Music Artist & Genre Identification using Machine Learning Techniques *

Johnny Sellers †

April 1, 2019

Abstract

This report covers an experiment in classifying music audio samples using feature extraction and machine learning techniques. Audio signal data was assembled and Principle Component Analysis was used to extract the features of the samples' spectrogram data and provided the necessary separability for the classification algorithms to produce good results. This method produced accuracies of 96.1% for some models.

1 Introduction and Overview

Identification or classification of music audio data involves the use of signal processing techniques and statistical methods. The thrust of the problem is to extract features having the greatest separability from the data for the machine learning algorithm to train on. The tool used to do so in this experiment was the *singular value decomposition* (SVD). The component matrices of the SVD and their interpretation conveyed insights into the data structure that were used to exploit the derived patterns for better machine learning model performance. Generally, more pronounced features of the data produce better classification results from the machine learning models. [5] Hence the invaluable dimension-reducing, feature extracting power of the SVD in such tasks.

Feature spaces of the data were obtained through examination of the low-dimensional components of the SVD. The singular value spectrum and histogram of principle modes given by the right singular vectors of each sample was visually inspected to determine which modes yielded the sought-after separability. This identification procedure could be replaced with an automated approach based on computing the energy carried by the singular values.

Three tests were carried out in this experiment: (1) for three artists, and three songs per artist, thirty 5-second song samples were taken from each song and assembled into a labeled data set to classify the samples by artist. (2) For three bands of the same genre, and one song per band, fifty 5-second audio samples were taken to produce a classification

*This paper is an extension of a report originally written for Dr. J.N. Kutz's course, *AMATH 582: Computational Methods for Data Analysis*, in the Winter 2019 quarter at the University of Washington, Seattle, Wa.

†Graduate student, Applied Mathematics Department, University of Washington, Seattle, WA

for the samples by band. (3) For three music genres, and three artists from within each genre, thirty 5-second samples were taken from each song and used to produce a classifier for the samples by genre. The number of samples and their lengths were decided on with the computational time as the primary concern; accuracy was next in importance for experiment design criterion. The tools used in the second experiment were from the Python open-source machine learning library, scikit-learn [8]. The Fast Fourier transform methods of Python's Numpy library [6] were also used to transform the audio samples to wavelet space in order to better extract features.

2 Theoretical Background

2.1 Feature Engineering and SVD Interpretation

Machine learning tasks rely heavily on feature engineering and feature extraction algorithms for accuracy. Features that show strong separability or variance in the data should be those exploited for machine learning tasks. This is because they help models to identify highly distinguishable features in higher-dimensional space, leading to accurate classification, clustering and prediction. [5] Analysis and interpretation of the SVD of the assembled data matrix were the primary means for extracting a feature space from the audio sample data.

The data matrix X was formed from assembling the spectrograms of the samples reshaped into column vectors (see §3). Then the SVD of X was given by

$$X = U\Sigma V^*. \tag{1}$$

From (1) the intrinsic, low-rank feature space was produced. The matrix V^* holds information about the importance of the features of the audio samples. In particular, each column of V shows the unique loading or weighting of each feature onto a specific sample. [5] The histograms in Figure 4 of §4 illustrate these unique loadings. The feature modes (i.e., columns of V^*) with higher variation are apparent from the misalignment of the curves; they show best the separation of the different classes. The modes with greater separation provide a feature space that is easy for classifiers to generate data boundaries. The matrix U is analogous to the "coordinate system" of the feature space; it provides a means to measure separation. The singular value matrix, Σ, is the weight or measure of importance to the direction of the feature loadings in the columns of V. Together, these insights are essential for generating a good feature space for machine learning algorithms to exploit.

2.2 The Supervised Learning Problem

For a mathematical statement of supervised learning, let $\mathcal{D} \subset \mathbb{R}^n$, where $\mathcal{D}$ is an open bounded set of dimension n. Now let $\mathcal{D}' \subset \mathcal{D}$. Then the aim of classification is to label all data in $\mathcal{D}$ given the data in $\mathcal{D}'$. Each of the three tests in this experiment is a ternary classification problem. Therefore the inputs and outputs for the learning task can be stated

as

Input

$$\text{data} \quad \{x_j \in \mathbb{R}^n,\ j \in \mathcal{Z} := \{1, 2, ..., m\}\} \tag{2a}$$

$$\text{labels} \quad \{y_j \in \{-1, 0, 1\},\ j \in \mathcal{Z}' \subset \mathcal{Z}\} \tag{2b}$$

Output

$$\text{labels} \quad \{y_j \in \{-1, 0, 1\},\ j \in \mathcal{Z}\}, \tag{2c}$$

where the labels $\{-1, 0, 1\}$ represent a band (tests 1 and 2) or a genre (test 3). [5] Then the training data $x_j \in \mathbb{R}^n$ for $j = 1, 2, ..., m$, is comprised of the extractions from the feature engineering process described in §2.1.

2.3 Naive Bayes Classifier

A *generative classifier* has the form

$$p(y = c|\mathbf{x}, \theta) = \frac{p(y = c|\theta)p(\mathbf{x}|y = c, \theta)}{\sum_{c'} p(y = c'|\theta)p(\mathbf{x}|y = c', \theta)}, \tag{3}$$

and it is called so because it specifies how to generate data using the *class conditional density* $p(\mathbf{x}|y = c)$ and the class prior $p(y = c)$. The generative approach requires specifying the class conditional density, and the form of the specification depends on the type of the features. For real-valued features, the model assumption is given by

$$p(\mathbf{x}|y = c, \theta) = \prod_{j=1}^{D} \mathcal{N}(x_j|\mu_{jc}, \sigma_{jc}^2), \tag{4}$$

where μ_{jc} is the mean of feature j in class c, and σ_{jc}^2 is its variance. If the features are assumed to be conditionally independent given the class label, then the class conditional density can be written as a product of one-dimensional densities:

$$p(\mathbf{x}|y = c, \theta) = \prod_{j=1}^{D} p(x_j|y = c, \theta_{jc}). \tag{5}$$

The resulting model is called the naive Bayes classifier (NBC). NBC models have $O(CD)$ parameters, where C is the number of classes and D the number of features, so they are relatively immune to overfitting. [4]

2.4 Random Forest

Ensemble learning is an approach that combines multiple methods to a learning problem. The idea is that various methods will learn different things about the data than others, so combining them will result in a more robust model. This fundamental idea is the basis

for the *random forests* technique, which combines the results of numerous *decision trees*, or *classification and regression trees* (CARTs).

A CART forms a prediction by learning useful feature directly from the data. They are based on the *adaptive basis-function model* (ABM), of the form

$$f(x) = \omega_0 + \sum_{m=1}^{M} \omega_m \phi_m(x), \tag{6}$$

where $\phi_m(x)$ is the mth basis function, which is learned from the data. This model is not linear in the parameters. [4]

CARTs are defined by recursively partitioning the input space and defining a local model in each resulting region of input space. This is represented byt he tree with one leaf per region. Computing the optimal partitioning of the data is NP-complete. A greedy procedure is commonly used to compute a locally optimal maximum likelihood estimation (MLE).

Classification cost is used to measure the quality of the proposed split. Quality is measured The *splitting function* is defined for choosing the best feature and the best value for that feature by

$$(j^*, t^*) = \arg \min_{j \in \{1,\ldots,D\}} \min \text{cost}(\{\mathbf{x_i}, y_i : x_{ij} \leq t\}) + \text{cost}(\{\mathbf{x}_i, y_i : x_{ij} > t\}), \tag{7}$$

where t is a thresholding value, and D is the number of features. The class-conditional probabilities are estimated by fitting a multinoulli model tot he data in the leaf that satisfy the test $\mathbf{X}_j < t$:

$$\hat{\pi}_c = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathcal{I}(y_i = c), \tag{8}$$

where $\mathcal{D}$ is the data in the leaf. There are several error measures for evaluating a proposed partition−misclassification rate, entropy, and the Gini index. The Gini index is given by

$$\sum_{c=1}^{C} \hat{\pi}_c(1 - \hat{\pi}_c) = 1 = \sum_c \hat{\pi}_c^2, \tag{9}$$

where $\hat{\pi}_c$ is the probability a random entry in the leaf belongs to class c and $(1 - \hat{\pi}_c)$ is the probability of a misclassification. [4]

Overfitting can occur if the error does not decrease fast enough. The standard approach is to grow a full tree then perform *pruning*. This means removing branches that give the smallest decrease in error. Murphy in [4] outlines a method picking a tree whose cross-validation is within one standard deviation of the minimum.

CARTs alone may not produce accurate predictions due to the greed nature of their construction. Also that can be unstable, i.e., small changes to the input data can vastly affect the tree structure. Hence the trees are referred to as "high-variance estimators". The random forests method reduces the variance (instability) of an estimate by averaging estimates of many trees.

Bagging refers to training, for example, M different trees on randomly chosen subsets of the data (with replacement) and then computing the ensemble

$$f(\mathbf{x}) = \sum_{m=1}^{M} f_m(\mathbf{x}), \tag{10}$$

where f_m is the mth tree. Bagging stands for "bootstrap aggregating". Re-running the same learning algorithm on different subsets of data can lead to highly correlated predictors, limiting the amount of variance reduction. The random forests method attempts to decorrelate the base learners by learning trees based on a randomly chosen subset of input variables, as well as a randomly chose subset of data cases.

2.5 Model Evaluation with K-Fold Cross-Validation and the Bootstrap

Resampling methods of *cross validation* and *bootstrap* were used to evaluate the models' performances in classification. The true error rate, i.e., the error rate of the classifier when tested on the entire population. Since the amount data available is limited, the resampling techniques were necessary.

These methods each train and test the classifier on a variation of the available data so that problems such as overfitting on the training data and overly optimistic error rate estimates. The overfitting problem is associated with high variance in the model's output; it causes the model to be weak at generalization to new data. (Overfitting is more pronounced with models having a large number of parameters.)

The following techniques allow for the every example of the available data set to be used for training and testing. The diagram in fig. 1 depicts the procedure used in *k-fold cross-validation*. The training and testing sets are varied for k experiments, or k folds, and the average error rate of the folds, E_i, $i = 1, 2, ..., k$, is taken to be the true error E estimate, i.e.,

$$E = \frac{1}{k} \sum_{i=1}^{k} E_i. \tag{11}$$

With an increasing number of folds the variance (and computational time) of the true error rate estimator will increase, and its bias will decrease; and vice versa when the number of folds decreases. Kohavi in [3] recommends, and it is common practice now, to use $k = 10$ *stratified k-fold cross-validation*.

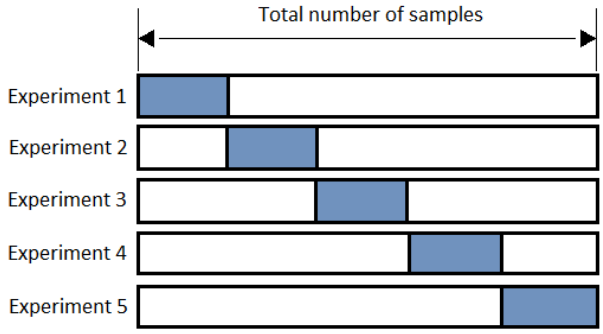

Figure 1: Diagram depicting the K-Fold cross-validation procedure. The blue-colored areas represent the portion of the available data that is take as the test set; the white area represents the training data portion. Here $k = 5$.

In stratified k-fold cross-validation, the folds are chosen so that each one contains an equally distributed number of observations from each class. This method ensures a good representation of the data in every fold. Kohavi in [3] concluded that "stratification is generally a better scheme, both in terms of bias and variance, when compared to regular cross-validation."

Bootstrapping is a resampling technique where N randomly selected examples of the data are selected and used for training while the remaining examples are then used for testing. This selection is repeated for each of k folds, and the value of N is likely to change between the folds. The true error is estimated as the average of the error rate on the test examples.

According to Efron and Tibshirani in [1], the bootstrap increases the variance that can occur in each fold, more realistically simulating the real-life experiment from which the data was obtained. The replacement of the bootstrap resampling preserves the a priori probabilities of the classes throughout the selection process since $p(c_i) = N_i/N$, the probability of selecting class c_i for all i. The bootstrap is able to obtain accurate measures of both the bias and variance of the true error estimate.

3 Algorithm Implementation and Development

The data pre-processing step involved converting the `Mp3` files to the larger `Wav` format to read the signals using the Python `soundfile` package. The method `soundfile.read()` extracts a left and right stereo signals, so the single signal used in processing was taken as the average of these two.

In Test 1, 30 five-second samples were taken from each song; in Test 2, 50 five-second samples from each song; and in Test 3, 40 five-second samples. In each test, the collection of five-second samples, each sample being a one-dimensional vector with length $n = 220,500$, were assembled into a n by m matrix X, where m was the total number of samples. The spectrograms were computed from each column of X and then used to construct a matrix S whose right singular vectors were used as the features to train the classifiers. Figure 2 below shows the Python code for the `svd_spectrogram()` from [7] which programs the process.

```python
def svd_spectrogram(data):
    Fs = 44100  # Expected sample rate for all songs
    Spec_matrix = np.zeros((126936, data.shape[1])) # (129*984, X.shape[1])
    for j, S in enumerate(data.T):
        _, _, Sxx = signal.spectrogram(S, Fs, window=('gaussian', 7),
                                       return_onesided=True,
                                       mode='magnitude',
                                       scaling='spectrum')
        Spec_matrix[:,j] = Sxx.flatten()
    subsample_size = 2**2
    Spec_matrix = Spec_matrix[::subsample_size][:]
    Spec_matrix = Spec_matrix - np.mean(Spec_matrix, axis=0)
    u, s, vh = np.linalg.svd(Spec_matrix)
    return u, s, vh
```

Figure 2: Python function `svd_spectrogram()`. Returns the singular value decomposition of the matrix constructed from the spectrograms of the audio samples.

Figure 3 below shows the function `histogram_separability()` which takes the right singular vectors of the matrix S as a parameter and computes a measure of separability (statistical dispersion) between the class modes. The function ranks the modes (features) with the greatest separability—i.e., the high-variance features—so these can be used to train the classifiers. Four measures of separability were compared—the sum of absolute differences (SAD), the sum of squared differences (SSD), the Pearson correlation coefficient (PCC), and mutual information (MI).

In the file `main.py` of [7] the matrix S and a list of the ranked histogram modes were used to train and evaluate modes for a number of modes used. One resultant plot is shown in fig. 4 of §4.

```
1  def histogram_separability(data, measure):
2      devs = {}
3      num_classes = 3
4      xbin = np.linspace(-0.1, 0.1, 21)
5      ptr = int(data.shape[0]/num_classes)
6      for j in range(data.shape[1]):
7          h_vecs = np.zeros((3,len(xbin)-1))
8          for jj in range(num_classes):
9              h, _ = np.histogram(data[jj*ptr:(jj+1)*ptr,j], bins=xbin)
10             h_vecs[jj] = h
11         dm = np.zeros(3)
12         reverse = True
13         if measure == 'SAD':
14             dm[0] = np.sum(np.abs(h_vecs[0] - h_vecs[1]))
15             dm[1] = np.sum(np.abs(h_vecs[0] - h_vecs[2]))
16             dm[2] = np.sum(np.abs(h_vecs[1] - h_vecs[2]))
17         elif measure == 'SSD':
18             dm[0] = np.sum(np.square(h_vecs[0] - h_vecs[1]))
19             dm[1] = np.sum(np.square(h_vecs[0] - h_vecs[2]))
20             dm[2] = np.sum(np.square(h_vecs[1] - h_vecs[2]))
21         elif measure == 'PCC':
22             reverse = False
23             dm[0] = np.corrcoef(np.array((h_vecs[0],h_vecs[1])))[0,1]
24             dm[1] = np.corrcoef(np.array((h_vecs[0],h_vecs[2])))[0,1]
25             dm[2] = np.corrcoef(np.array((h_vecs[1],h_vecs[2])))[0,1]
26         elif measure == 'MIS':
27             dm[0] = mutual_info_score(h_vecs[0],h_vecs[1])
28             dm[1] = mutual_info_score(h_vecs[0],h_vecs[2])
29             dm[2] = mutual_info_score(h_vecs[1],h_vecs[2])
30         devs[j] = sum(dm)
31     sorted_devs = sorted(devs.items(), key=lambda x: x[1], reverse=reverse)
32     return [k for k, v in sorted_devs if not math.isnan(v)]
```

Figure 3: Python code for the `train_and_evaluate()` function. The `sklearn.model_selection.train_test_split()` function is called to break the input data into subsets for training and testing. Then the SVM model is trained on the training data (line 4) and evaluated on the testing data (line 5).

The classifiers used in this experiment were from the Python package *scikit-learn* in [8]. The following methods were evaluated: 1) Decision Tree, 2) Random Forests, 3), AdaBoost, 4) Naive Bayes, 5) Gaussian Process, 6) Linear Discriminant Analysis (LDA), 7) Quadratic Discriminant Analysis (QDA), 8) K-Nearest Neighbors (KNN), 9) Support Vector Machine (SVM), and 10) Multi-layer Perceptron (NN).

4 Computational Results

The total energy carried by the first 150 singular values of 90%, and the first 200 singular values carried 97% of the total energy.

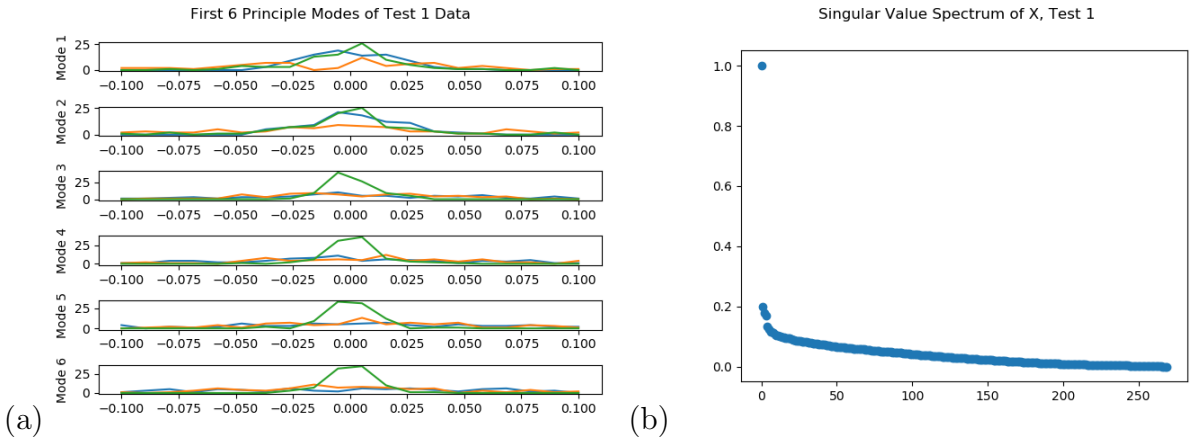

Figure 4: Histogram (a) and singular value spectrum of data matrix (b) in Test 1.

Figure 5(a) below shows the histograms for modes 234 to 239. The rankings of their dispersion measures were 196, 212, 263, 256, 266, and 269 in order from mode 234 to 239. By visual inspection of fig. 5 it was concluded that the mutual information separability measure was an accurate representation of the metric. Figure 5(b) shows more separation in modes 1 and 2 than observed in fig. 5(a).

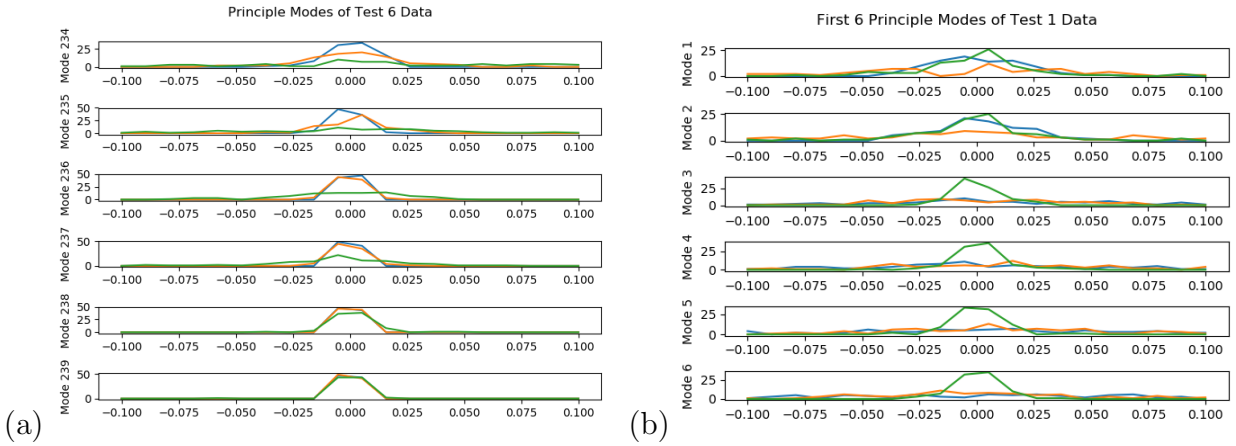

Figure 5: Histogram with relatively small separation (a) and with high separation (e.g., mode 2) (b).

Figure 6 below shows the model accuracies achieved versus the number of modes taken in the training data. The measure of dispersion used to rank the histogram modes was mutual information. The top performing classifier was Random Forests; however, it took a large percentage of the data to train the model up to this accuracy. Table 4 below shows the highest accuracies achieved by the models on the number of modes taken in the training data for Test 1 (artist classification).

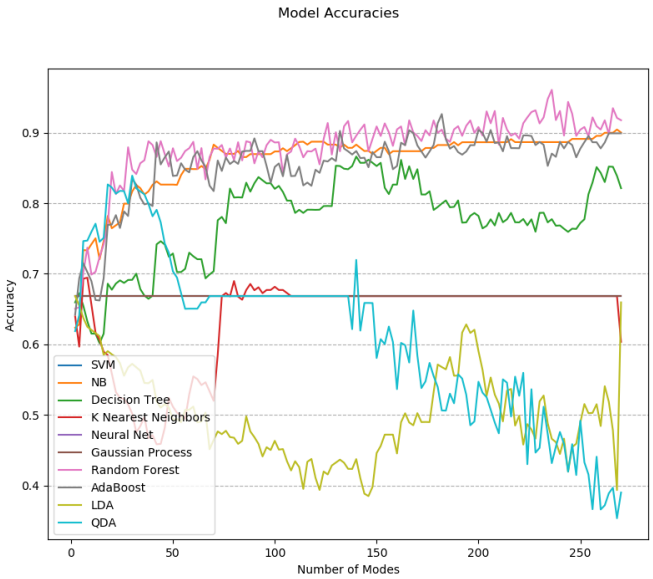

Figure 6: Model accuracies versus number of modes taken in training data. The modes were ranked using the mutual information dispersion measure.

Table 1: Classifier performances with stratified K-fold cross-evaluation and mutual information separability measure. The "Number of Modes" column corresponds to the number of modes used when the maximum accuracy of the model was achieved.

Classifier	Accuracy %	Number of Modes
Random Forests	96.1	117
AdaBoost	92.6	90
Naive Bayes	90.4	133
Decision Tree	86.6	69
QDA	83.9	14
KNN	69.4	3
SVM	66.8	2
NN	66.8	2
Gaussian Process	66.8	2
LDA	68.8	2

5 Summary and Conclusions

References

[1] B. Efron, R. Tibshirani, *An Introduction to the Bootstrap*, International Conference on Artificial Intelligence (1993).

[2] J. Friedman et al., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer (2008).

[3] R. Kohavi, *A Study of Cross-Validation and Bootstrap for Accuracy Estimatio and Model Selection*, International Conference on Artificial Intelligence (1995).

[4] K.P. Murphy, *Machine Learning: A Probabilistic Perspective*, MIT Press (2012).

[5] J. N. Kutz, *AMATH 582 Course Notes*, (Univ. of Wash., Seattle, 2019).

[6] *Numpy Documentation*, available at `https://docs.scipy.org/doc/numpy/reference/`.

[7] J. Sellers, *Music Classification GitHub Repository*, available at `https://github.com/JohnSell620/music-classification`.

[8] F. Pedregosa et. al., *Scikit-learn: Machine Learning in Python*, available at JMLR 12 (2011), pp. 2825-2830.

[9] *Soundfile Documentation*, available at `https://pysoundfile.readthedocs.io/en/0.9.0/#module-soundfile`.