AUTOMATIC MUSIC CLASSIFICATION

18-551 Capstone Project, Group #8

Murium Iqbal <miqbal88@gmail.com>
Apoorv Khandelwal <akhandel@andrew.cmu.edu>
Karanhaar Singh <singhk@cmu.edu>

Spring 2012

Table of Contents

1 PROJECT OVERVIEW **3**

1.1 Motivation 3

1.2 Solution 4

2 FEATURES **5**

2.1 Timbral Texture 5

2.2 Rhythmic Content 8

2.3 Pitch Content 10

3 CLASSIFIERS **11**

3.1 Naïve Bayesian 11

3.2 Full Covariance Bayesian 12

3.3 Support Vector Machine 14

4 CHALLENGES **16**

5 RESULTS **21**

5.1 Definitions 21

5.2 Breakdown 24

5.3 Summary 30

5.4 Final Demo 32

6 TEAM LOGISTICS **34**

7 FUTURE WORK **36**

8 REFERENCES **37**

SECTION 1: PROJECT OVERVIEW

Music is wide and varied, with divisions among genres being almost entirely subjective. For our capstone project, we decided to attack the timeless problem of classifying songs into genres. During this semester, we built a classifier which automatically classifies music into user-selected genres.

SECTION 1.1: MOTIVATION

No human can effectively wade through the ocean of digital music available on the Internet, finding all the songs he likes. Thus, researchers have explored numerous ways of automatically identifying, classifying, and organizing music based on their waveforms alone. This is a difficult problem because the music covering any given genre is composed from instruments and musicians of various types and backgrounds, and thus, is extremely diverse and constantly evolving. An automatic music classification Android application would be practical to the average user for multiple reasons. One potential use is to help the user sort through a large set of unlabeled music, perhaps from mix CDs obtained from friends. Another potential use is to recognize music recorded by the tablet's built-in microphone.

Group 5 from Spring 2011 had a similar project idea as ours, but they mapped songs to emotions. We were motivated towards classifying songs into predefined genres because we could perform more objective tests than that group. While both problems may be subjective, the emotions stirred from person to person upon hearing a song probably vary much more than the people's opinions of the song's genre. Group 5's final

implementation utilized an SVM for separating music based on its arousal and valence. While our final Android implementation uses a naïve Bayes classifier instead of an SVM, we have discovered through our Matlab implementation that an SVM classifier would work well too.

SECTION 1.2: SOLUTION

Most approaches that apply rigid hard-coded rules to separate music are eventually rendered obsolete due to the mercurial nature of musical genres. In their paper "Musical Genre Classification of Audio Signals" [1], Tzanetakis and Cook present a way to extract three sets of features from music waveforms that could be used for classifying music into genres. These sets of features are identified as Timbral Texture, Rhythmic Content, and Pitch Content. Using these features, standard machine learning techniques can be applied to identify music of various genres and a classifier can be trained to accurately classify music provided by a user. For our project, we implemented a feature extraction system as well as classifiers for this very purpose.

2 FEATURES

For the feature extraction part of the project in Matlab, we initially started out with a framework for feature extraction set up by Professor Richard Stern. With his approval, we heavily modified his code to compute the features we used in this project. The computation of MFCC coefficients in Matlab was done via an external library of functions, while the extraction of pitch content and rhythmic content features was done with in-house code.

2.1 Timbral Texture

According to [3] and [4], the features which comprise the timbral texture are commonly used in music-speech discrimination and speech recognition. To obtain these features, we first break the input signal into small blocks of a particular size, window each block, and take the discrete Fourier transform (DFT) of each block. This process is also known as the short time Fourier transform (STFT) and is commonly used in music processing. Then, using the STFT of the input signal, we compute the spectral centroid, spectral rolloff frequency, and spectral flux attributes. Finally, we compute the time-domain zero crossings and Mel-Frequency Cepstral Coefficients (MFCC). The latter, motivated by how humans perceive frequencies, represent the short term power spectrum of a given sample of sound. Further details regarding STFT and MFCC are in [2], while the definitions of the other four timbral texture attributes are below.

Spectral Centroid: During a frame for STFT, think of the frequency in the song as a random variable whose probability distribution is the period's normalized magnitude spectrum. Then the spectral centroid is the expected value of the song's frequency for the time period of that STFT.

$$C_t = \frac{\sum_{f=1}^{N} M_t[f] f}{\sum_{f=1}^{N} M_t[f]} \; where \; M_t[f] \; is \; the \; magnitude \; of \; the \; Fourier \; transform.$$

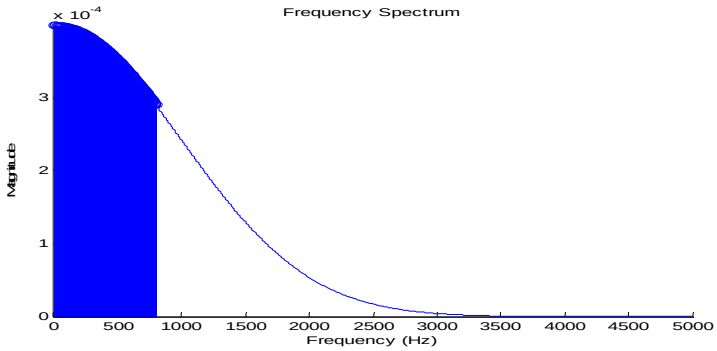

Figure 2.1.1: The frequency at the dividing line represents the spectral centroid of the frequency spectrum.

Spectral Rolloff Frequency: This is the frequency below which 85% of the song's content is contained.

$$\sum_{f=1}^{R_t} M_t[f] = 0.85 \sum_{f=1}^{N} M_t[f] \; where \; R_t \; is \; the \; spectral \; rolloff \; frequency.$$

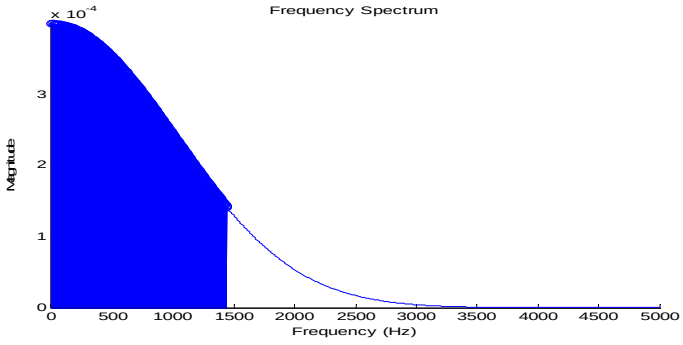

Figure 2.1.2: The frequency at the dividing line represents the spectral rolloff frequency for this spectrum.

Spectral Flux: This is the squared difference between the normalized magnitudes of temporally consecutive spectral distributions.

$$F_t = \sum_{f=1}^{N} (N_t[f] - N_{t-1}[f])^2 \ \ where\ N_t[f]\ is\ the\ normalized\ M_t[f].$$

Time Domain Zero Crossings: This provides a measure of the noisiness of the signal by computing the number of times the signal crosses zero in the time domain.

$$Z_t = \frac{1}{2}\sum_{n=1}^{N} |sign(x[n]) - sign(x[n-1])| \ \ where\ x[n]\ is\ the\ input\ signal.$$

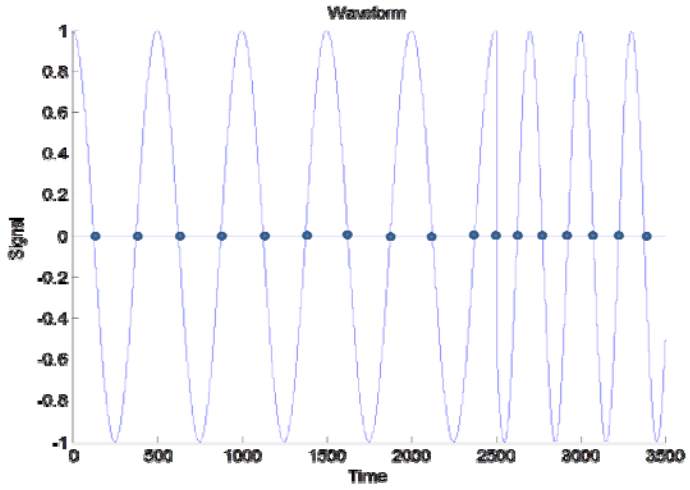

Figure 2.1.3. The number of highlighted points represent the time domain zero crossings of the signal

2.2 Rhythmic Content

This set of features is based on the detected major beats in the music. To extract the main beats, the input is broken into three-second length windows, on which the Discrete Wavelet Transform (DWT) is computed [3]. The outputs of the DWT are N sets of samples, one set for each octave range. Each set is downsampled so that they all lie in the same octave. To this resultant set, we apply full wave rectification and low pass filtering to extract the envelope of the samples. We then downsample the result again, this time to reduce computation time. Finally, we estimate the strongest beats of the result by finding the peaks of correlating the signal with itself. After cleaning this signal and normalizing it, we have a histogram of the beats, from which we can extract known relevant features. The steps required to obtain Rhythmic Content features are explained below:

Full Wave Rectification: This is used to extract the time-domain envelope of the signal.

$$y[n] = |x[n]| \; where \; x[n] \; is \; the \; input \; signal.$$

Low-pass Filtering: This is a one-pole filter to smooth out the envelope.

$$y[n] = (1 - \alpha)x[n] + \alpha y[n-1] \; where \; \alpha = 0.99$$

Downsampling: This reduces the computation time for the subsequent autocorrelation of the function.

$$y[n] = x[kn] \; where \; k = 16.$$

Mean-removal: This is required to make the signal's average amplitude zero for correct autocorrelation.

$$y[n] = x[n] - \frac{1}{N}\sum_{n=1}^{N} x[n]$$

Enhanced Autocorrelation: The peaks of this output signal designate the major beat periodicities in the signal.

$$y[k] = \frac{1}{N}\sum_{n=1}^{N} x[n]x[n-k].$$

2.3 Pitch Content

This set of features is computed on a time scale over two orders of magnitude less than that used to obtain the Rythmic Content features. The histogram peaks (as obtained by the description in section 2.2) are mapped to MIDI note numbers which are used to compute the folded and unfolded histograms. From these two histograms, the following features are computed—amplitude of the maximum peak of the folded histogram, period of the maximum peaks of the unfolded and folded histograms, pitch interval between the two highest peaks of the folded histogram, and finally, the overall sum of the histogram.

FA0: This is the dominant pitch class of the song. This peak is higher for songs that do not have many harmonic changes.

UP0: This is the maximum peak of the unfolded histogram, corresponding to the major octave range of the song.

FP0: This is the maximum peak of the folded histogram, corresponding to the major pitch class of the song.

IPO1: This is the interval between the two most prominent peaks in the folded histogram, which corresponds to the major tonal interval relation.

SUM: Adding up the values of the histogram measures how strong the pitch detection is for the song.

3 CLASSIFIERS

We experimented with three classifiers for our system. Specifically, we started off with the classifier easiest to implement—the naive Bayesian classifier. Upon obtaining promising results, we decided to test how well our features distinguish between genres by modifying the Bayesian classifier to one which accounts for the correlation between features. Finally, we implemented our system using completely different machinery in the form of support vector machines. For the remainder of this section, m refers to the number of features and n refers to the number of classes.

3.1 Naive Bayesian

This was the first classifier we implemented in Matlab and the one we ended up with on the Android tablet. This classifier makes several strong (and highly unlikely) assumptions, rendering its positive results all the more surprising. This classifier assumes, first of all, that all features for each genre are distributed normally, with some defining mean and variance. This assumption is proven to be reasonable as it follows from the Central Limit Theorem, which states that the distribution of a random variable

$$X = \sum_{i=1}^{I} X_i \ where\ X_n\ are\ independent\ from\ each\ other$$

converges to a Gaussian as $I \to \infty$. But this classifier also assumes that the features being used are independent of each other, which is most likely an oversimplification [7]. Because of these strong assumptions, the naive Bayesian classifier had the worst (although satisfactory) results out of our three classifier implementations. The

combination of these assumptions can lead to learning distributions of classes as shown in Figure 3.1.1.

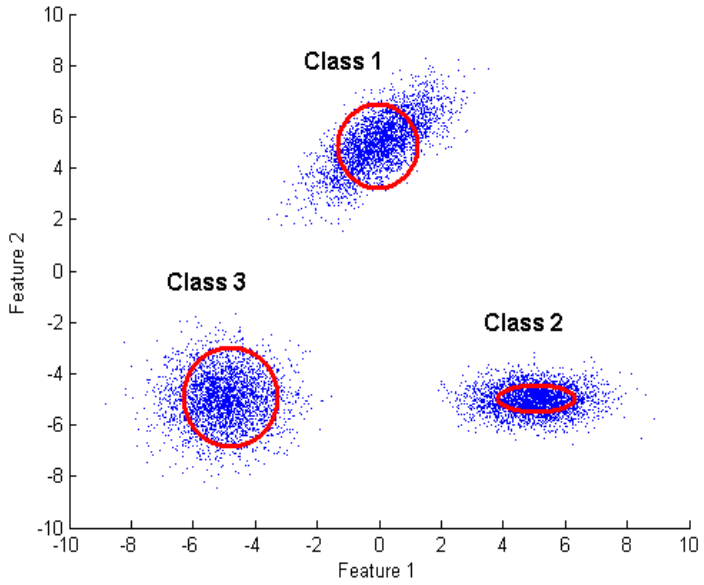

Figure 3.1.1: Distributions of classes determined by a naïve Bayesian classifier

3.2 Full Covariance Bayesian

This was the second classifier we implemented in Matlab and one that can be implemented on Android in the future without too much trouble. This classifier no longer assumed that all the features were independent from each other. This weakening of our assumptions could allow the classifier to learn distributions as shown in Figure 3.2.1. Like the naive Bayesian classifier, we assumed that all features for each genre had a normal distribution. In this case, we defined each multidimensional Gaussian probability density function with a mean vector and a covariance matrix. We computed the mean vector by finding the mean of all the features for each class from the training data. The covariance matrix was computed by taking the covariance for each pair of

features for each class from the training data. Since we were computing a total of $\frac{m}{2}(m+1)$ total covariances for this classifier as opposed to m variances in the naive Bayesian classifier, training took longer, especially as the number of features m increased. In terms of classifier performance, the full covariance classifier did better than the naive classifier when we had more than three classes. Due to the fact that our self-validation results were good, we believed that poor performance for a smaller number of classes was due to overfitting.

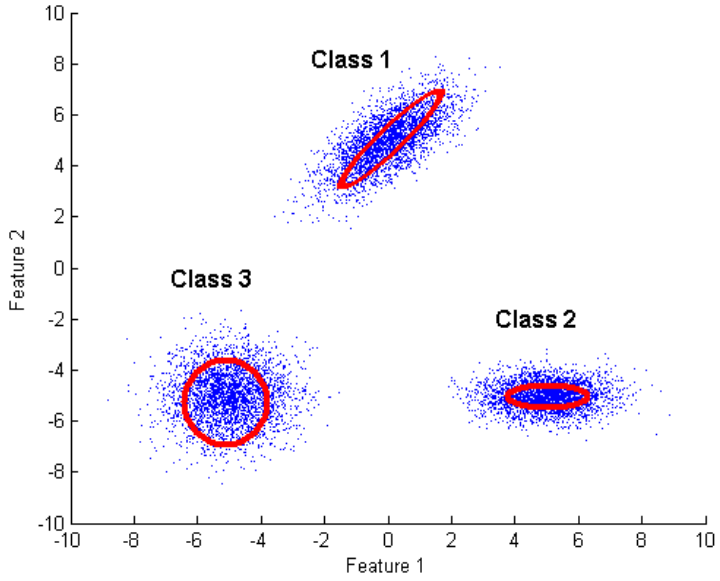

Figure 3.2.1: Distributions of classes determined by a full covariance classifier

3.3 Support Vector Machine

This was the third classifier we implemented on Matlab and one that we tried to import to the tablet but failed. Since typical SVMs can only handle 2-class problems, we used multiple SVMs to classify between all of our n classes. There are two possible ways to approach this problem—either using a 1 vs. all or a 1 vs. 1 algorithm [6].

1 vs. all: This algorithm was initially more attractive to us since we could use a smaller number of SVMs. However, due to a variety of issues, we decided not to use this implementation. Thus, it was not fully completed and is not in our included MATLAB code. Specifically, the algorithm works as follows: for each class from our training set of data, define the selected class to be the respective class name (such as "rock"), and define all other $n - 1$ classes to be NOT the selected class (such as "not rock"). After this, train the SVM on these two classes and save the SVM trained object. Repeat this process for all other $n - 1$ classes to get an array of n SVM classifiers. To classify a song, we input the extracted features into each of the n SVM objects. Ideally, one of the n classifiers should return a positive result (IS class X) and all other classes should return a negative result (IS NOT class X). In this case, classify the song to be the positive class. In the case where two or more SVMs give a positive result, we randomly picked between the positive results. Finally, in the case where all SVMs gave negative results, we simply refused to class. The biggest issue for us is that in our preliminary tests, our beta implementation was giving poor results, so we completely scrapped our code for our 1 vs. 1 implementation. Although this wasn't heavily tested (since 1 vs. 1 worked well and

we decided to go ahead with it), we believed the reason the 1 vs. all implementation did not work was due to the fact that the "not X" class had a multimodal distribution, which is something linear SVMs handle rather poorly.

1 vs. 1: This is the algorithm that we used in our final Matlab implementation. Specifically, the algorithm works as follows: for all pairs of classes, we trained an SVM object using these two classes (such as rap and rock), and saved the resultant SVM object, obtaining $\frac{n}{2}(n-1)$ total SVM objects. One way to represent this set of SVM objects is to have a symmetric SVM matrix where rows and columns both refer to classes. Notice that although we generated more SVM objects for the 1 vs. 1 implementation, the generation of each SVM object was faster since, on average, we used only $\frac{2}{n}$ of the training data for each object. For testing, we took the extracted features from our test song and tested on all $\binom{n}{2}$ SVM objects. We then selected the class represented by the song based on summing up the votes from each SVM object. Furthermore, we easily obtained our class rank list by counting up the votes. *Example*: Consider four classes: [rock rap classical country]. If we test a new country rock song, the votes may look like [**rock** vs. rap, **rock** vs. classical, rock vs. **country**, rap vs. **classical**, rap vs. **country**, and classical vs. **country**]. In this case, our votes are [country: 3, rock: 2, classical: 1, rap: 0]. So we would say that this song is most likely to be country, somewhat likely to be rock, not very likely to be classical, and almost surely not rap.

4 CHALLENGES

We faced a major challenge in porting an SVM implementation to Android. We initially found a library named WEKA which contained a (tested) working SVM implementation. But after numerous attempts to include the WEKA jar file with our project, we discovered that Android does not support the use of the WEKA library. As there was not enough time to implement our own SVM classifier in Java, we stuck with 5the naive Bayesian classifier.

We had trouble implementing the final ten of the timbral texture set of features—the means and variances of the first five MFCC coefficients--on Android. The external libraries containing the code for computing MFCC coefficients were quite difficult to utilize. So taking into account the relative difficulty of porting their computation to Android, the delay to the user from having to wait for MFCC coefficient computation, and the fact that we were already obtaining reasonable results using just the first eight of the timbral texture features, we decided not to add these features to our Android app.

We wanted to see whether our classifier could distinguish between songs sung in different languages. Thus, on top of rock, rap, classical, country, and techno, we tried adding Hindi music to our classifier. But the results obtained were, although better than random, completely unsatisfactory compared to our previous results. This showed us that our classifier only worked for distinguishing genres which were disjoint (as we should have guessed). We also tried testing Hindi music against just one other genre, which combined rock, rap, and country English songs. We hoped this would provide a

greater distinction between the classes. However, as can be seen by the results in the following matrices, the features we use do not work well for distinguishing between songs of different languages. They are apparently better at distinguishing the instruments in the music than the language being spoken.

Note: For six genres, the amount of training music used for self-validation was the same as described in Section 5.2 for the five genres, adding on 94 Hindi songs. For cross validation, the amount of each genre is as described in Section 5.2 for the first five genres, with 21 Hindi songs. In the two-genre case of English vs. Hindi, there were 100 English and 94 Hindi songs used for training in self-validation. 25 English and 21 Hindi songs were separated from the rest of the songs for testing in cross-validation.

Naïve Bayesian:

Six Genres:

Self-Validation Accuracy:

	Rck	Rap	Cls	Cnt	Tch	Hnd
Rck	0.61	0.03	0.04	0.16	0.08	0.07
Rap	0.03	0.81	0.01	0.02	0.04	0.10
Cls	0.04	0.01	0.89	0.05	0.00	0.01
Cnt	0.18	0.01	0.08	0.63	0.01	0.08
Tch	0.08	0.11	0.01	0.01	0.73	0.07
Hnd	0.10	0.16	0.03	0.27	0.06	0.38

Cross-Validation Accuracy:

	Rck	Rap	Cls	Cnt	Tch	Hnd
Rck	0.73	0.06	0.02	0.14	0.02	0.04
Rap	0.05	0.78	0.00	0.00	0.03	0.15
Cls	0.13	0.00	0.88	0.00	0.00	0.00
Cnt	0.20	0.04	0.16	0.55	0.00	0.04
Tch	0.07	0.08	0.00	0.07	0.64	0.14
Hnd	0.05	0.24	0.05	0.19	0.10	0.38

Self-Validation Rank Accuracy:

Rank (Wt)	Rck	Rap	Cls	Cnt	Tch	Hnd
1 (1)	0.61	0.81	0.89	0.63	0.73	0.38
2 (4/5)	0.17	0.08	0.05	0.25	0.12	0.28
3 (3/5)	0.13	0.04	0.03	0.07	0.09	0.28
4 (2/5)	0.06	0.03	0.02	0.05	0.03	0.06
5 (1/5)	0.02	0.04	0.01	0.00	0.01	0.00
6 (0)	0.00	0.00	0.00	0.00	0.01	0.00
Rank Acc:	**0.86**	**0.92**	**0.96**	**0.89**	**0.90**	**0.80**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rck	Rap	Cls	Cnt	Tch	Hnd
1 (1)	0.73	0.78	0.88	0.55	0.64	0.38
2 (4/5)	0.14	0.10	0.00	0.35	0.11	0.48
3 (3/5)	0.04	0.03	0.00	0.04	0.13	0.14
4 (2/5)	0.08	0.03	0.08	0.06	0.06	0.00
5 (1/5)	0.02	0.08	0.04	0.00	0.04	0.00
6 (0)	0.00	0.00	0.00	0.00	0.03	0.00
Rank Acc:	**0.89**	**0.90**	**0.92**	**0.88**	**0.83**	**0.85**

Two Genres:

Self-Validation Accuracy:

	English	Hindi
English	0.55	0.45
Hindi	0.16	0.84

Cross-Validation Accuracy:

	English	Hindi
English	0.60	0.40
Hindi	0.19	0.81

Self-Validation Rank Accuracy:

Rank (Wt)	English	Hindi
1 (1)	0.55	0.84
2 (0)	0.45	0.16
Rank Acc:	**0.55**	**0.84**

Cross-Validation Rank Accuracy:

Rank (Wt)	English	Hindi
1 (1)	0.60	0.81
2 (0)	0.40	0.19
Rank Acc:	**0.60**	**0.81**

Full Cov. Bayesian:

Six Genres:

Self-Validation Accuracy:

	Rck	Rap	Cls	Cnt	Tch	Hnd
Rck	0.71	0.06	0.02	0.15	0.01	0.06
Rap	0.05	0.86	0.01	0.01	0.03	0.05
Cls	0.03	0.00	0.96	0.01	0.00	0.00
Cnt	0.10	0.01	0.04	0.80	0.01	0.04
Tch	0.02	0.02	0.01	0.01	0.92	0.02
Hnd	0.14	0.06	0.00	0.16	0.02	0.62

Cross-Validation Accuracy:

	Rck	Rap	Cls	Cnt	Tch	Hnd
Rck	0.78	0.08	0.00	0.10	0.02	0.02
Rap	0.05	0.78	0.00	0.00	0.05	0.13
Cls	0.04	0.00	0.96	0.00	0.00	0.00
Cnt	0.16	0.04	0.10	0.63	0.00	0.06
Tch	0.11	0.04	0.00	0.01	0.76	0.07
Hnd	0.10	0.19	0.05	0.19	0.05	0.43

Self-Validation Rank Accuracy:

Rank (Wt)	Rck	Rap	Cls	Cnt	Tch	Hnd
1 (1)	0.71	0.86	0.96	0.80	0.92	0.62
2 (4/5)	0.20	0.09	0.02	0.14	0.04	0.18
3 (3/5)	0.07	0.03	0.01	0.04	0.02	0.17
4 (2/5)	0.02	0.02	0.00	0.02	0.01	0.03
5 (1/5)	0.00	0.00	0.00	0.00	0.01	0.00
6 (0)	0.00	0.00	0.01	0.00	0.00	0.00
Rank Acc:	**0.92**	**0.96**	**0.98**	**0.94**	**0.97**	**0.88**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rck	Rap	Cls	Cnt	Tch	Hnd
1 (1)	0.78	0.78	0.96	0.63	0.76	0.43
2 (4/5)	0.10	0.13	0.00	0.18	0.07	0.19
3 (3/5)	0.08	0.08	0.00	0.10	0.06	0.19
4 (2/5)	0.02	0.03	0.00	0.06	0.03	0.10
5 (1/5)	0.02	0.00	0.00	0.02	0.07	0.05
6 (0)	0.00	0.00	0.04	0.00	0.01	0.05
Rank Acc:	**0.92**	**0.93**	**0.96**	**0.87**	**0.88**	**0.74**

Two Genres:

Self-Validation Accuracy:

	English	Hindi
English	0.81	0.19
Hindi	0.05	0.95

Cross-Validation Accuracy:

	English	Hindi
English	0.72	0.28
Hindi	0.33	0.67

Self-Validation Rank Accuracy:

Rank (Wt)	English	Hindi
1 (1)	0.81	0.95
2 (0)	0.19	0.05
Rank Acc:	**0.81**	**0.95**

Cross-Validation Rank Accuracy:

Rank (Wt)	English	Hindi
1 (1)	0.72	0.67
2 (0)	0.28	0.33
Rank Acc:	**0.72**	**0.67**

Support Vector Machine:

Six Genres:

Self-Validation Accuracy:

	Rck	Rap	Cls	Cnt	Tch	Hnd
Rck	0.72	0.04	0.01	0.14	0.01	0.08
Rap	0.03	0.86	0.00	0.01	0.01	0.08
Cls	0.01	0.00	0.96	0.03	0.00	0.00
Cnt	0.14	0.01	0.04	0.71	0.00	0.10
Tch	0.02	0.02	0.00	0.01	0.92	0.02
Hnd	0.12	0.06	0.01	0.13	0.02	0.67

Cross-Validation Accuracy:

	Rck	Rap	Cls	Cnt	Tch	Hnd
Rck	0.76	0.08	0.00	0.12	0.02	0.02
Rap	0.10	0.70	0.00	0.00	0.08	0.13
Cls	0.04	0.00	0.92	0.04	0.00	0.00
Cnt	0.12	0.02	0.08	0.69	0.00	0.08
Tch	0.04	0.04	0.00	0.01	0.79	0.11
Hnd	0.05	0.19	0.05	0.19	0.00	0.52

Self-Validation Rank Accuracy:

Rank (Wt)	Rck	Rap	Cls	Cnt	Tch	Hnd
1 (1)	0.72	0.86	0.96	0.71	0.92	0.67
2 (4/5)	0.17	0.09	0.04	0.24	0.03	0.19
3 (3/5)	0.09	0.03	0.00	0.04	0.03	0.11
4 (2/5)	0.02	0.02	0.00	0.01	0.01	0.03
5 (1/5)	0.00	0.00	0.00	0.01	0.01	0.00
6 (0)	0.00	0.00	0.00	0.00	0.00	0.00
Rank Acc:	**0.92**	**0.95**	**0.99**	**0.93**	**0.97**	**0.90**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rck	Rap	Cls	Cnt	Tch	Hnd
1 (1)	0.76	0.70	0.92	0.69	0.79	0.52
2 (4/5)	0.16	0.18	0.08	0.20	0.04	0.29
3 (3/5)	0.06	0.08	0.00	0.08	0.04	0.14
4 (2/5)	0.02	0.05	0.00	0.02	0.04	0.05
5 (1/5)	0.00	0.00	0.00	0.00	0.07	0.00
6 (0)	0.00	0.00	0.00	0.00	0.01	0.00
Rank Acc:	**0.93**	**0.91**	**0.98**	**0.91**	**0.88**	**0.86**

Two Genres:

Self-Validation Accuracy:

	English	Hindi
English	0.83	0.17
Hindi	0.24	0.76

Cross-Validation Accuracy:

	English	Hindi
English	0.68	0.32
Hindi	0.48	0.52

Self-Validation Rank Accuracy:

Rank (Wt)	English	Hindi
1 (1)	0.83	0.76
2 (0)	0.17	0.24
Rank Acc:	**0.83**	**0.76**

Cross-Validation Rank Accuracy:

Rank (Wt)	English	Hindi
1 (1)	0.68	0.52
2 (0)	0.32	0.48
Rank Acc:	**0.68**	**0.52**

5 RESULTS

To test our choices for our system's classifiers and features, we ran numerous Matlab simulations on both training and testing data. While we obtained promising results for up to five different genres, we had several sanity checks in place while measuring the accuracy to make sure our results made sense.

5.1 Definitions

Assume our classifier is trained with m genres $C_1, \dots, C_m$. Let's say we are given N input songs $A_1, \dots, A_N$ of genres $g_1, \dots, g_N$, respectively. Then our initial naïve measure of classifier accuracy was simply to calculate

$$\frac{1}{N}\sum_{i=1}^{N} f(A_i) \ \ where \ f(A_i) = \begin{cases} 1, classifier \ chooses \ genre \ g_i \ for \ song \ A_i \\ 0, otherwise \end{cases}.$$

This accuracy measure is shown in the plots in section 5.3 for all our implemented classifiers. A more detailed version of this accuracy measure is shown in the confusion matrices in section 5.2, where we show, given a song from a particular genre (row), the likelihood that our classifier will classify it as any genre (column). This accuracy measure is shown in the cells labeled "Self-Validation" and "Cross Validation". For example, "*Naïve Bayesian Results*: Three Genres: Self-Validation" states that 86% of rock songs were classified correctly, 7% were classified as rap, and 6% were classified as classical. Moreover, 92% of rap songs were classified correctly, 7% were classified as rock, and 1% were classified as classical. Finally, 93% of classical songs were classified correctly, 6%

were classified as rock, and 1% were classified as rap. For comparison against a worst-case baseline, we can determine this expected accuracy for a random classifier:

$$E\left[\frac{1}{N}\sum_{i=1}^{N} f(A_i)\right] = \frac{1}{N}\sum_{i=1}^{N} E[f(A_i)] = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{m} = \frac{1}{m}.$$

Both our classifiers' accuracies and the random classifier's accuracy are plotted in Figures 5.3.1 and 5.3.2.

Apart from the above accuracy measure, we also calculated an accuracy measure which gave a more complete picture of our classifier's accuracy. Specifically, if our classifier gave an ordered list of its m trained genres $[h_1, \ldots h_m]$ for a song A_i, then we could determine where g_i is in the list, assigning that position to r_i. So, if our classifier could be allowed to make additional guesses for the genre of a song upon being told it was wrong, r_i would be the number of guesses needed for the classifier to get song A_i's genre correct. In that case, we can find a "rank accuracy" for our classifier by computing the following:

$$\frac{1}{N}\sum_{i=1}^{N}\left(1 - \frac{r_i - 1}{m - 1}\right)$$

For example, given that the classifier has 4 genres and ranks the correct genre for a song as number 3, then that song would contribute $1 - \frac{3-1}{4-1} = \frac{1}{3}$ to the total rank accuracy. A variation of this accuracy measure is shown in the "Rank Accuracy" results in section 5.2. Again, for comparison against a worst-case baseline, we can determine this expected rank accuracy for a random classifier:

$$E\left[\frac{1}{N}\sum_{i=1}^{N}\left(1-\frac{r_i-1}{m-1}\right)\right]=\frac{1}{N}\sum_{i=1}^{N}1+\frac{1}{N}\sum_{i=1}^{N}\frac{1}{m-1}-\frac{1}{N(m-1)}\sum_{i=1}^{N}E[r_i]$$

$$=1+\frac{1}{m-1}-\frac{1}{m(m-1)}\sum_{i=1}^{m}i=\frac{m(m-1)+m-\frac{m}{2}(m+1)}{m(m-1)}=\frac{1}{2}.$$

Both our classifiers' rank accuracies and the random classifier's rank accuracies are plotted in Figures 5.3.3 and 5.3.4.

As can be seen by all figures in section 5.3, our classifiers did much better than random chance and work well enough to save most users the headache involved with manually organizing their music.

5.2 Breakdown of Results

A detailed breakdown of all our results are given in this section. Sections labeled "Five Genres" refer to the genres Rock, Rap, Classical, Country, and Techno. Similarly, a section labeled "Three Genres" refers to Rock, Rap, and Classic al. A total of 485 rock, 789 rap, 168 classical, 445 country, and 425 techno songs were used for training for self-validation. The amount of testing music partitioned from the rest of the music for cross-validation was 51 rock, 40 rap, 24 classical, 49 country, and 72 techno.

***Naïve Bayesian Results*:**

Two Genres:

Self-Validation:

	Rock	Rap
Rock	0.92	0.08
Rap	0.07	0.93

Cross-Validation:

	Rock	Rap
Rock	0.90	0.10
Rap	0.10	0.90

Self-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap
1 (1)	0.92	0.93
2 (0)	0.08	0.07
Rank Acc:	**0.92**	**0.93**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap
1 (1)	0.90	0.90
2 (0)	0.10	0.10
Rank Acc:	**0.90**	**0.90**

Three Genres:

Self-Validation Accuracy:

	Rock	Rap	Classical
Rock	0.86	0.07	0.06
Rap	0.07	0.92	0.01
Classical	0.06	0.01	0.93

Cross-Validation Accuracy:

	Rock	Rap	Class.
Rock	0.88	0.10	0.02
Rap	0.10	0.90	0.00
Class.	0.13	0.00	0.88

Self-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.
1 (1)	0.86	0.92	0.93
2 (1/2)	0.12	0.06	0.07
3 (0)	0.01	0.01	0.00
Rank Acc:	**0.93**	**0.96**	**0.96**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.
1 (1)	0.88	0.90	0.88
2 (1/2)	0.12	0.10	0.13
3 (0)	0.00	0.00	0.00
Rank Acc:	**0.94**	**0.95**	**0.94**

Four Genres:

Self-Validation Accuracy:

	Rock	Rap	Class.	Count.
Rock	0.71	0.06	0.04	0.18
Rap	0.05	0.90	0.01	0.04
Class.	0.04	0.01	0.89	0.05
Count.	0.20	0.05	0.08	0.67

Cross-Validation Accuracy:

	Rock	Rap	Class.	Count.
Rock	0.75	0.08	0.02	0.16
Rap	0.08	0.90	0.00	0.03
Class.	0.13	0.00	0.88	0.00
Count.	0.20	0.06	0.16	0.57

Self-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.	Count.
1 (1)	0.71	0.90	0.89	0.67
2 (2/3)	0.20	0.04	0.06	0.31
3 (1/3)	0.08	0.05	0.05	0.01
4 (0)	0.01	0.01	0.00	0.00
Rank Acc:	**0.87**	**0.95**	**0.95**	**0.89**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.	Count.
1 (1)	0.75	0.90	0.88	0.57
2 (2/3)	0.20	0.03	0.00	0.43
3 (1/3)	0.06	0.08	0.13	0.00
4 (0)	0.00	0.00	0.00	0.00
Rank Acc:	**0.90**	**0.94**	**0.92**	**0.86**

Five Genres:

Self-Validation Accuracy:

	Rock	Rap	Class.	Count.	Tech.
Rock	0.63	0.06	0.04	0.18	0.09
Rap	0.03	0.88	0.01	0.04	0.04
Class.	0.04	0.01	0.89	0.05	0.00
Count.	0.18	0.04	0.08	0.67	0.02
Tech.	0.08	0.14	0.01	0.02	0.75

Cross-Validation Accuracy:

	Rock	Rap	Class.	Count.	Tech.
Rock	0.73	0.08	0.02	0.16	0.02
Rap	0.05	0.88	0.00	0.03	0.05
Class.	0.13	0.00	0.88	0.00	0.00
Count.	0.20	0.06	0.16	0.57	0.00
Tech.	0.07	0.17	0.00	0.13	0.64

Self-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.	Count.	Tech.
1 (1)	0.63	0.88	0.89	0.67	0.75
2 (3/4)	0.26	0.05	0.06	0.25	0.19
3 (1/2)	0.09	0.03	0.04	0.07	0.04
4 (1/4)	0.02	0.04	0.01	0.00	0.01
5 (0)	0.00	0.00	0.00	0.00	0.01
Rank Acc:	**0.87**	**0.94**	**0.96**	**0.90**	**0.91**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.	Count.	Tech.
1 (1)	0.73	0.88	0.88	0.57	0.64
2 (3/4)	0.18	0.03	0.00	0.35	0.24
3 (1/2)	0.08	0.03	0.08	0.08	0.06
4 (1/4)	0.02	0.08	0.04	0.00	0.04
5 (0)	0.00	0.00	0.00	0.00	0.03
Rank Acc:	**0.90**	**0.93**	**0.93**	**0.87**	**0.85**

***Full Covariance Bayesian Results*:**

Two Genres:

Self-Validation:

	Rock	Rap
Rock	0.92	0.08
Rap	0.07	0.93

Self-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap
1 (1)	0.92	0.93
2 (0)	0.08	0.07
Rank Acc:	**0.92**	**0.93**

Cross-Validation:

	Rock	Rap
Rock	0.90	0.10
Rap	0.13	0.88

Cross-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap
1 (1)	0.90	0.88
2 (0)	0.10	0.13
Rank Acc:	**0.90**	**0.88**

Three Genres:

Self-Validation Accuracy:

	Rock	Rap	Classical
Rock	0.90	0.07	0.02
Rap	0.07	0.93	0.01
Classical	0.03	0.00	0.97

Self-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.
1 (1)	0.90	0.93	0.97
2 (1/2)	0.09	0.07	0.02
3 (0)	0.00	0.01	0.01
Rank Acc:	**0.95**	**0.96**	**0.98**

Cross-Validation Accuracy:

	Rock	Rap	Classical
Rock	0.90	0.08	0.02
Rap	0.13	0.88	0.00
Classical	0.04	0.00	0.96

Cross-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.
1 (1)	0.90	0.88	0.96
2 (1/2)	0.08	0.13	0.00
3 (0)	0.02	0.00	0.04
Rank Acc:	**0.94**	**0.94**	**0.96**

Four Genres:

Self-Validation Accuracy:

	Rock	Rap	Class.	Count.
Rock	0.75	0.07	0.02	0.17
Rap	0.06	0.92	0.01	0.01
Class.	0.03	0.00	0.96	0.01
Count.	0.11	0.02	0.04	0.83

Cross-Validation Accuracy:

	Rock	Rap	Class.	Count.
Rock	0.80	0.08	0.00	0.12
Rap	0.13	0.88	0.00	0.00
Class.	0.04	0.00	0.96	0.00
Count.	0.18	0.06	0.10	0.65

Self-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.	Count.
1 (1)	0.75	0.92	0.96	0.83
2 (2/3)	0.23	0.06	0.02	0.14
3 (1/3)	0.02	0.02	0.01	0.03
4 (0)	0.00	0.00	0.01	0.00
Rank Acc:	**0.91**	**0.97**	**0.98**	**0.93**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.	Count.
1 (1)	0.80	0.88	0.96	0.65
2 (2/3)	0.18	0.10	0.00	0.27
3 (1/3)	0.00	0.03	0.00	0.08
4 (0)	0.02	0.00	0.04	0.00
Rank Acc:	**0.92**	**0.95**	**0.96**	**0.86**

Five Genres:

Self-Validation Accuracy:

	Rock	Rap	Class.	Count.	Tech.
Rock	0.73	0.06	0.02	0.17	0.02
Rap	0.06	0.90	0.01	0.01	0.03
Class.	0.03	0.00	0.96	0.01	0.00
Count.	0.11	0.02	0.04	0.83	0.01
Tech.	0.03	0.03	0.01	0.01	0.92

Cross-Validation Accuracy:

	Rock	Rap	Class.	Count.	Tech.
Rock	0.78	0.08	0.00	0.12	0.02
Rap	0.10	0.85	0.00	0.00	0.05
Class.	0.04	0.00	0.96	0.00	0.00
Count.	0.18	0.06	0.10	0.65	0.00
Tech.	0.11	0.06	0.00	0.04	0.79

Self-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.	Count.	Tech.
1 (1)	0.73	0.90	0.96	0.83	0.92
2 (3/4)	0.22	0.07	0.02	0.13	0.04
3 (1/2)	0.04	0.02	0.01	0.04	0.02
4 (1/4)	0.00	0.01	0.00	0.00	0.01
5 (0)	0.00	0.00	0.01	0.00	0.00
Rank Acc:	**0.92**	**0.97**	**0.98**	**0.95**	**0.97**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.	Count.	Tech.
1 (1)	0.78	0.85	0.96	0.65	0.79
2 (3/4)	0.16	0.08	0.00	0.22	0.06
3 (1/2)	0.04	0.08	0.00	0.10	0.07
4 (1/4)	0.02	0.00	0.00	0.02	0.07
5 (0)	0.00	0.00	0.04	0.00	0.01
Rank Acc:	**0.93**	**0.94**	**0.96**	**0.88**	**0.89**

***Support Vector Machine Results*:**

Two Genres:

Self-Validation:

	Rock	Rap
Rock	0.95	0.05
Rap	0.05	0.95

Cross-Validation:

	Rock	Rap
Rock	0.92	0.08
Rap	0.13	0.88

Self-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap
1 (1)	0.95	0.95
2 (0)	0.05	0.05
Rank Acc:	**0.95**	**0.95**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap
1 (1)	0.92	0.88
2 (0)	0.08	0.13
Rank Acc:	**0.92**	**0.88**

Three Genres:

Self-Validation Accuracy:

	Rock	Rap	Classical
Rock	0.93	0.05	0.02
Rap	0.05	0.95	0.00
Classical	0.01	0.00	0.99

Cross-Validation Accuracy:

	Rock	Rap	Classical
Rock	0.90	0.08	0.02
Rap	0.13	0.88	0.00
Classical	0.04	0.00	0.96

Self-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.
1 (1)	0.93	0.95	0.99
2 (1/2)	0.07	0.05	0.01
3 (0)	0.00	0.00	0.00
Rank Acc:	**0.96**	**0.97**	**0.99**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.
1 (1)	0.90	0.88	0.96
2 (1/2)	0.10	0.13	0.04
3 (0)	0.00	0.00	0.00
Rank Acc:	**0.95**	**0.94**	**0.98**

Four Genres:

Self-Validation Accuracy:

	Rock	Rap	Class.	Count.
Rock	0.79	0.04	0.01	0.16
Rap	0.05	0.94	0.00	0.02
Class.	0.01	0.00	0.96	0.03
Count.	0.16	0.01	0.04	0.79

Cross-Validation Accuracy:

	Rock	Rap	Class.	Count.
Rock	0.80	0.08	0.00	0.12
Rap	0.13	0.88	0.00	0.00
Class.	0.04	0.00	0.92	0.04
Count.	0.14	0.02	0.08	0.76

Self-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.	Count.
1 (1)	0.79	0.94	0.96	0.79
2 (2/3)	0.19	0.04	0.04	0.20
3 (1/3)	0.03	0.02	0.00	0.01
4 (0)	0.00	0.00	0.00	0.00
Rank Acc:	**0.92**	**0.97**	**0.99**	**0.93**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.	Count.
1 (1)	0.80	0.88	0.92	0.76
2 (2/3)	0.14	0.13	0.08	0.22
3 (1/3)	0.06	0.00	0.00	0.02
4 (0)	0.00	0.00	0.00	0.00
Rank Acc:	**0.92**	**0.96**	**0.97**	**0.91**

Five Genres:

Self-Validation Accuracy:

	Rock	Rap	Class.	Count.	Tech.
Rock	0.78	0.04	0.01	0.15	0.01
Rap	0.05	0.92	0.00	0.01	0.02
Class.	0.01	0.00	0.96	0.03	0.00
Count.	0.16	0.01	0.04	0.79	0.00
Tech.	0.03	0.03	0.00	0.01	0.92

Cross-Validation Accuracy:

	Rock	Rap	Class.	Count.	Tech.
Rock	0.78	0.08	0.00	0.12	0.02
Rap	0.13	0.78	0.00	0.00	0.10
Class.	0.04	0.00	0.92	0.04	0.00
Count.	0.14	0.02	0.08	0.76	0.00
Tech.	0.07	0.04	0.00	0.07	0.82

Self-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.	Count.	Tech.
1 (1)	0.78	0.92	0.96	0.79	0.92
2 (3/4)	0.19	0.05	0.04	0.19	0.05
3 (1/2)	0.04	0.02	0.00	0.01	0.01
4 (1/4)	0.00	0.01	0.00	0.01	0.01
5 (0)	0.00	0.00	0.00	0.00	0.00
Rank Acc:	**0.94**	**0.97**	**0.99**	**0.94**	**0.97**

Cross-Validation Rank Accuracy:

Rank (Wt)	Rock	Rap	Class.	Count.	Tech.
1 (1)	0.78	0.78	0.92	0.76	0.82
2 (3/4)	0.16	0.18	0.08	0.20	0.04
3 (1/2)	0.06	0.05	0.00	0.04	0.06
4 (1/4)	0.00	0.00	0.00	0.00	0.07
5 (0)	0.00	0.00	0.00	0.00	0.01
Rank Acc:	**0.93**	**0.93**	**0.98**	**0.93**	**0.90**

5.3 Results Summary

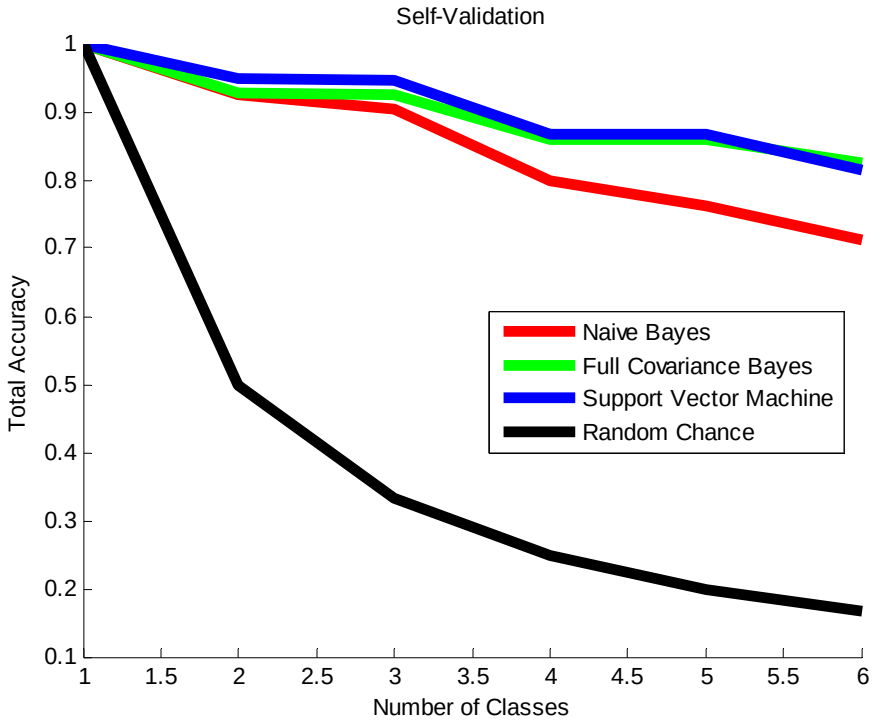

Figure 5.3.1: Regular Accuracy for Self-Validation Tests over Different Number of Genres

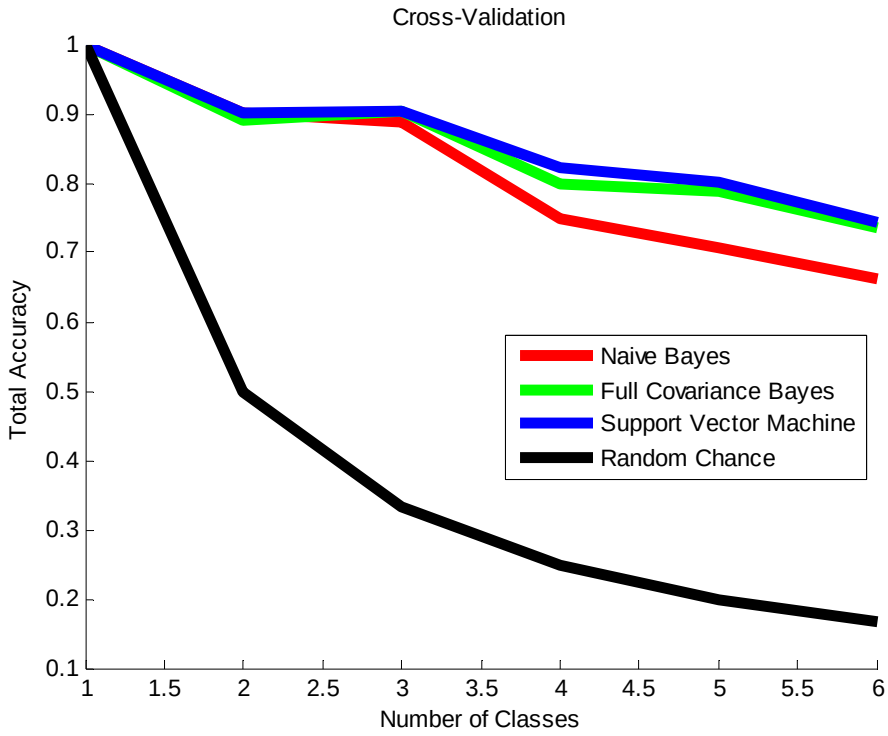

Figure 5.3.2: Regular Accuracy for Cross-Validation Tests over Different Number of Genres

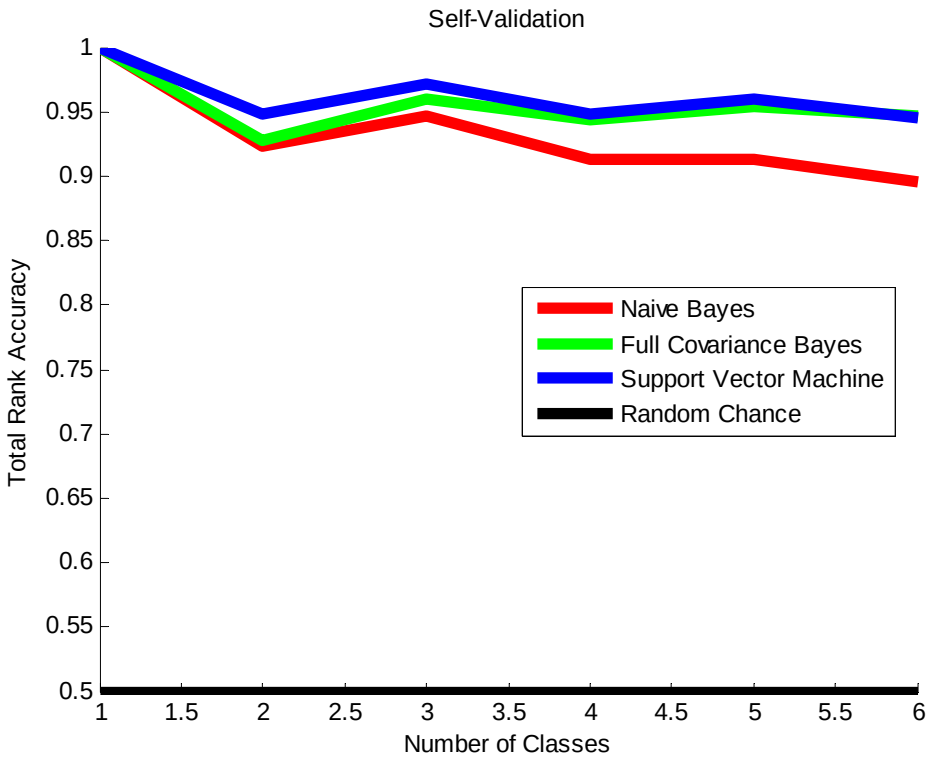

Figure 5.3.3: Rank Accuracy for Self-Validation Tests over Different Number of Genres

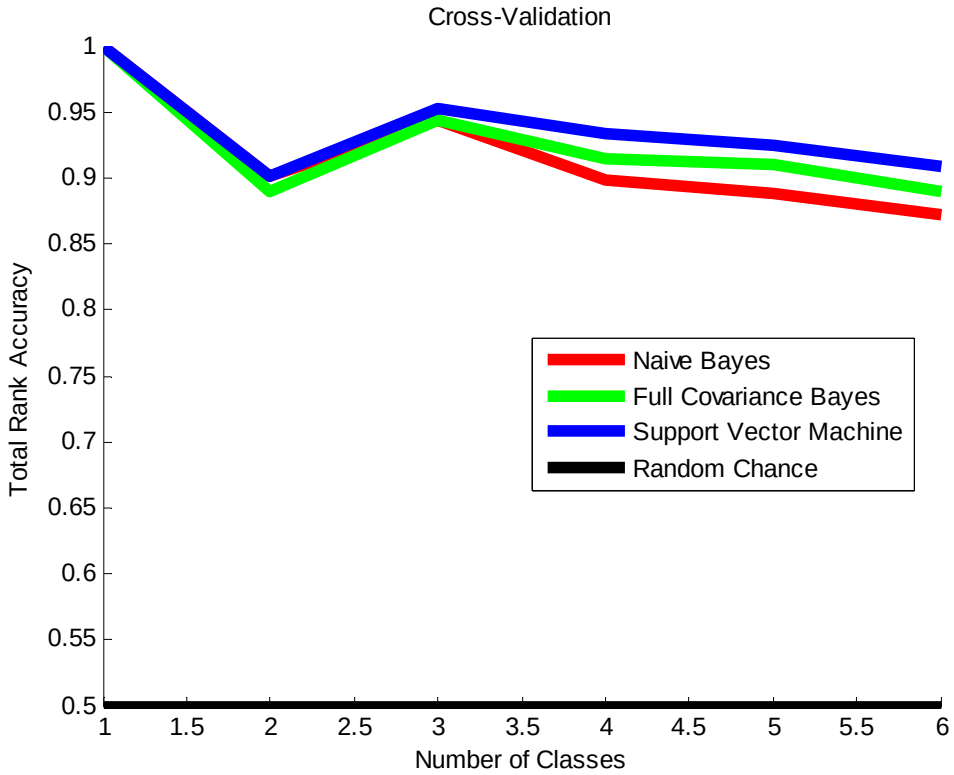

Figure 5.3.4: Rank Accuracy for Cross-Validation Tests over Different Number of Genres

5.4 Final Demo

For our final demo, we were able to showcase a working prototype of our automatic music classification application on the Android tablet. We ended up creating an application with several major functionalities but not the optimal classifier. Specifically we ran out of time to implement a full covariance Bayesian classifier on the tablet; for our final demo, the tablet contained a naïve Bayesian classifier that performs almost as well. The user can, first of all, look at a helpful tutorial to understand how to use our app.

Upon starting the app, a default classifier is quickly trained from a database of features collected from a website. The site's owners (we) can constantly update the website with features from new songs and new or old genres. If the user does not like the default given genres (rock, rap, classical, country, and techno), then he has several options. He can modify the downloaded database file directly to add or remove features from specific songs or genres. He can also decide to make his own classifier genres from scratch. All he has to do is place all his training music into separate folders based on genre. At the click of a button, the app can then extract features from these songs (about five seconds per song) and create classes named after the folders the songs were organized into. Finally, the user can also load an old, previously saved classifier if he doesn't like the default or custom ones. This optionality gives the user freedom to choose how to organize his library of music while maintaining the efficiency gained through automatic classification.

Once the user is satisfied with the genres being used by the classifier (which are updated readily on the main screen), he can find out the classifier's performance on an individual song. He will be able to view how confident the classifier is that the beginning, middle, and end portions of the song belong to any of the currently loaded genres. But testing a song on three different portions multiplies the computations by the same factor. If the user instead wants to organize hundreds or thousands of songs, he can click a button and let the classifier sort all music files in a directory to subdirectories labeled with the correct genre, based on just extracting features from the middle of the song. The user has an easy-to-use interface for browsing through the tablet's file system, one we based off of OI File Manager [5]. If the user mistakenly organizes his library based on the wrong genres, we also provide an "Unorganize" option, which undoes the organization by moving all the music files from the subdirectories back to the main directory.

Thus, the final demo accomplished our goal; we created an app that allows the user to easily sort his music however he wants with minimal effort.

6 TEAM LOGISTICS

Week of:	Apoorv:	Karanhaar:	Murium:
01/23:	Brainstorm ideas for the project (translating other language signs vs. music classification)	Brainstorm ideas for the project (translating other language signs vs. music classification)	Join the class
01/30:	Develop proposal for project	Develop proposal for project	Develop proposal for project
02/06:	Read Tzanakis paper	Read Tzanakis paper	Read Tzanakis paper
02/13:	Help finish timbral texture Matlab implementation	Help finish timbral texture Matlab implementation	Help finish timbral texture Matlab implementation
02/20:	Start rhythmic content implement.	Start rhythmic content implement.	Start rhythmic content implement.
02/27:	Test naïve Bayesian classifier w/ implemented feats.	Finish rhythmic content features	Help test classifier w/ implemented feats.
03/05:	Try naïve Bayesian classifier on Android	Try SVM classifier and full cov. On Matlab	Try naïve Bayesian on Android
03/12:	Research how to import WEKA onto Android	Collect results from Matlab implement.	Try classifier w/ different genre music
03/26:	Research how to import WEKA onto Android	Collect results from Matlab implementation	Test classifier w/ different genre music
04/02:	Implement the basic Android UI	Test the classifiers on data partitions	Implement the basic Android UI
04/09:	Try feature / classifier changes	Try out alternatives on Matlab	Add to design of Android UI
04/16:	Add to design of Android UI	Add to design of Android UI	Add to design of Android UI
04/23:	Finish project implement. On Android	Finish project implement. On Android	Finish project implement. On Android
04/30:	Default train reads from a website	Try neural network on Matlab	Finish w/ Android tutorial

Our fundamental breakdown of work was as follows. Karanhaar was in charge of implementing feature extraction and classifiers on Matlab, since he was also working on the code for a research project with Professor Stern. Apoorv helped with the Matlab implementation at roadblocks and figured out which machine learning algorithms to use for classification. Murium figured out how to make the Android UI user-friendly and how to port our project to the tablet. Our contributions were extremely mingled though; we often met for hours at a time to discuss problems and help each other get past each other's roadblocks.

7 FUTURE WORK

Given more time, we could experiment and implement several options for our application. Specifically, the next steps for our implementation would be to modify the features being extracted from music as well as to experiment with other classifiers for distinguishing the music.

To provide our classifier with more flexibility to adapt to new genres and music, many of our features can be generalized so as to provide the greatest distinction between genres. For example, our classifier can compute the spectral rolloff frequency for several thresholds instead of merely 85%, choosing the threshold which provides greatest distinction for the genres at hand. Another example of greater adaptability would be changing the weight of the one-pole low pass filter used while extracting rhythmic content features.

In terms of classifiers, we have successfully implemented the naive Bayesian, full covariance Bayesian, and support vector machine for our system. In the future, we could implement a working classifier using a neural network. Matlab has a Neural Networks toolbox which allows training and testing over various network parameters, such as the number of hidden layers of neurons. While we obtained preliminary results from using neural networks, this classifier certainly has room for further research for the purpose of music classification.

8 REFERENCES

[1] Tzanetakis, G.; Cook, P.; , "Musical genre classification of audio signals," *Speech and Audio Processing, IEEE Transactions on* , vol.10, no.5, pp. 293- 302, Jul 2002. doi: 10.1109/TSA.2002.800560

[2] L. Rabiner and B. H. Juang, Fundamentals of Speech Recogni-tion . Englewood Cliffs, NJ: Prentice-Hall, 1993.

[3] S. Davis and P. Mermelstein, "Experiments in syllable-based recognition of continuous speech,"IEEE Trans. Acoust., Speech, Signal Processing , vol. 28, pp. 357–366, Aug. 1980.

[4] E. Scheirer and M. Slaney, "Construction and evaluation of a robust multifeature speech/music discriminator," in Proc. Int. Conf. Acoustics, Speech, Signal Processing (ICASSP) , 1997, pp. 1331–1334.

[5] 2008, OI File Manager. http://code.google.com/p/openintents/source/browse/trunk/FileManager/src/org/openintents/filemanager/?r=1503.

[6] Nello Cristianini, John Shawe-Taylor, An introduction to Support Vector Machines: and other kernel-based learning methods, Cambridge University Press, New York, NY, 1999.

[7] Langley, Pat; Sage, Stephanie, "Induction of Selective Bayesian Classifiers," Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (1994). Seattle, WA: Morgan Kaufmann. http://www.isle.org/~langley/papers/select.uai94.pdf.

Special Thanks to Professor Richard M. Stern for introducing us to this interesting problem and spending time helping us understand the material behind the paper more thoroughly.