x and maps it to a probability $h(x)$ that the genre is "rock"(e.g.) or not. We experimented with different machine learning methods, for predicting the music genre of songs. The dataset which is provided for this task contains pre-processed audio information. In particular, the raw audio signals have been transformed to carefully chosen features.

The data set (song features, genres) is detailed in the data analysis section. Before undertaking this we wanted to:

- explore practically various models (classifiers) we learned in theory
- understand the end-to-end process of machine learning
- get comfortable with the available libraries
- learn how to contextually visualise/analyse relationships between data

During the course of the project we were able to perceive many subtleties involved with building models. One such significant aspect is the importance of knowledge on feature extraction process.

Though there are methods for inspecting and studying the obtained data, we figured that the basis on which this procurement happens would certainly influence every minute decision made throughout the design process.

The effects due to lack of this major information availability is apparent in the metrics detailed in the further sections.

2. Data Analysis

Before diving deep into applying some machine learning algorithms it is useful to explore the data a bit to get a general idea of the features and quirks of the data set. Initially we only knew that the datasets consists of 4363 songs which are divided into 264 features. Each song is assigned to a class that corresponds to a music genre. Our data analysis process began by printing the values for some randomly selected data points. The raw data does not make much sense and it is challenging to try to find some correlation between certain classes and some selected features. However, in that small sample a disproportionate share of the labels were *Pop_Rock* or class 1. The next logical step was to take a closer look at the class distribution of the given data and see if the observed unbalanced distribution is just randomness or a feature of the data set. The class distribution has been visualized in figure 1 and the dominance of class 1 can be clearly seen. This imbalance might be due to having two popular music genres, pop and rock, under one label but it can also be just a feature of the data set. Intuitively the disproportionate class distribution can be justified but for practical applications that have to work on any unlabeled data set, such assumptions cannot be made.

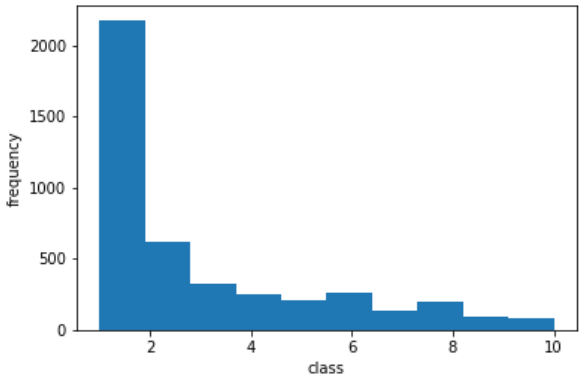


Figure 1. Distribution of classes in the data set

Another interesting feature of the data set is the low number of samples with relation to the amount of features. Since there are 4363 songs and each of them have 264 features, it is obvious that the dataset is nowhere near sufficient to explore all combinations of all features. Most likely the whole space is not needed to represent all common music genres but in spite of this, the size of the data set might be the limiting factor. We could overcome this by finding a subset of the original features that preserve the information of 264 features. Since the feature space includes multiple statistics of rhythm and chroma, there could be some correlation that could be exploited to reduce the dimensionality. The correlation heat-map for the training data in figure 2 shows that there is indeed some correlation and especially the seven rhythm statistics are nicely separable.

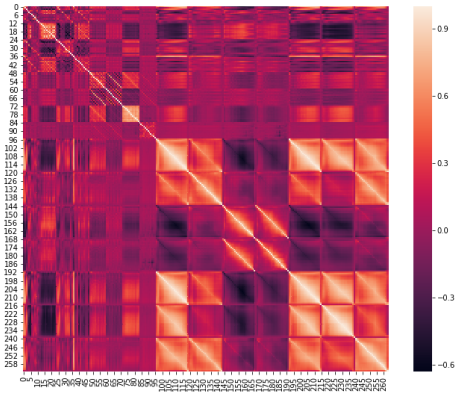


Figure 2. Correlation heatmap

The heatmap is a nice visualization but it would be hard to manually engineer a new feature space that captures the details that matter. For the actual dimensionality reduction task principal component analysis is a much better candidate. The variance explained by PCA is illustrated in figure 3. The figure suggests that the first few principal components explains the majority of the variance and with 10 principal components almost all the variance is explained. The heatmap also indicates that there could be some redundant features so a more interpretable dimensionality reduction technique would be to just discard features that do not have sufficient contribution to the total variance.

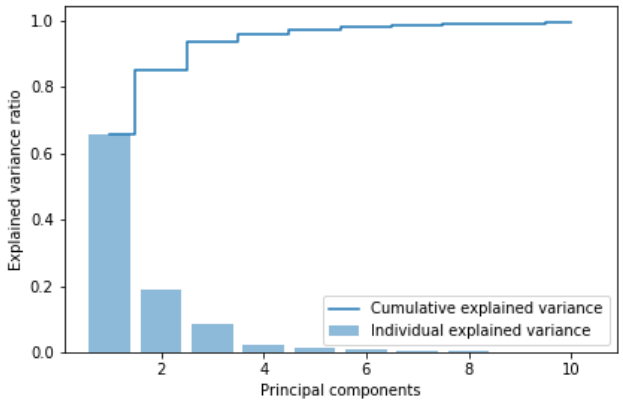


Figure 3. Variance explained by principal components

3. Methods and Experiments

In order to experiment with different ideas and approaches it is important to set up a machine learning pipeline. Such pipeline should take the raw data as input and

produce an accurate description of the quality of the underlying classifier and the classification output when desired. Ideally this pipeline can easily be used to test different classifiers and also to find sufficient hyper-parameters for them. Another desirable qualities are speed and accurate estimations of the performance. This section describes how we set up our pipeline and what were the most promising approaches we came up with.

3.1 Pipeline and evaluation methodology

We used pandas framework to read and write the csv-files and relied heavily on scikit-learn to build a robust machine learning pipeline that could effortlessly be used to try different preprocessing steps and machine learning algorithms with sufficient hyper-parameters. An example of such pipeline using logistic regression classifier is provided in appendix 6.1

As was mentioned in the previous section, the set of labelled data points is relatively small when considering the classification task. Therefore we decided to use cross-validation instead of a separate validation set. To be more specific, we used stratified k-fold that takes into account the unbalanced class weights. There is no single configuration which we used since we had to do some modifications in order to find the right balance between variance and performance. One of these configurations, for example, was to use 5x3 cross-validation where the data was split into five folds and then the data of the four training folds was once again split into three folds out of which two were used to train the model and one was used to optimize the hyper-parameters. This approach resulted in good hyper-parameter values and an accurate estimation of the performance.

All in all, our pipeline consists of reading the csv file, splitting the data into appropriate training and validation sets, standardizing the data, possibly reducing the dimensionality and then training and tuning a classifier. The tuned classifier is then either described by some performance metrics or it is used to label the unlabelled data. We relied heavily on this pipeline to evaluate the performance of a classifier instead of submitting the results of every prominent classifier to the Kaggle competition. As a result we might have overlooked some classifier that has been rated against an undesirable or unrepresentative validation set. However, this approach is more likely to work with a real-life machine learning problem.

3.2 Performance Metrics

Our goal is to accurately predict the genre of a song but also to have highest possible confidence in our prediction. For the first goal a single label must be predicted for each unlabeled song and the performance is measured by categorical accuracy which is illustrated in equation 1.

$$\text{accuracy} = \frac{|y_{\text{true}} = y_{\text{predicted}}|}{N} \tag{1}$$

In order to measure the confidence of a prediction the output must have ten predictions where ith prediction corresponds to the probability of the data point belonging to the ith category. The classification cost can then be computed using equation 2.

$$\text{logloss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j}) \tag{2}$$

It should be noted that in the following subsection we will be optimizing the classifier so that it will perform well on both of these tasks.

3.3 Algorithms and methods

Fundamental intuition for choosing models: Multi-class classification being the underlying aspect of the problem statement, we turned to learning models which constitute this feature.

3.3.1 Logistic Regression

Motivation: Logistic regression though by nature the default for binary classification, the fact that it can be extended for multi-class models was the motivation for choosing it.

Our model predicted the probabilities of different possible genres (labels) corresponding to a given feature space (see the data analysis section). For each of the multi-classification modelling techniques [one-vs-one - *ovo*, one-vs-all - *ovr*], we empirically used various algorithms as solvers (like stochastic average gradient) and consecutively varied the following parameters.

- Maximum iterations for convergence $(100-900)$
- Initial class weights (balanced weighing, initializing to 1)
- Weight regularization strength $(0.1-0.9)$
- Tolerance of error across runs

in focus of improving on the accuracy of and reducing loss of the classification.

To optimize the hyper-parameters for modelling the logistic regression classifier we made use of the exhaustive Grid-search cross validation technique.

3.3.2 Support Vector Machines

Motivation: As the feature-space is of a relatively higher dimensionality, we decided to try out support vector machines to improve on the statistics achieved using the logistic model.

Initially, we modelled a linear support vector classifier which used set of hyperplanes for learning the training data. As we did not observe any drastic improvements in the statistics, we further analysed the data and decided to experiment with the non-linear modelling of the support vector machines.

While empirically using both *ovo* and the *ovr* techniques, we tested various kernels:

- Gaussian radial basis function
- Polynomial (degree 3)
- Sigmoid

For each of the kernels we varied the following parameters:

- Maximum iterations for convergence $(100-300)$
- Initial class weights (balanced weighing, initializing to 1)
- Weight regularization strength $(0.1-0.9)$
- Tolerance of error across runs

To optimize the hyper-parameters for modelling the support vector classifier we made use of the exhaustive Grid-search cross validation technique.

3.3.3 Ensemble Classifier

Motivation: After experimenting with the above two models for sometime, we figured that logistic regression classifiers were good with accuracy metric and SVMs were minimizing on the log-loss statistic. So we decided to ensemble them together to understand if we can get the better of both worlds.

We used the **voting classifier** strategy to combine the above classifiers using a majority vote to predict the genres. We used both hard and the average predicted probabilities voting (soft vote) methods and found that the hard voting produced better results.

3.3.4 Semi-Supervised Classifier

Motivation: After not being able to further deduce any meaningful direct relationship between the feature-space, we decided to try the semi-supervised learning methodology.

We chose the label-spreading strategy over label propagation as it is more robust to noise[1]. Modelling steps:

Step 1 Splitting the labelled (train) data as based on some percentage (we chose $\frac{1}{3}^{rd}$ of the it for validation)

Step 2 Merging the split of labelled data from [step-1] with the unlabelled data.

Step 3 Training the classifier/kernel chosen for label-spreading with respective parameters (listed below)

Step 4 Scoring/validating the trained model using the split test data from [step-1]

Step 5 Predicting the genres for unlabelled (actual test) data

We varied the following parameters over runs:

- Kernels (rbf, k-nearest neighbours). Varying the gamma parameter (b/w $0.1-00001$) and the number of neighbours attribute (b/w $50-300$) respectively.
- Maximum iterations for convergence $(100-300)$

Other models

Apart from the above mentioned classifiers, we experimented with more models that did not pass our benchmarks for both the accuracy and loss metrics.

- Naive Bayes'
- Decision trees
- Random forests

Though we initially considered reducing the dimensionality by performing principal component analysis, after some substantial scrutiny of the feature space we dropped it. The rhythm bands, pitch classes and the timbre coefficients we spread out across the statistics to discard.

4. Results

Performance measures of the above experiments when evaluated with the test data. Eventual Kaggle testing accuracy is 0.653 and log-loss of 0.175

4.1 Accuracy, Log-Loss statistics

- Logistic Regression Classifier across various values for its parameters. Results for the best 5 combinations:

Table 1. metrics on test data (LR)

modelling strategy	solver	accuracy	loss
ovr	sag	0.653	0.175
multinomial	sag	0.653	0.177
ovr	liblinear	0.650	0.21
ovr	lbfgs	0.646	0.26
ovr	newton-cg	0.646	0.26

- Support vector classifiers across various values for its parameters. Results for the best 5 combinations:

Table 2. metrics on test data (SVC)

modelling strategy	kernel	accuracy	loss
multinomial	rbf	0.636	0.27
ovr	rbf	0.636	0.19
multinomial	linear	0.607	0.20
ovr	linear	0.607	0.20
ovr	poly (degree 3)	0.567	0.36

- Ensemble classifier with best parametric combination of logistic regression, SVC and voting strategy produces

$$accuracy = 0.631, loss = 0.29$$

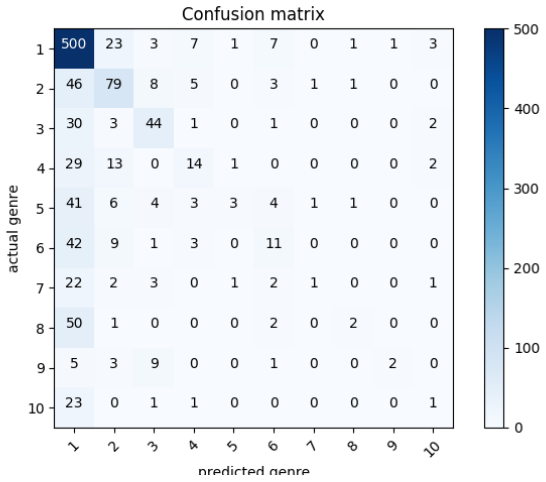
- Semi-supervised classifier with label spreading:

Table 3. metrics on test data (semi-supervised)

kernel	attribute	accuracy	loss
knn	neighbours(300)	0.533	0.188
rbf	$\gamma = 0.001$	0.612	0.29

4.2 Confusion matrix

The following confusion matrix represents the accuracy of classification. It is evident that the prediction is directly influenced by the frequency of respective genres in the training data.



The matrix entry i, j is the number of songs actually in genre i, but predicted to be genre j. Darker the blue, accurate the classification.

5. Discussion

As the data analysis section illustrated, we have been working with data that has unequal class distribution: the amount of pop/rock songs is prominent. Since the sample of labelled data is relatively small and we do not know how the training data or the testing data is sampled there is no reason to strongly believe that class distribution of the training data generalizes to the testing data. Also our experiments with the data advocate this view since both the accuracy and logistic loss classifiers had better performance with uniform class distribution than with the class distribution of the training data.

The biased class distribution, however, imposes a problem on model training. The space of class 1 songs is known well and therefore it is likely to be classified as class 1 song unless it matches the specific characteristics of songs with some other label. This phenomena can be seen in the confusion matrix: a high percentage of songs not belonging to class 1 are labelled as belonging to class 1. In context of logloss this is equal to overestimating the probability of a song belonging to class 1. With logloss performance metric the negative effect is even higher since it is not just the highest accuracy that matters but every percent in the probability is counted in the cost function. There is no such misclassification with any other class and we see a lot of space for improvement here.

Another interesting detail is the behaviour of the ensemble model. Our initial belief was that the ensemble model should be more robust to overfitting and also perform similarly or better in the classification task than the original classifiers. This belief is also supported by [5]. Against our expectations the ensemble classifier did not perform better than the individual classifiers. This could be due insufficient diversity between classifiers or our decision to rely heavily on the performance metrics obtained from cross-validation set. If the class distribution of the testing data is different from the distribution of the training set then the ensemble could actually be better than the individual classifiers.

If we were to redo this project we would probably tackle the problem of unbalanced class distribution and do some further research on semi-supervised learning which seems suitable approach. We could also focus on building more diverse and robust ensemble classifier and make better use of the possibility to test the classifier on data that is used to measure the performance.

6. Appendices

6.1 Pipeline using scikit-learn

The following code snippet illustrates how easy it is to set up a machine learning pipeline in scikit-learn assuming *load_data* is a function that loads the data set. *LogisticRegressionCV* is a class that trains a logistic regres-

sion classifier using all the listed C-values and uses the best-performing classifier to calculate accuracy on the test set. A similar class that can be used to optimize hyper-parameters in scikit-learn is *GridSearchCV* [2].

```
from sklearn.cross_validation
    import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model
    import LogisticRegressionCV

X, y = load_data()

X_train, X_test, y_train, y_test =
    train_test_split(X, y)

classifier = Pipeline([
    ('std', StandardScaler()),
    ('pca', PCA(n_components=100)),
    ('lrcv', LogisticRegressionCV(
        Cs=[0.1, 0.2, 0.5, 0.8, 1, 2, 5]))
])

classifier.fit(X_train, y_train)
accuracy = classifier.score(X_test, y_test)

print(accuracy)
```

References

[1] semi-supervised learning,

[2] GridSearchCV,

[3] scikit-learn library,

[4] heat-maps,

[5] Thomas G. Dietterich. Ensemble Methods in Machine Learning. International Workshoop on Multiple Classifier Systems, 2000