Automated Genre Classification of Musical Signals

Peter Wills & Kathleen Finlinson

December 15, 2015

1 Introduction

Automating the process of categorization and classification is fundamental to artificial intelligence and machine learning. Of particular interest are problems which require distinctions and discernments made easily by a human mind but which are challenging for a computer. Speech recognition and natural language processing stand as classic examples of problems involving extracting extremely abstract, high-level content from an audio signal. A closely related problem is that of musical analysis. The human ear can often quite easily assess qualities of a piece such as mood, intensity, genre, instrumentation, etc, but it is quite difficult to train a computer to reliably and accurately perform this type of analysis.

The purpose of this project is to design an algorithm which can discern the genre of a musical track. The problem as we pose it is quite limited in scope. Our algorithm chooses from a list of five possible genres: classical, electronic, jazz/blues, rock/pop, and metal/punk. The basic approach used is that of statistical learning. The severely restricted scope of this problem allows us to explore different approaches given time constraints and limited training data.

It is worth noting that the problem as originally posed included a sixth genre, so-called "world" music. This category is culturally troubling for a variety of reasons, which we will not delve into here; however, it also poses challenges to any algorithmic classifier. And this is to be expected; world music is defined as the complement of Western music, and therefore has no unifying acoustic characteristics. World music can sound, to an untrained ear, like anything from classical to folk to electronic to jazz, depending on the culture out of which it arises. For this reason, we exclude it from our analysis. The consequence of this, obviously, is that no track will be categorized as "world" music. However, given that this is a genre that is best discarded, we take this as a strength rather than a weakness.

This report is structured as follows. First, we will discuss methods for reducing the dimensionality of the problem, which include randomized methods, network-based approaches, and feature extraction. Then, we will give a brief introduction to the two primary statistical learning algorithms used, naive Bayes and support vector classification. We will report results for each method, and conclude with a discussion of improvements and extensions toward a useful, robust classifier.

2 Preprocessing

The data as initially given to us is in the form of a `.wav` file, a representation of the oscillations of the pressure field in air which generates sound. This form, however, does not map well to the human experience of sound. Humans hear sound in frequency space, essentially performing an internal Fourier transform and discerning the coefficients of different frequencies. Unlike a typical Fourier transform, however, humans discern frequency on a logarithmic scale: a musical octave is a doubling of frequency. Thus we would hope to perform some kind of frequency transform with logarithmic binning.

The mel-frquency cepstrum is a representation of a signal in just this form. To implement the transform, we take a Fourier transform of (a windowed segment of) the signal, then convolve it with logarithmically spaces triangular filters. We then perform a final cosine transform to end up with a set of coefficients, called the mel-frequency cepstrum coefficients ("MFCCs" hereafter). The details of our implementation of this transform can be found in Appendix A. We end up with a time series of roughly 1000 points in $\mathbb{R}^{30}$.

It is worth noting that this is not a necessary first step. We will discuss some methods of feature extraction which operate directly on the signal, or on the vanilla Fourier transform of the signal, rather than the mel-frequency cepstrum. However, we find the mel-frequency cepstrum to be very useful, and it is our primary preprocessing tool.

2.1 Normalization

It is worth noting that we have normalized the MFCCs used by our classifier, meaning that we divide by the total l_2 norm, taken by flattening the track to be a point in $\mathbb{R}^{30,000}$. This introduces additional challenges, as the norm of a track is in fact a good indicator of genre. In particular, we observe that the norm (i.e. loudness) of a track performs quite accurate binary classification between classical and all other genres. However, if we hope to eventually have tracks input via a microphone as is often the case, the volume of the track will not correlate to the volume of the recording. Thus this quantity will be unreliable, and we must normalize. From here on, all discussed approaches use the normalized MFCCs as their foundation.

3 Dimensionality Reduction

Many emerging problems in modern data science involve data points which live in an extremely high dimension. This becomes problematic, as the geometric properties of high-dimensional space often inhibits the efficacy of standard classification and regression algorithms. In mathematics this phenomenon is given the name "concentration of measure," as is stated in a wonderfully succinct way by Talagrand [1]:

> A random variable which depends (in a "smooth" way) upon many independent variables (but not too much on any of them) is essentially constant.

The challenge is thus to move our data into a lower-dimensional space without losing the structure essential to effective categorization. We will explore three approaches to this problem. The first two, randomized projections and graph embedding, are chosen because they are strongly emphasized in the curriculum of the course and we hope to obtain some hands-on experience working with them. The third, loosely reffered to as "feature extraction," applies specifically to the problem of audio content analysis.

3.1 Random Projections

Perhaps the most common method of dimensionality reduction, and often the one first learned, is principal component analysis. This method is based on finding the perpendicular axes of maximum variation of a dataset, and is analogous to the principal axes theorem of mechanics. When used as a dimensionality reduction tool, one projects down onto only the first few principal axes of the dataset, hoping that the important structure is preserved in the process. One drawback of this method is that is requires the computation of the eigenvalues of an $N \times N$ matrix, where N is the dimensionality of the space. This cost can be prohibitive when we are dealing with spaces of extremely high dimension.

Suppose that, rather than attempt to intelligently select what axes we project onto, we were to choose from a uniform distribution over the sphere in $\mathbb{R}^N$. This would certainly reduce the dimension of our data, but we have no reason to suspect it would maintain the structure we seek to discern. However, the same concentration of measure which presents us with the challenge also provides us with the solution. Indeed, this phenomenon is the crux of the remarkable lemma of Johnson and Lindenstrauss, which (informally) states that randomized projections of n data points onto spaces of dimension $\mathbb{R}^m$ with $m = \mathcal{O}(\log(n)/\epsilon^2)$ will with high probability preserve pairwise distances between points to within a factor of $1 - \epsilon$ (see [2] for a precise statement). This approach has been used with great success in scientific computing [3] (under the guise of randomized low-rank matrix approximations) and in nearest-neighbor algorithms [4] (there referred to as local-sensitive hashing).

Although the dimension of our data is not so high as to be prohibitive for many of our statistical learning schemes, we will implement randomized-projection based methods and explore their efficacy.

3.2 Graph Construction

There are many different ways to discuss the distance between two nodes on a graph, but no obvious way to imbue the graph with an inner product structure. To that end, we examine methods of embedding the graph into $\mathbb{R}^n$. Once this embedding has been performed, we can take advantage of resulting structure in our classification algorithms.

It is not obvious how this graphical embedding method applies, since we do not in fact have a graph to begin with. If we take each track and flatten the time series of MFCCs, we have a collection of points in $\mathbb{R}^N$ where $N \approx 30,000$. We must begin by forming a graph on which to perform this construction. We do this by using some notion of distance between points to establish a measure of similarity, and then construct a graph with edges weighted by this similarity.

We employ two notions of distance. The first is the simple l_2-distance between two points, taken in the usual way. The second is a slightly more involved measure, the symmetric Kullback-Leibler divergence, which is used to measure the distance between probability distributions.

To calculate the symmetric KL-divergence, we regard the time series as a point cloud in $\mathbb{R}^{30}$, disregarding temporal information. We can then form the covariance matrix and mean vector of this point cloud, and build a corresponding multivariate Gaussian distribution. We thus have a Gaussian distribution corresponding to each track. The KL-divergence between two multivariate Gaussians is known in closed form.

We have two measures of distance between nodes on a graph. We scale them so as to be comparable, then take the minimum of the two. The resulting similarity matrices are shown in Figure 1. We also apply a sharp cutoff, so that if the similarity is below some value κ, we set it to zero. In this situation, we applied a cutoff of $\kappa = 0.8$. This cutoff is not shown Figure 1.

Once we have built a graph, we must embed it down into $\mathbb{R}^m$, hopefully for a reasonably small m. This is enacted via the well-known spectral embedding method. We will not go into detail of the method here, but the

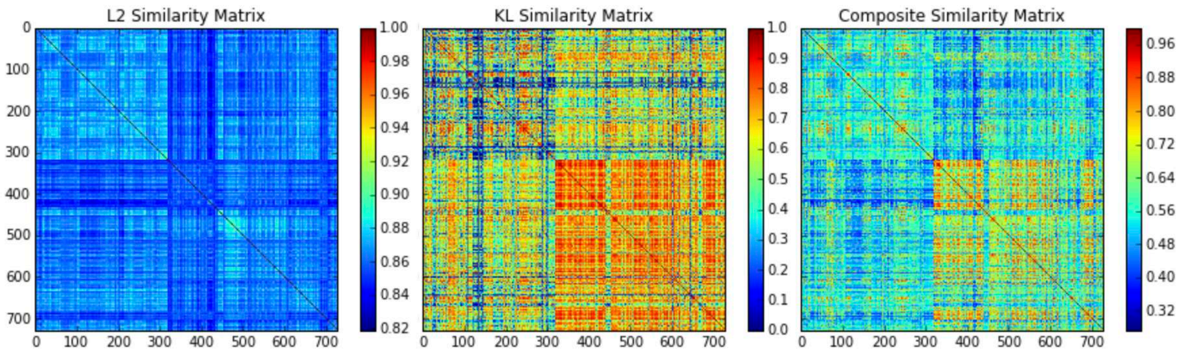


Figure 1: Similarity matrix used to construct graph. Final graph also applies a cutoff $\kappa = 0.8$ (not shown).

basic idea is that if the i^{th} eigenvector of the scaled graph Laplacian $\mathcal{L}$ is $\varphi^{(i)}$, then the embedding for node j is

$$j \to (\varphi_j^{(1)}, \varphi_j^{(2)}, \ldots)$$

and the desired number of coefficients can be kept. Using only the second coefficient reproduces the common method for performing two-community detection on the graph.

3.3 Feature Extraction

Another common method for analyzing audio content is to extract certain "features" from the signal. These might include the l_2 norm, autocorrelation, zero-crossing rate, or envelope shape of the signal itself or the MFCCs. Unfortunately, we do not delve into an examination of these methods in this report. We found that a naive application of statistical learning on features produced results comparable to randomized projections of the MFCCs. We are confident that an intelligent application of feature-based classification is the approach which would yield the most fruit, but it would take some substantial thinking and tinkering to get the right balance. Due to time constraints, we exclude this method entirely rather than present a half-hearted attempt.

4 Statistical Learning

4.1 Naive Bayes

The naive Bayes algorithm assumes the data is well-described by a Gaussian mixture model. That is, it assumes that each genre is represented by a multi-variate Gaussian distribution in feature space. It also makes the very strong assumption that the covariance of each Gaussian distribution is diagonal, meaning that each feature is independent. This is what makes the algorithm "naive". However, naive Bayes often works remarkably well in practice; and the independence assumption makes the run-time very fast.

To build a classifier, we first look at a single dimension in feature space. We assume each genre obeys a normal distribution, so we only need to calculate a mean and variance for each genre. This is shown in Figure 2. From these distributions, we classify a query track as shown in Figure 3. If $\hat{x}$ is the value of the query track in the first feature, we evaluate the probability that $\hat{x}$ was drawn from the distributions of each genre. We obtain a list of numbers, the probability that the query track belongs to each genre. To complete the classifier, we only need to multiply these probabilities across each dimension, according to the naive assumption of independence. We simply choose the genre with the maximum probability.

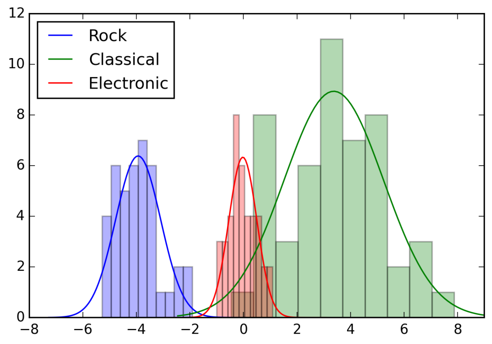


Figure 2: This plot represents a single dimension of feature space. For each genre, we plot a histogram of the values of that feature, across all the tracks in that genre. We fit a bell curve to each histogram. Now each genre has a univariate Gaussian distribution in this dimension.

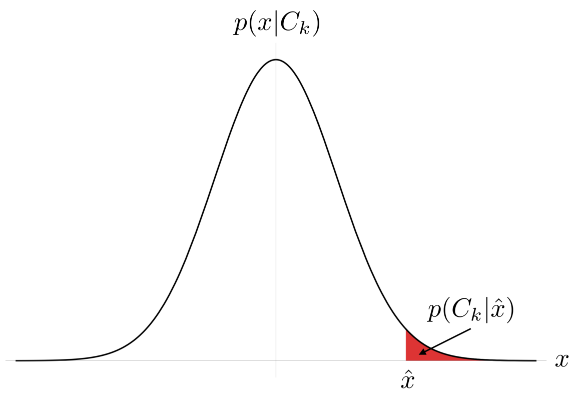


Figure 3: Calculating the likelihood that a query track is from a given gaussian distribution.

4.1.1 Pros and Cons

We chose to try naive Bayes because it is well-known to work very well in high dimensions. It avoids the curse of dimensionality by assuming feature independence. Also, we believed a Gaussian mixture model would aptly capture the genre classifications.

Of course, if feature dependencies are very important for classification, naive Bayes will perform poorly. Zhang explores this question in [5]. Zhang shows that under mild assumptions, dependencies across many features may "cancel out", in such a way that they no longer affect the dependencies. So we may expect naive Bayes to perform well, despite existing dependencies.

4.2 Support Vector Machines

A support vector machine is a binary classification algorithm. Given two sets of labeled points in some feature space, the algorithm finds a hyperplane dividing the two sets. In particular, it finds the unique hyperplane that stays farthest away from the labeled points on either side. This is known as the *maximum-margin hyperplane*. Here we describe the mathematical formulation for finding this hyperplane.

We describe the set of labeled points by a list of ordered pairs $(\mathbf{x}_i, y_i)$ where $\mathbf{x}_i$ is the n-dimensional feature vector for the i^{th} point, and $y_i \in \{1, -1\}$ indicates which class the point belongs to. The maximum-margin hyperplane can be found by minimizing

$$\begin{aligned} &||\mathbf{w}|| \\ \text{subect to} \quad &y_i(\mathbf{w}\cdot\mathbf{x_i} - b) \geq 1 \quad \text{for each i} \end{aligned} \tag{1}$$

where $\mathbf{w}$ is the (unnormalized) normal vector to the hyperplane and $b/||\mathbf{w}||$ is the distance between the hyperplane and the origin in the direction of $\mathbf{w}$.

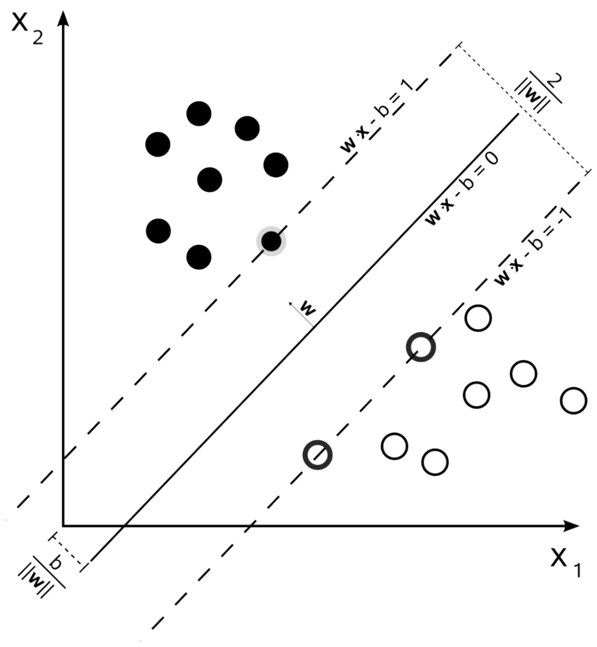


Figure 4: The maximum-margin hyperplane calculated by a support vector classifier. (Courtesy of Wikipedia)

4.2.1 Kernel Trick

Suppose the points in the original space are not separable by a hyperplane, but rathers by a nonlinear curve. We can transform this situation into a linear separation problem by embedding the original points into a higher-dimensional feature space with a nonlinear embedding. The nonlinear embedding function is called the kernel. If we choose the kernel function well, the points will be linearly separable in the feature space. We find the maximum-margin hyperplane in feature space, and then project back down to the original space by inverting the embedding. This gives us a nonlinear classifier in the original space; an example is shown in Figure 5.

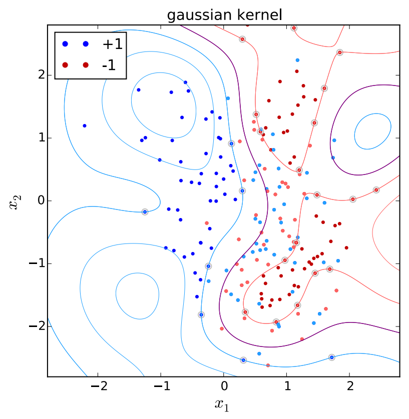


Figure 5: This nonlinear classifier is obtained from a Gaussian kernel.

For this project, we've used a Gaussian radial basis kernel function. This means our feature space is an infinitely-dimensional Hilbert space. The infinite-dimensional embedding ensures that the points are linearly separable in feature space; and fortunately our algorithm still works, despite the high dimension.

A technique called the *kernel trick* actually allows us to build the nonlinear classifier without performing the full embedding described above. We just replace the dot product from Equation 1 with a nonlinear kernel function. For a Gaussian kernel, the kernel function becomes $k(x, y) = \exp(-\gamma||x - y||^2)$ for $\gamma > 0$. We use the default γ from `scikit-learn`, which is $1/N$ where N is the dimensionality of the data put into the algorithm.

4.2.2 Multi-class SVM

The SVM algorithm produces a binary classifier. In order to handle multiple genres, we have to build multiple binary SVM classifiers and combine them. We used *one-vs-one* classification. For each pair of genres, we built a single SVM classifier distinguishing between those two genres.

5 Results

We now present the results of our methods. We report both the total accuracy of classification and a representative confusion matrix. We believe that the total accuracy is a better metric of performance, as it weighs more heavily the better-represented genres. This is natural to do, as we can only expect good performance on genres for which there is a large amount of training data.

Our collection, after having removed all tracks labeled with the "world" genre, contains 607 tracks. We perform cross-validation on this collection. First, we separate the tracks into 10 roughly equal sets. Then we classify the first set, training on the remaining nine. We do the same with the second set, and so on, until all tracks have been tested. We then compute an accuracy measure by comparing the classified genre to the known actual genre. We repeat this process 10 times in order to obtain a distribution of our accuracy measure. The data points and error bars shown are, unless otherwise noted, the mean and standard deviation of this set of 10 accuracy measures.

5.1 Random Projections

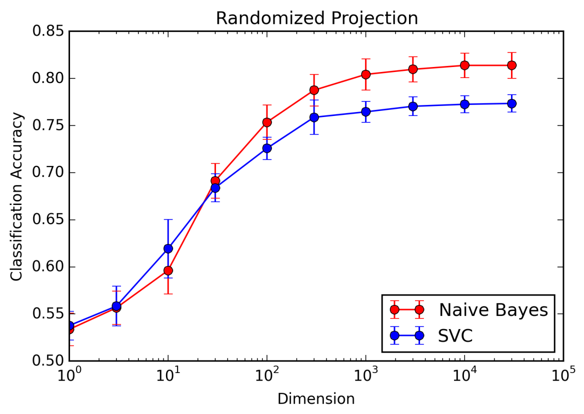
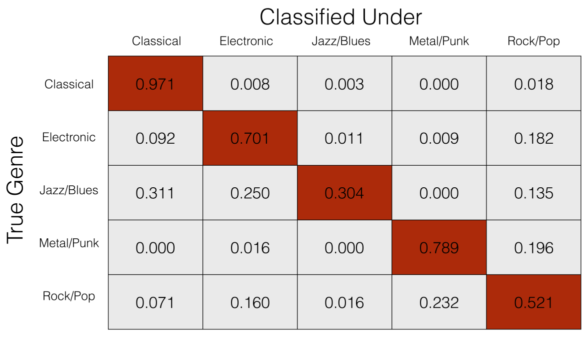


Figure 6: Performance of randomized projection methods.

The results for randomized projections are shown in Figure 6. We observe that the performance of the naive Bayes classifier surpasses that of the support vector classifier once the dimension is higher than 10. We have no significant gains in accuracy when the dimension is increased beyond 1000, and therefore the best method seems to be to randomly project the data into $\mathbb{R}^{1000}$ and apply a naive Bayes classifier.



Classified Under

True Genre	Classical	Electronic	Jazz/Blues	Metal/Punk	Rock/Pop
Classical	0.971	0.008	0.003	0.000	0.018
Electronic	0.092	0.701	0.011	0.009	0.182
Jazz/Blues	0.311	0.250	0.304	0.000	0.135
Metal/Punk	0.000	0.016	0.000	0.789	0.196
Rock/Pop	0.071	0.160	0.016	0.232	0.521

Figure 7: Confusion matrix using naive Bayes classifier on the MFCCs, randomly projected into $\mathbb{R}^{1000}$

A representative confusion matrix is shown in Figure 7. The most problematic genre is jazz, which is not surprising, given the low quantity of training data and high acoustic variability of the genre. We see two groups emerge: one is classical and jazz/blues, while the other is electronic, metal/punk, and rock/pop.

5.2 Graph Embedding

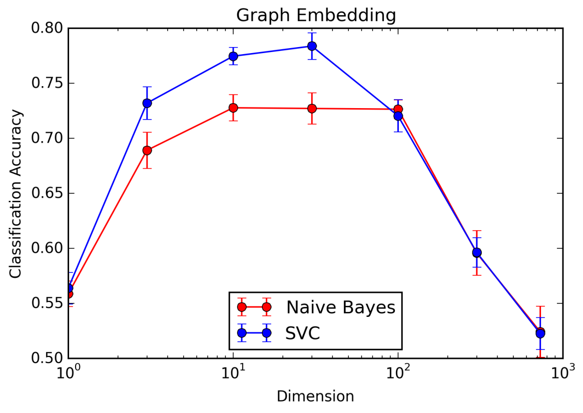
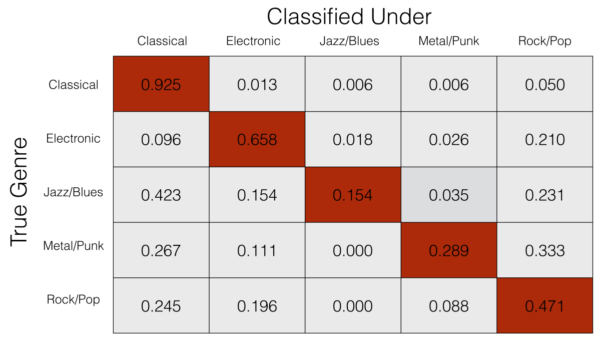


Figure 8: Performance of graph embedding methods.

We now look at the results of the graphical embedding method, shown in Figure 8. We see that, unlike randomized projections, there is a steep decrease in quality of classification when we embed into $\mathbb{R}^m$ for $m > 10$ or so. This is to be expected, as there is a natural ordering present here; the initial dimensions are useful for classifying graphs with a low number of communities, while the higher dimensions classify graphs with many communities. Since our graph does not have more than 5 apparent communities (indeed, only two are readily apparent from Figure 1) these higher dimensions are effectively noise, and strongly decrease the efficacy of our method.



Classified Under / True Genre

	Classical	Electronic	Jazz/Blues	Metal/Punk	Rock/Pop
Classical	0.925	0.013	0.006	0.006	0.050
Electronic	0.096	0.658	0.018	0.026	0.210
Jazz/Blues	0.423	0.154	0.154	0.035	0.231
Metal/Punk	0.267	0.111	0.000	0.289	0.333
Rock/Pop	0.245	0.196	0.000	0.088	0.471

Figure 9: Confusion matrix using naive Bayes classifier on the spectral embedding, keeping 100 coefficients.

A representative confusion matrix is shown in Figure 9. The method performs quite well on classical, but is lackluster in the areas of jazz/blues and rock/pop. This is again not unexpected, but a discussion of why this occurs (beyond a lack of training data) is beyond the scope of this work.

6 Improvements & Extensions

The methods employed here are not expected to be optimal, for various reasons. This project is intended as an exploration of dimensionality reduction techniques as taught in the course. These methods show some effectiveness, but we do not expect them to excel when the problem is extended to a realistic scale.

The methods that we expect would be most effective will be dimensionality reduction techniques specific to audio analysis, that is to say audio feature extraction. One would want to incorporate low-level features, such as autocorrelation and zero-crossing rate, as well as high-level features, such as tonic frequency, tempo, time signature, instrumentation, etc. These would have to be handled in an intelligent way, with different weights applied in the classifier algorithm. One must also carefully consider how we order such quantities. For example, $A^\flat$ is musically much closer to $E^\flat$ than it is to $A^\natural$, and in the case of tonic frequency a circle-of-fifths ordering would likely be the appropriate one. When it comes to time signatures the issue becomes even stickier: is 4/4 time closer to 3/4 or 6/8? In this scenario we may have to consider the features as unordered discrete variables.

Effective classification is dependent upon a reasonable genre taxonomy. However, there is no consensus in the musical community about such a taxonomy. [6] One must define genres that are acoustically meaningful; any genre without defining acoustic properties will not be discerned by a classifier that looks at the audio signal alone. This is the reason we have discarded the "world" genre from our examination; were we to delve into a larger and more complicated taxonomy of genres, we would need to make many more such decisions.

An effective classifier would also require a training set that is orders of magnitude larger than the one used here. Obtaining such a set is difficult given copyright laws. Often internet radio platforms provide the genre of a track, and a program could be written that trains off of signals obtained from such services. However, the legality of this would become highly dubious were this to become a commercial product.

7 Conclusion

We have studied the application of dimensionality reduction techniques to the problem of automated genre classification. We find that one can reduce the dimension of the audio signal (MFCCS) by 97% using randomized projections and still retain $> 98\%$ performance when using a Gaussian naive Bayes classifier. We find support vector classifiers to have similar but slightly less desirable performance. Methods based on graph embeddings are significantly less effective, due to the simplicity of our graph construction.

Appendix

A Computational Details

Here we will outline the process used to compute the MFCCs of a given track. We begin by trimming a track to 2 minutes using the following algorithm.

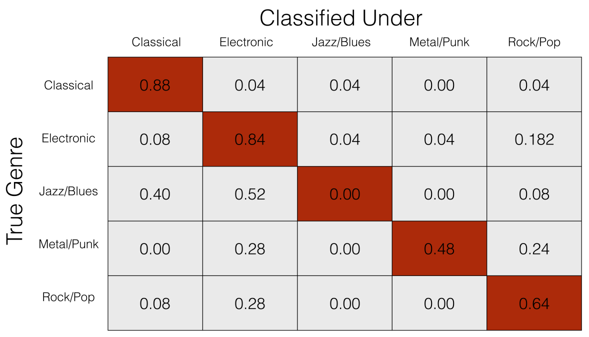
```
if length(track)*(2/3) > 2 min
    grab middle 2 min of track
else
    grab middle 2/3rds of track
    loop until length is > 2 min
    truncate to 2 min
end
```

This removes any introduction or conclusion portion of the song, which is often a poor indicator of genre. We then use the `features` module in python [7] to compute the MFCCs of a given track, using a window length of .2 seconds, window overlap of .1 seconds, and 30 frequency bins. This yields a matrix which is 1199×26.

All statistical learning algorithms are implemented using the `scikit-learn` module in python [8]. Naive Bayes is performed using the default gaussian kernel. The support vector machine classifier is implemented by first scaling all data using the included `preprocessing` module.

B Test Data Results

We report the results for the test data provided recently. We used our naive Bayes classifier with 1000 dimensions. We obtain an accuracy of 56.8%. The confusion matrix is shown in Figure 10.



Classified Under

True Genre

	Classical	Electronic	Jazz/Blues	Metal/Punk	Rock/Pop
Classical	0.88	0.04	0.04	0.00	0.04
Electronic	0.08	0.84	0.04	0.04	0.182
Jazz/Blues	0.40	0.52	0.00	0.00	0.08
Metal/Punk	0.00	0.28	0.00	0.48	0.24
Rock/Pop	0.08	0.28	0.00	0.00	0.64

Figure 10: Confusion matrix using naive Bayes classifier on the test data.

References

[1] M. Talagrand. A new look at independence. *Ann. Probab.*, 24(1):1–34, 01 1996.

[2] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary Mathematics*, 26, 1984.

[3] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[4] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, 1999.

[5] H. Zhang. The optimality of naive bayes. *AA*, 1(2):3, 2004.

[6] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. In *IEEE Transactions on Speech and Audio Processing*, pages 293–302, 2002.

[7] J. Lyons. `python_speech_features`. GitHub repository, 2013. `https://github.com/jameslyons/python_speech_features`.

[8] 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.

[9] Md. Sahidullah and G. Saha. Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition. *Speech Communication*, 54(4):543 – 565, 2012.

[10] A. Lerch. *An Introduction to Audio Content Analysis: Applications in Signal Processing and Music Informatics*. Wiley-IEEE Press, 2012.