Name That Tune

AMath 582, Homework 5
Dan Jinguji, danjj@uw.edu

Due: Tuesday 13 March 2012

Abstract

This work examines the creation of a classifier using singular value decomposition (SVD) and linear discriminant analysis (LDA). The data classified are musical clips in varying genres. The clips have undergone time-frequency analysis using a modified Gábor filter (short-time discrete Fourier transform) to produce spectrograms of the musical clips. The goal of this work is to illustrate the flexibility of these techniques to produce analytically useful tools.

1 Introduction and Overview

This paper examines a discrimination task using linear discriminant analysis (LDA). The input data will be short musical clips from different genres and artists. The specific discrimination tasks will be to see how well the system does identifying artists and genres, given "appropriate" training data.

The work will progress in three separate tests:

1. Band Classification
2. The Case for Seattle
3. Genre Classification

In each of these tests, three different music sources (artist/composer) will be used to create the classifier using representative samples from the given source. The test cases, cases for classification, will be drawn from other works as described below.

In the first test, Band Classification, the three music sources will be from different genres, specifically: rag-time, early polyphony, and Seattle Grunge. The test cases will be drawn from different works of the same sources.

In the second test, The Case for Seattle, the three sources will be selected from the same genre, specifically, Seattle Grunge. The test cases will be drawn from the same sources as the training data.

The third test, Genre Classification, will use the same classifier as the first test. However, for this test, the test cases will be drawn from other sources in the same genres as the training data.

2 Theoretical Background

Linear discriminant analysis (LDA) attempts to create a classifier based on the features of a data set. Obviously, since this is a classifier, the desired output is categorical information, membership within one of a given set of categories, or classes. LDA is a technique that finds the linear combination of features which best distinguishes between the classes. The features can be any direct or derived measure taken from the data.

Minimally, one might chose the individual data points in a sound clip. A 5-second sound clip at the standard sampling rate of 44.1 kHz would give 220,500 data points. First, this number is quite unwieldy, but more importantly, this approach would do nothing to recognize the patterns within the data set. For example two samples of "pure" A-440, a generated sine wave of 440 Hz would not necessarily correlate with itself, if the inputs are out of phase. However, our perception of these sounds is that they are identical. We would hope our rigorous analysis would also lead to identifying them as exactly the same sound. So, rather than using the raw data, spectrograms of the sound clips will give time-frequency information ignoring things like phase differences.

A similar level of abstraction is needed to identify larger scale patterns within the spectrograms. Singular value decomposition (SVD) is a mathematical technique that transforms a series of observations onto an alternate orthogonal basis such that correlations within the data are reflected in the new orthogonal basis. This technique can be used to reduce the dimensionality of a data set. By identifying correlations within the data set, SVD could also identify large scale patterns within the data set. The implementation of SVD is built-in functionality in MATLAB.

So, the basic approach to the problem will be construct a classifier using a three-step process: first, extracting spectrograms from the sound clips; second, using SVD to identify patterns within the time-frequency data, the spectrograms; third, applying LDA to the transformed data. The output of LDA will be a sample space for the segregation of the different classes. The classification task will then be to extract the spectrogram from the test clips and project it onto the sample space identified by LDA to determine the membership of the test data.

LDA can be reduced to an eigenvector problem. Essentially, the problem is to find a projection of the data that maximizes the scatter between the classes (S_b) relative to the scatter within the classes (S_w).

$$S_b = \sum_{i=1}^{C} (\mu_i - \mu)(\mu_i - \mu)^T \tag{1}$$

and

$$S_w = \sum_{i=1}^{C} \sum_{j=1}^{M_i} (y_{i,j} - \mu_i)(y_{i,j} - \mu_i)^T \tag{2}$$

where: C is the number of classes; μ is the mean of the complete data set; μ_i is the mean of class i; M_i is the number of samples in class i; $y_{i,j}$ is the value of sample j within class i. The maximum is also a solution to a generalized eigenvector problem, so this can be solved using the built-in MATLAB eigenvector solver. This generated up to $C-1$ significant (non-zero) eigenvalues. The corresponding eigenvectors will transform the data into a space which provides maximum separation of the training data.

One primary concern is the upfront cost of creating the classifier. This is $O(n^3)$. The next section describes this process in greater detail, including steps taken to decrease the cost of creating the classifier.

3 Algorithmic Implementation and Development

The implementation of this work can be viewed segmentally. First, the process of obtaining the spectrogram data from each sound clip will be described. Then, there will be a discussion of the creation of the classifier from those spectrograms. This section closes by describing the method for using the classifier on the test data.

3.1 Creating Spectrograms

The sound clips will be analyzed using a short-time Fourier transform (STFT). The use of spectrograms will give frequency information, as well as time course of the signal.

A simple Gábor transform would provide a reasonable time-frequency analysis of the sound files. However, rather than using a Gaussian filter of the classic Gábor transform to select time slices, a modified Mexican Hat wavelet (Richter wavelet) was used, equation (3).

$$\psi(t) = \left(1 - \frac{t^2}{\sigma^2}\right) e^{\frac{-t^2}{2\sigma^2}} \tag{3}$$

The "real" Mexican Hat Wavelet, equation (4), has a scaling factor such that its integral from $-\infty$ to ∞ is 1. For this application, as a time-based signal filter, this scaling factor is not needed.

$$\psi(t) = \frac{2}{\sqrt{3\sigma}\pi^{\frac{1}{4}}} \left(1 - \frac{t^2}{\sigma^2}\right) e^{\frac{-t^2}{2\sigma^2}} \tag{4}$$

The wavelet filter was initially constructed using a σ value of 0.015. This results in an effective width for the signal of approximately 0.08 seconds, see Figure 1.[1] The sound clips are oversampled, with the temporal displacement of the filter at 0.02 seconds. This results

[1]The code that produced this image is found in Appendix B.3.

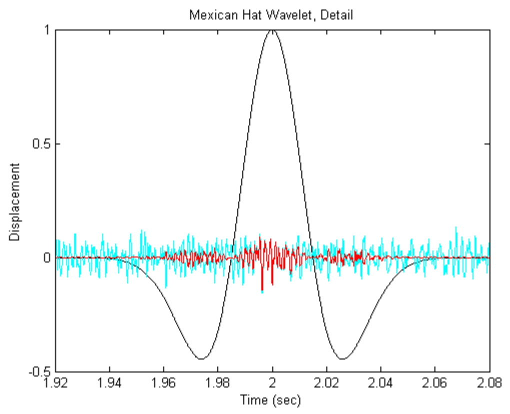


Figure 1: The Mexican Hat Wavelet Filter, with $\sigma = 0.015$

in frequency analysis 50 times per second. This captures most of the detail of the musical clips.

This results in a huge volume of data for each clip. For this analysis, the sound clips will be resampled at 8 kHz, rather than the industry standard of 44.1 kHz. We loose high-frequency input since the Nyquest frequency has been cut from 22.05 kHz to 4 kHz. This effectively decreases the size of the data set by a factor of just over 5.5. The low-fidelity version of the sound clips should be more than adequate for this classification task. In fact, previous work[2] which also analyzed musical clips using STFT indicated that the frequency for analysis could be capped at 1500 Hz and still capture the primary features of the input sound. Overall, this reduces the size of the dataset approximately 14.7 fold.

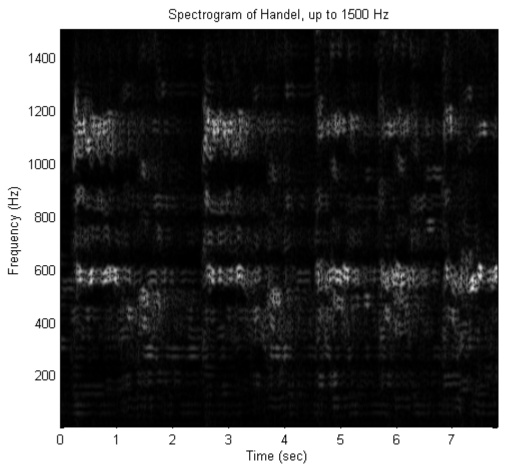
The written specification for this work describes 5-second musical clips. Communication on the discussion board for the assignment[3] indicated that slightly longer clips might prove helpful in increasing accuracy. So, these experiment began using 8-second clips. With oversampling collecting 50 short-time spectra per second, this results in 400 slices for each sound clip. Capping the frequency of interest at 1500 Hz means that the first 12,000 Fourier modes in each spectrum are being used, based on the sampling rate for the sound clips, 8 kHz.

A further reduction in the size of the data can be affected by "compressing" those 12,000 Fourier modes into 400 data points. Thus, the 12,000 modes can be segmented into 400 groups of 30 modes each. This results in each spectrogram being 400 by 400 data points. As shown in Figure 2[4], a very reasonable results seem to be obtained by taking the maximum value in each group. This preprocessing code for the sound clips is found in Section B.1

[2] "The Sound of Music", Homework 2

[3] `https://catalyst.uw.edu/gopost/area/kutz/94698`

[4] The code that generated these images is found in Appendix B.2.



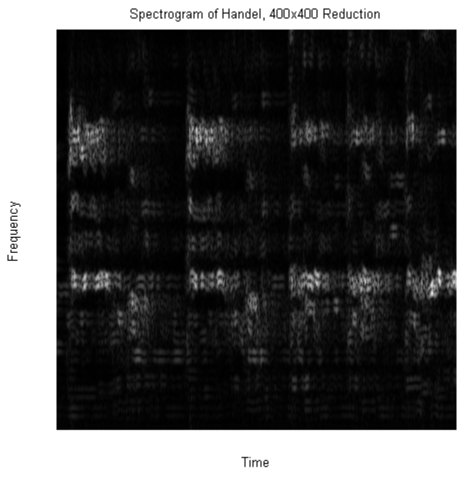


Figure 2: Comparison of a "True" Spectrum with the 400x400 Reduction

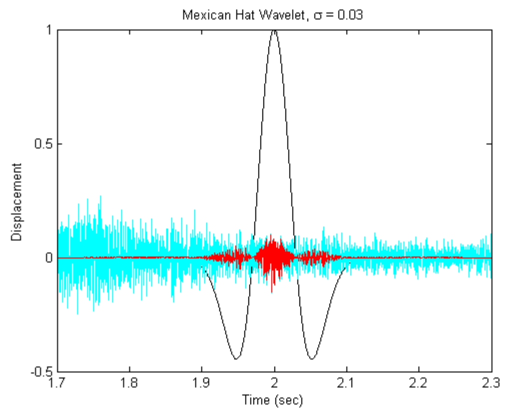


Figure 3: The Mexican Hat Wavelet Filter, with $\sigma = 0.03$

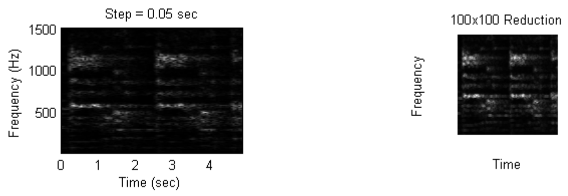


Figure 4: Comparison of a "True" Spectrum with the 100x100 Reduction

However, this proved to be too much data for creating the classifier. So, the preprocessing was amended to use a wider filter, $\sigma = 0.03$. This gives an effective signal width of approximately 0.2 seconds, as shown in Figure 3[5].

Using this wider wavelet filter allowed the sampling rate to drop to twenty frames per second. Changing the length of each sound clip to the suggested five seconds results in each spectrogram being 100 samples wide. By changing the size of the mode groups, the spectrogram could be reduced to 100 by 100 data points. The results are generally good, as shown in Figure 4[6]

This preprocessing code for the sound clips is found in Section B.6

[5]The code to create this image appears in Appendix B.5.

[6]The code that generated these images is found in Appendix B.4.

3.2 Sample Collection

A word is in order here about how the individual samples were collected and preprocessed. All of the sound clips used in this work were extracted from YouTube[7] using RealPlayer Downloader[8] on Firefox 9.0.1[9] This resulted in Flash Video (.flv) files. These were converted into MP3 files using VLC media player[10] version 2.0.0. The MP3 files were downsampled to 8 kHz using Audacity[11] version 1.2.6. Audacity was also used to edit the clips, for example, blank leader and trailer as well as other non-musical portions of the files, such as applause in live recordings.

The resulting .wav files are split into 5-second segments for creating the spectrograms as described in Section 3.1, code in Appendix B.6, up to twenty segments per file. The code that automates is shown in Appendix B.7. Many of the sound clips have an introductory section which is not always representative of the full sound file. Similarly, there is not infrequently a "coda" section at the end which may have a different character. To help account for this, the code calculates an initial offset into the .wav file, based on the length of the sound file when compared with the desired 100 seconds of data that will be extracted.

3.3 Creating the Classifier

The creation of the classifier was relatively straight-forward. The classifier supports a three-way distinction in classes. A singular value decomposition (SVD) is run on the training data from the three classes. This creates an alternate orthogonal basis for describing the data. Based on this new basis, a lower-dimensional representation of the data is extracted. Based on Fisher's work, a multi-class linear discriminant analysis (LDA) is run over this lower-dimensional representation of the data to provide $C-1$-dimensional space to serve as the basis for discrimination. The actual "heavy-lifting" is accomplished through two build-in MATLAB functions, `svd` and `eig`. However, there are two unresolved issues which must be addressed: first, what dimensionality is to be selected for the low-dimensional representation of the data; and second, given the mapping of the three classes in each experiment onto a plane, what are the criteria for assigning test data to classes. Let's examine these two issues separately.

[7] `http://www.youtube.com`, retrieved: 8 March 2012.

[8] `http://www.real.com/realplayer`, retrieved: 29 December 2007.

[9] `http://www.mozilla.org/en-US/firefox`, retrieved: 29 December 2011.

[10] `http://www.videolan.org/vlc`, retrieved: 2 March 2012.

[11] `http://audacity.sourceforge.net`, retrieved: 4 March 2011.

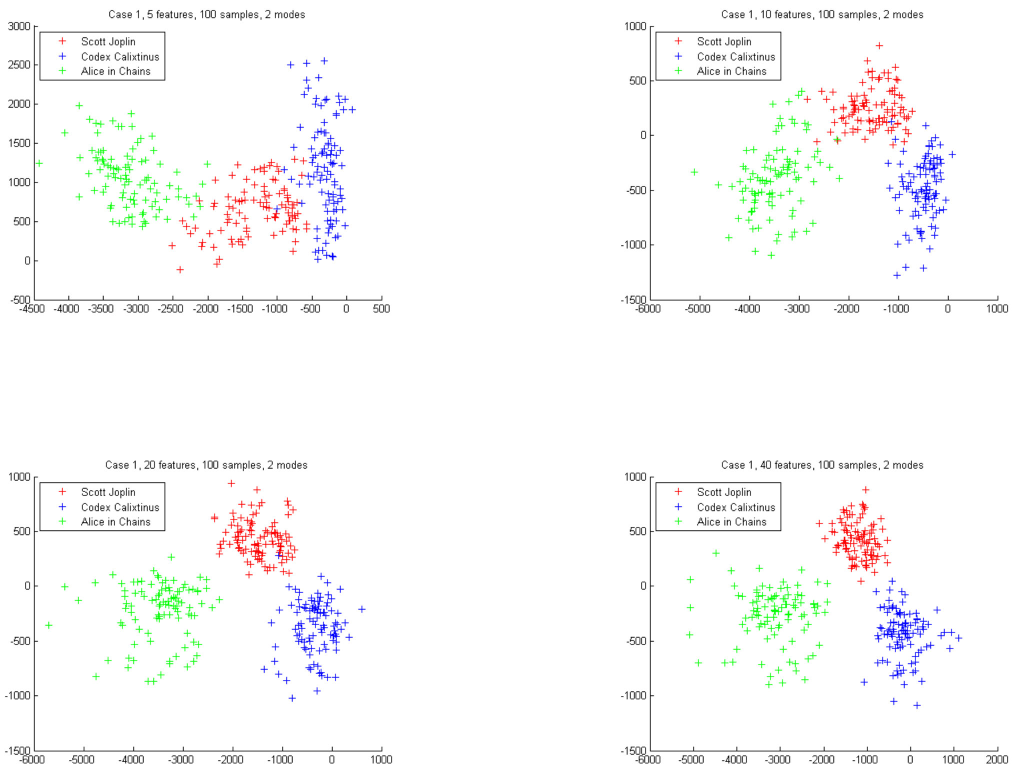


Figure 5: Effect of the Number of Features on Class Separation, set 1

3.3.1 Number of Features

It is not clear what the optimal number of features to include when creating the low-dimensional representation of the training data. Figures 5 and 6 show the kinds of variation in separation of the training data based on varying the dimensionality of the low-dimensional representation of the training data. One might expect at the separation to be relatively directly related to the number of features retained. In fact, there was communication to this effect on the class discussion board[12]. The data presented in Figure 5 is consistent with that conjecture. However, what is shown in Figure 6 is no long consistent with suggestion. Notice that with forty features (Figure 5, lower right) the separation of the three training data sets is complete. As the number of features retains continues to grow in Figure 6, the separate becomes markedly worse. Indeed when one hundred features are used in the low-dimensional representation we see the least separation found in any of the eight trials shown in Figures 5 and 6. I'm not sure what would account for this, but the empirical data presented here is clearly contrary to the notion that the more features one retains, the better the separation

[12]`https://catalyst.uw.edu/gopost/area/kutz/94698`

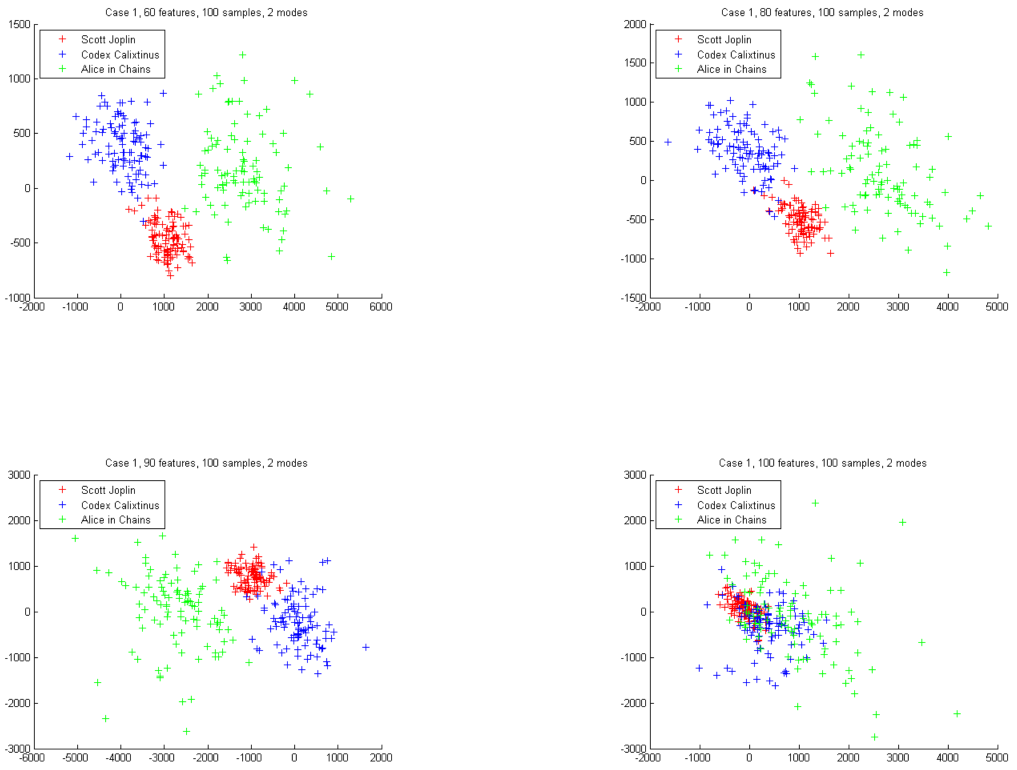


Figure 6: Effect of the Number of Features on Class Separation, set 2

of the classes.

3.3.2 Distinguishing the Classes

Even given "perfect" separation of the classes, as shown in Figure 7, there is no obvious the automated mechanism to divide the plane into three partitions for the three classes. The best approach would be to use a support vector machine (SVM). The approach would be to draw three lines, each of which separate two classes. The output class would be that class where two of the three classification lines "agree". A cartoon of how this scheme would work is shown in Figure 8. In this figure, the training data are represented by the ovals labeled "A", "B", and "C". There are lines separating each pair of classes. Areas where two of the classification lines give the same value are indicated with background shading. Notice that there is potentially an area where this scheme will fail since all three of the classification lines give different results. However, there is a greater problem. SVM support is provided within the Bioinformatics toolkit. So, some other mechanism for determining the lines for pair-wise class separation must be determined.

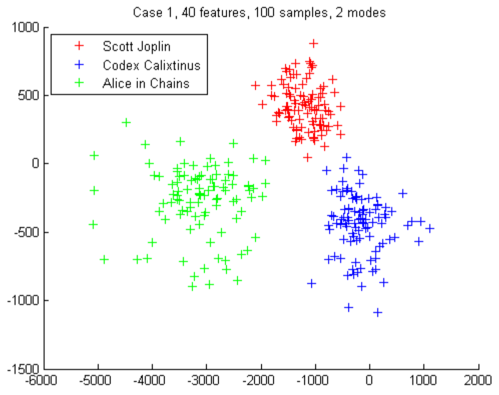


Figure 7: Effect of the Number of Features on Class Separation, set 1

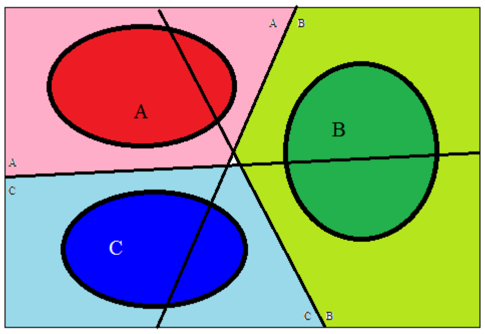


Figure 8: Scheme for Class Identification

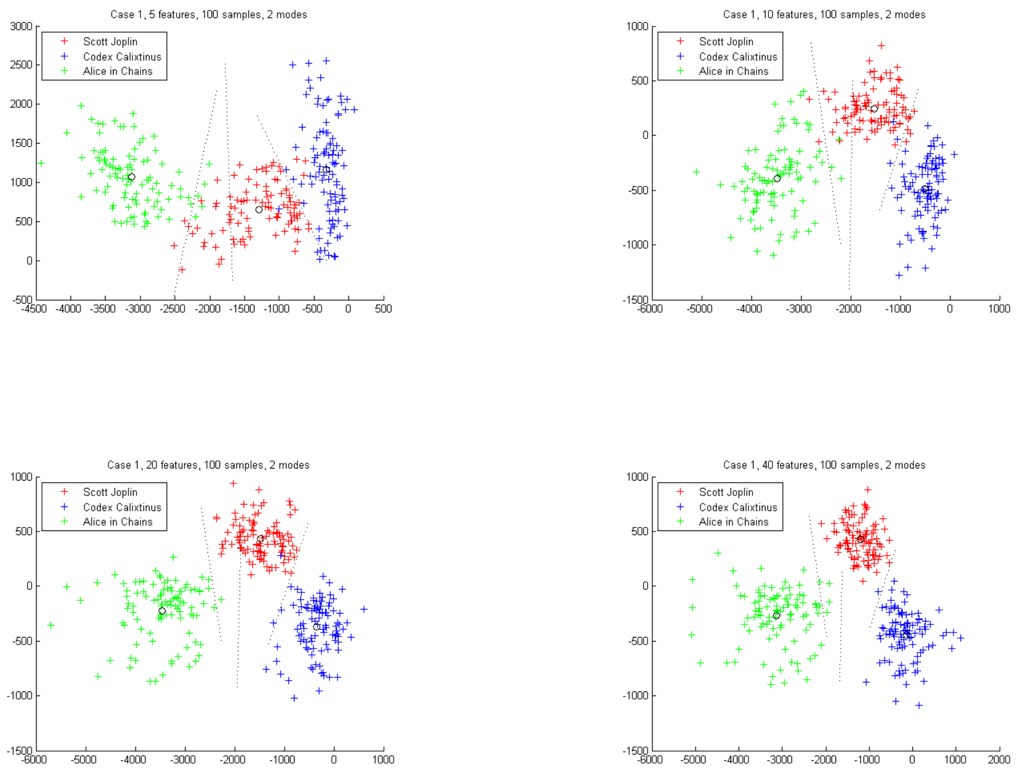


Figure 9: Examples of Using the Normal at the Midpoint for the Separation lines

The notion came to create an optimizer to draw the appropriate lines. The biggest issue is that the task is essentially discrete, and as such, does not lend itself to finding an extremum using the derivative. That is, one would not like a single very distant outliner to skew the location separation line. It should be counted as simply one error. Its distance from the separation line is immaterial.

A first approximation would use the centroids of the classes. The separation line would pass through the midpoint of the line joining the centroids of the two classes and be normal to that line.

This first approximation is actually not too bad, as shown in Figure 9[13]. This figure shows the same low-dimensional representations as Figure 5, but this time with marked centroids and the proposed separation lines. It's a reasonable first guess at a separation line.

There are two pieces of data that describe this line: the slope and the point that it intersects with the line connecting the centroids of the two training data sets. The process of improving this line would entail changing the slope of the separation line, in both the

[13]The lines do not appear normal to the line connecting the centroids because the horizonal and vertical scales in these plots are not equal.

clockwise and counter-clockwise directions, while retaining the intercept point until the score changes. Similarly, while keeping the slope constant, changing the intercept point until the score changes. If any of these four changes result in an increase in the score, those changes should be explored as potential ways to optimize the line. As noted earlier, probably the best metric for assessing the goodness of the separation line (score) is to count the number of correct identifications for the training date in the two classes upon which the line is based. This process would "walk up" to a local maximum. However, this task is too daunting to attempt for the current project.

However, rather than working on optimizing the separation lines, the initial guess of normal to the midpoint of the centroids will be used *as the separation line*. There is still the possibility of tuning the classifier by adjusting the dimensionality (number of features) of the low-dimensional representation of the training data. The metric for tuning is the total number of errors in classification using the training data. The number of features will be adjusted to minimize this value.

So, there is minimal optimization of the classifier by running the inexpensive LDA portion of the code for multiple different numbers of features and selecting the number of features based on the goodness of fit of the separation lines. The goodness of fit of the separation lines is determined by counting the number correct identifications using separator lines drawn normal to the midpoint of lines connecting the centroids of training data for each pair of classes in the training data. The code to classify the training data is found in Appendix B.9. The code to generate the classifier is found in Appendix B.8.

3.4 Using the Classifier

The use of the classifier is quite simple. When you run the hw5genclass method, you pass it three training data sets, one for each of the source classes. The method returns three things: the matrices w and U and a list of points *centers*. The transpose of the matrix U takes data (spectrograms) and projects them into the low-dimensional representation created using the SVD analysis of the training data. The transpose of the matrix w then projects the low-dimensional representation of the test data onto the plane identified by LDA which "optimal" separation of the three classes. The third return value is the list of centroids for the training data set within the LDA projection space.

Here is an example of creating the classifier:

$$[w\ U\ centers] = hw5genclass(train1, train2, train3) \tag{5}$$

Membership in the class is based on the algorithm described in Section 3.3.2. This is determined by the projection of the test dataset onto the LDA projection space. The location of the projected point relative to the lines normal to the midpoints of lines connecting the centroids provide class membership. This code is implemented in the MATLAB function hw5classify, shown in Appendix B.9. As noted above, this same function is used to assess the goodness of the number of features to use in creating the LDA classifier. The parameters for hw5classify are the list of centroids from the hw5getclass function and the data points for the test data after transforming them using the U and w matrices. The return value is a list of calculated class membership values for the test data.

Here is an example of classifying a set of test data:

$$results = hw5classify(centers, w' * (U' * test)) \tag{6}$$

The accuracy can be calculated by comparing the *results* with the expectations.

4 Computational Results

This section discusses the three experiments and their results.

4.1 Band Classification

For the first experiment, Band Classification, a classifier is created using three music sources of different genres. For this experiment, the three music sources are:

1. Rag-time Piano Music written by American composer Scott Joplin (1867 - 1917)
2. Early polyphonic choral music from the Codex Calixtinus (12th century Spanish manuscript)
3. Alice in Chains, Seattle grunge band, formed 1987

As noted in Section 3.2, 20 samples are extracted from each source recording. The individual records are listed in Appendix C.

The code for this experiment is found in Appendix B.10.

In this experiment, the classifier is created using 5 recordings per source, or 100 music clips for each source. This leaves two recordings for each source that can be used as test cases. One of sets of test cases was chosen because the selected recordings subjectively seem to be characteristic of the training data. The other set of test cases was chosen because there are subjective stylistic differences that may make the identification more challenging.

The output for this experiment is found in Appendix B.13. The salient summary information is repeated below:

```
Accuracy of training data
Class 1: 98 / 100, Class 2: 97 / 100, Class 3: 100 / 100
Accuracy of test set a, subjectively similar
Class 1: 17 / 20, Class 2: 19 / 20, Class 3: 20 / 20
Accuracy of test set b, subjectively a bit different
Class 1: 8 / 20, Class 2: 18 / 20, Class 3: 18 / 20
```

The classifier is very robust. In classifying the training data, it is correct in 295 out of the 300 training data clips. In some ways, this result is not too surprising since the classifier was tuned to the training data.

It performs very well with the data which is subjectively similar to the training data, getting 56 out of 60 samples correctly classified. The accuracy for this set of test data is 93.33%.

The performance on the subjectively different test data is still rather respectable. The classifier correctly classified 44 of the 60 test clips. This is an accuracy of 73.33%.

4.2 The Case of Seattle

For the second experiment: The Case of Seattle, a classifier is created using three music sources from the same genre. In keeping with the suggesting name for this experiment, the three sources are different bands in the Seattle grunge genre, specifically:

1. Alice in Chains, formed 1987
2. Soundgarden, formed 1984
3. Nirvana, formed 1987

As in Experiment 1, there are 5 recordings for training data from each band. The code for this experiment is found in Appendix B.11.

The code for this experiment is found in Appendix B.11.

The output for this experiment is found in Appendix B.14. The salient summary information is repeated below:

```
Accuracy of training data
Class 1: 96 / 100, Class 2: 94 / 100, Class 3: 95 / 100
Accuracy of test set a, subjectively similar
Class 1: 17 / 20, Class 2: 17 / 20, Class 3: 11 / 20
Accuracy of test set b, subjectively a bit different
Class 1: 8 / 20, Class 2: 9 / 20, Class 3: 5 / 20
```

As before, the training accuracy is expectedly high, 285 out of 300 samples, or 95.00%.

The performance on the subjectively similar test data is relatively good, 45 of the 60 test clips are correctly identified, for an accuracy of 75.00%. Not surprisingly, the accuracy here is less than for the samples drawn from different genres. In fact, because of the stylistic similarities of the input classes (bands), the similar and different test cases change. Perception is based on context.

For the subjectively stylistically different samples, the accuracy plummets precipitously. In this case, there are only 22 correctly identified of the 60 test clips, for an accuracy of 36.67%. In many ways, I suppose this is not surprising because something that seems stylistically different from the source group could well “sound” like a sample from a different class.

4.3 Genre Classification

In the third experiment, the same classifier that was used in the first experiment is now tested against test samples from the same genres but different sources.

As outlined in Section 4.1, the training data comes from: Scott Joplin, rag-time piano; 12th-century Spanish polyphony from the Codex Calixtinus; and Seattle grunge band, Alice in Chains.

The first test used other sources of rag-time music, specifically, Champagne Rag by Joseph Lamb; Cantina Band from Star Wars by John Williams; and Jazz me Blues played by the Original Dixieland Jazz Band. Surprisingly, the accuracy was very low. There were only 5 correct identification for Champagne Rag. Nothing else was correctly classified.

The second test case used other early polyphonic works, specifically, Kyrie from Missa L'homme armé by Dufay (Flemish, 1397 - 1474); Viderunt omnes by Pérotin (School of Notre Dame, c 1200); and Worldes blis ne last no throwe (England, c 1265). Here the accuracy was very good, 59 out of the 60 samples are correctly identified, for 98.33%.

The third test case used recordings from Experiment 2, so, other grunge band from Seattle. Here also the accuracy is quite good, 57 out of 60 are correctly classified, giving 95.00% accuracy.

The strikingly bad classification results in the rag-time test prompt a re-examination of what might be used in LDA. It does not seem that the salient feature of the musical clips is the macro-structure of the music. That is, aspects of the music which characterize genre, things such as tonality, harmonic rhythm, chordal progressions, and the like, do not seem to be used prominently by this classifier. If they were, one would hypothesize that the rag-time tests would have an accuracy much higher than 8.33%.

This suggests that the LDA may be based on other aspects of the music. One feature that differs significantly between these training data is timbre. The composition of the musical ensemble, for example, might be one of the key factors. Based on this, piano would imply Joplin, *a capella* voice would imply Codex Calixtinus, and guitars and drums would imply Alice in Chains.

To check this alternate notion about the features that are being selected by LDA, another set of second source test data was created. However, the selection criterion was not genre *per se*, but the overall sonic "impression" of the music.

The Joplin sound clips are primarily taken from piano rolls dating back to the early 20th century. So, the alternate Joplin recordings for this second test set are all taken from piano rolls, though stylistically not rag time at all. The three selections are "Beer Barrel Polka" from 1927, "New York New York" from the epinymous Scorsese movie of 1977, and a medley from the musical "Oliver" of 1960. The accuracy of the classifier was significantly higher, 47 correct out of 60 samples for 78.33%. Interestingly, each of these three recordings had higher accuracy than was obtained for the Joseph Lamb piano rag.

Similarly, for the alternate Codex Calixtinus recordings, other *a capella* choral works were chosen. These selections are: "Cherubic Hymn" by Bortniansky (1751 - 1825), "The Lion Sleeps Tonight" (1939) performed by Straight No Chaser, "Loch Lomond" (traditional Scottish folksong, first published in 1841) performed by Chanticleer. None of this is medieval or Renaissance polyphony. The accuracy for the first two recordings is quite good. Curiously, it is quite poor for the third recording. Overall, this set has 41 correct out of 60, or 68.33%. This prompted an additional case, “Locus Iste” by Anton Bruckner (1824-1896) where the accuracy was in line with the first two recordings, 19 out of 20 correctly classified.

Finally, for the alternate Alice in Chains recordings, other rock selections were used, specifically, “Runnin’ with the Devil”, Van Halen (1978); “Back in the USSR”, The Beatles (1968); and “Born in the USA”, Bruce Springsteen (1984). The results for these alternates was mixed. Overall, 29 out of 60 correctly classified, or 48.33%. The Van Halen and Springsteen did fair to middling, 13 and 14 out of 20, respectively. The Beatles, on the other hand, had only two correctly classified out of 20, 10.00%. Listening to the recordings again, the Van Halen and Springsteen both are much more prominent drums than The Beatles recording. So a fourth rock recording was added which has a more pronounced drum line, “Zvezda po imeni solntse” (The Star called Sun) by Soviet rock group “KINO” (1988). However, this recording received zero correct classifications out of its twenty clips.

The code for this experiment is found in Appendix B.12.

The output for this experiment is found in Appendix B.15. The salient summary information is repeated below:

```
Accuracy of training data
Class 1: 98 / 100, Class 2: 97 / 100, Class 3: 100 / 100
Accuracy of rag-time test data
Case 1: 5 / 20, Case 2: 0 / 20, Case 3: 0 / 20
Accuracy of early polyphony test data
Case 1: 20 / 20, Case 2: 19 / 20, Case 3: 20 / 20
Accuracy of grunge test data
Case 1: 20 / 20, Case 2: 17 / 20, Case 3: 20 / 20
Accuracy of player piano test data
Case 1: 20 / 20, Case 2: 11 / 20, Case 3: 16 / 20
Accuracy of acapella voice test data
Case 1: 17 / 20, Case 2: 19 / 20, Case 3: 2 / 20
Accuracy of rock test data
Case 1: 13 / 20, Case 2: 2 / 20, Case 3: 14 / 20
Accuracy of "additional" test data, Tsoy and Bruckner
Case 1: 0 / 20, Case 2: 19 / 20
```

5 Summary and Conclusions

The summary can be considered in two different ways: first, a summary for this particular exercise; second, a summary for the course overall.

As commentary on this particular exercise: It is gratifying to see how quickly additional data can be analyzed using this type of technique. The processing bottle-neck is in the creation of the SVD decomposition of the training data set. From this, it is relatively quick and easy to create a number of low-dimensional representations of the data and then process them using LDA to create a classifier. The classifier does a very reasonable job, correctly classifying a significant percentage of the test data. This test data also indicates that the features "selected" by LDA are not necessarily those which characterize the features upon which human perception of class membership is based. It would be interesting to further explore the nature of the errors in classification seen in these experiments. It is also quite likely that a more sophisticated means to move from the projected data to actual classification might increase the accuracy of these models.

This work outlines the creation of a relatively sophisticated classifier using various techniques that have been presented within the course of this class. It is helpful to see the application of these techniques, not only in isolation, but also used in conjunction with each other to formulate this type of larger solution.

A MATLAB Functions used and Brief implementation explanations

A table of the MATLAB functions which were used in this analysis.

Function	**Description**
xlabel	Sets the text for the label on the x-axis.
ylabel	Sets the text for the label on the y-axis.
title	Sets the text for the title of a plot.
length	Returns the size of a vector.
size	Returns the size of a matrix.
floor	Returns the largest integer value not greater than the argument.
pcolor	Plots a surface using pseudo-coloring.
colormap	Specifies the color scale to be used by pcolor.
sprintf	Formats output values into a string.
strcat	Concatenates multiple strings.
disp	Displays a line of text in the Command window.
zeros	Creates a matrix of the given size filled with zero.
exp	Calculates the exponential for each element in a matrix.
abs	Returns the norm of a vector.
max	Returns the maximum value of a vector.
reshape	Reorders the values within a matrix to fit new dimensions.
save	Saves variables to file for later use.
load	Loads values previous stored using save.
uint8	Converts values to unsigned 8-bit integer.
figure	Selects the output window for plotting.
imshow	Displays an image.
wavread	Loads the acoustic data from a .wav file.
fft	Returns the discrete Fourier transform of a vector.
plot	Draws a plot, of the given function.
mean	Calculates the mean value of a vector.
svd	Performs singular value decomposition on a matrix.
eig	Calculates the eigenvalues and eigenvectors for a matrix.

B MATLAB Code

B.1 Code to Generate the 400x400 Spectra

```
% generate the 400x400 spectrum and write it out
% file: hw5spectrum16.m

function out = hw5spectrum16(sound, name, freq, leng, incr, filesave)
```

```
% check the number of parameters
if nargin < 2
    disp('Usage: hw5spectrum(wavdata, name[, freq, leng, incr, filesave])')
    return
end

% fill in default values
if nargin < 3
    freq = 8000;
end
if nargin < 4
    leng = 8;
end
if nargin < 5
    incr = 30;
end
if nargin < 6
    filesave = 0;
end

% transpose the sound
v = sound';
count = length(v);
if count ~= leng*freq
    msg = sprintf('Warning: data for %s is not %d seconds', name, leng);
    disp(msg)
end
t2 = 0:1/freq:leng;
t = t2(1:count);
sigma = 0.015;
step = 0.01:0.02:7.99;
spec = zeros(400,400);

% Gabor transform on sound clip, using Mexican hat wavelet
tic;
for j = 1:length(step)
    t2 = (t-step(j)).^2;
    filt = (1 - t2/sigma^2).*exp(-t2/(2*sigma^2));
    s = v.*filt;
    st = fft(s);
    for k = 1:400
```

```
        spec(401-k,j) = max(abs(st(k*incr-(incr-1):k*incr)'));
    end
end
time1 = toc;
msg = sprintf('sound input %s: %.2f seconds', name, time1);
disp(msg)
out = reshape(spec*255/max(spec(:)),400*400, 1);

if filesave
    filename = sprintf('%s.txt', name);
    save(filename, 'spec', '-ascii')
end
```

B.2 Code for Figure 2

```
% compare "real" spectrogram and the reduction

% start clean
clear all; close all; clc;

load handel;
v = y(1:64000)'/2;

count = length(v);
end_time = count/Fs;
t2 = 0:1/Fs:end_time;
t = t2(1:count);
top = floor(count/2);

% mexican hat filter
width = .015;
sigma = width;
step_size = 0.020;

freq = (1:top) / end_time;
step = 0:step_size:end_time;
res = zeros(top,length(step));
tic;
for j = 1:length(step)
    t2 = (t-step(j)).^2;
    filt = (1 - t2/sigma^2).*exp(-t2/(2*sigma^2));
    s = v.*filt;
    st = fft(s);
    res(:,j) = abs(st(1:top)');
end
time1 = toc;

figure(1)
imshow(uint8(zeros(400,400)))
n1500 = find(freq>1500, 1);
freq2 = freq(1:n1500);
res2 = abs(res(1:n1500,:));
pcolor(step,freq2,res2), shading interp
colormap('gray')
xlabel('Time (sec)')
ylabel('Frequency (Hz)')
title('Spectrogram of Handel, up to 1500 Hz')

figure(2)
pic = hw5spectrum16(y(1:64000), 'handel');
imshow(uint8(reshape(pic, 400, 400)));
xlabel('Time')
ylabel('Frequency')
title('Spectrogram of Handel, 400x400 Reduction')
```

B.3 Code for Figure 1

```
% plot Mexican Hat wavelet

j = 101;
t2 = (t-step(j)).^2;
filt = (1 - t2/sigma^2).*exp(-t2/(2*sigma^2));
s = v.*filt;
figure(1),plot(t,filt,'k', t,v,'c', t,s,'r');
set(gca, 'xlim', [step(j-4) step(j+4)]);
xlabel('Time (sec)')
ylabel('Displacement')
title('Mexican Hat Wavelet, Detail')
```

B.4 Code for Figure 4

```
% compare the spectrum sampled at 0.05 seconds and the 100x100 reduction

load handel;
v = y(1:40000)'/2;
```

```
count = length(v);
end_time = count/Fs;
t2 = 0:1/Fs:end_time;
t = t2(1:count);
top = floor(count/2);

% mexican hat filter
width = .03;
sigma = width;
step_size = 0.050;

freq = (1:top) / end_time;
step = 0:step_size:end_time;
res = zeros(top,length(step));
tic;
for j = 1:length(step)
    t2 = (t-step(j)).^2;
    filt = (1 - t2/sigma^2).*exp(-t2/(2*sigma^2));
    s = v.*filt;
    st = fft(s);
    res(:,j) = abs(st(1:top)');
end
time1 = toc;

figure(1)
imshow(uint8(zeros(100,100)))
n1500 = find(freq>1500, 1);
freq2 = freq(1:n1500);
res2 = abs(res(1:n1500,:));
pcolor(step,freq2,res2), shading interp
colormap('gray')
xlabel('Time (sec)')
ylabel('Frequency (Hz)')
title('Step = 0.05 sec')

figure(2)
pic = hw5spectrum(y(1:40000), 'handel');
pic2 = uint8(reshape(pic, 100, 100));
imshow(pic2);
xlabel('Time')
ylabel('Frequency')
title('100x100 Reduction')
```

B.5 Code for Figure 3

```
% plot the wider Mexican Hat wavelet

j = 41;
sigma = 0.03;
t2 = (t-step(j)).^2;
filt = (1 - t2/sigma^2).*exp(-t2/(2*sigma^2));
s = v.*filt;
figure(4),plot(t,filt,'k', t,v,'c', t,s,'r');
set(gca, 'xlim', [step(j-6) step(j+6)]);
xlabel('Time (sec)')
ylabel('Displacement')
title('Mexican Hat Wavelet, \sigma = 0.03')
```

B.6 Code for Generate the 100x100 Spectra

```
% generate the 100x100 spectrum
% file: hw5spectrum.m

function out = hw5spectrum(sound, name, freq, leng, incr, filesave)

if nargin < 2
    disp('Usage: hw5spectrum(wavdata, name[, freq, leng, incr, filesave])')
    return
end

if nargin < 3
    freq = 8000;
end
if nargin < 4
    leng = 5;
end
if nargin < 5
    incr = 72;
end
if nargin < 6
    filesave = 0;
end

% transpose the sound
v = sound';
count = length(v);
```

```
if count ~= leng*freq
    msg = sprintf('Warning: data for %s is not %d seconds', name, leng);
    disp(msg)
end
t2 = 0:1/freq:leng;
t = t2(1:count);
sigma = 0.03;
step = 0.025:0.05:4.975;
spec = zeros(100,100);

% Gabor transform on sound clip, using Mexican hat wavelet
tic;
for j = 1:length(step)
    t2 = (t-step(j)).^2;
    filt = (1 - t2/sigma^2).*exp(-t2/(2*sigma^2));
    s = v.*filt;
    st = fft(s);
    for k = 1:100
        spec(101-k,j) = max(abs(st(k*incr-(incr-1):k*incr)'));
    end
end
time1 = toc;
msg = sprintf('sound input %s: %.2f seconds', name, time1);
disp(msg)
out = reshape(spec*255/max(spec(:)),100*100, 1);

if filesave
    filename = sprintf('%s.txt', name);
    save(filename, 'spec', '-ascii')
end
```

B.7 Code for Extracting Spectrograms from .wav Files

```
% code to create the spectrograms.
% file: hw5makeclips.m

function out = hw5makeclips(file)

input = wavread(file);
len = length(input(:,2));
ct = floor(len / (5*8000));
if ct > 20
    ct = 20;
```

```
end
if len < 5*8000*20
    msg1 = sprintf('file %s is too short to create 20 5-second clips', file);
    msg2 = sprintf(', creating %d instead', ct);
    msg = strcat(msg1, msg2);
    disp(msg)
end

% set the amount of time to skip
if len > 5*8000*20 + 60*8000
    start = 30*8000;
elseif len > 5*8000*20 + 20*8000
    start = 10*8000;
elseif len > 5*8000*20 + 10*8000
    start = 5*8000;
else
    start = 0;
end

out = zeros(100*100, ct);

for j = 1:ct

    num = sprintf('%s, clip %d', file, j);
    beg = 1 + start + 5*8000 * (j-1);
    fin = start + 5*8000 * j;
    wav = input(beg:fin, 2);
    out(:,j) = hw5spectrum(wav, num);

end
```

B.8 Code for Generating the Classifier

```
% the LDA classifier
% file: hw5genclass.m

function [w, U, centers] = hw5genclass(cat1, cat2, cat3)

% pre-checking
len1 = length(cat1(1,:));
len2 = length(cat2(1,:));
len3 = length(cat3(1,:));
len = len1 + len2 + len3;
```

```
if len1 ~= len2 || len1 ~= len3 || len2 ~= len3
    msg = sprintf('Warning: Vectors not the same length: %d %d %d', len1, len2, len3);
    disp(msg)
end

% svd the data
tic
[U S V] = svd([ cat1 cat2 cat3 ], 0);
timesvd = toc;
msg = sprintf('svd took %.2f seconds', timesvd);
disp(msg);

% create list of potential numbers of features
features = 5:5:len1;
l = length(features);
right = zeros(1,l);

% loop through the features
for k=1:l

    f = features(k);

    % perform LDA using the current feature count
    tunes = S * V';
    tunes = tunes(1:f,:);
    vec1 = tunes(:,1:len1);
    vec2 = tunes(:,1+len1:len1+len2);
    vec3 = tunes(:,1+len1+len2:len1+len2+len3);
    mean1 = mean(vec1,2);
    mean2 = mean(vec2,2);
    mean3 = mean(vec3,2);
    mall = mean(tunes,2);

    % calculate S_w, scatter within classes
    Sw = zeros(f,f);
    for j=1:len1
        Sw = Sw + (vec1(:,j)-mean1)*(vec1(:,j)-mean1)';
    end
    for j=1:len2
        Sw = Sw + (vec2(:,j)-mean2)*(vec2(:,j)-mean2)';
    end
```

```
for j=1:len3
    Sw = Sw + (vec2(:,j)-mean3)*(vec3(:,j)-mean3)';
end

% calculate S_b, scatter between classes
Sb = (mean1-mall)*(mean1-mall)';
Sb = Sb + (mean2-mall)*(mean2-mall)';
Sb = Sb + (mean3-mall)*(mean3-mall)';

% solve the eigenvector problem
[eV, D] = eig(Sb, Sw);

% select the two "non-zero" eigenvalues
d = abs(diag(D));
modes = find(d>0.001);
l = length(modes);
% construct the mapping matrix
w = zeros(f,l);
for j=1:l
    w(:,j) = eV(:,modes(j));
end

% map the input points
pt1 = w'*vec1;
pt2 = w'*vec2;
pt3 = w'*vec3;

% calculate the means
c1 = mean(pt1,2);
c2 = mean(pt2,2);
c3 = mean(pt3,2);
centers = [c1 c2 c3];

% check accuracy of training data
res1 = hw5classify(centers,pt1);
res2 = hw5classify(centers,pt2);
res3 = hw5classify(centers,pt3);
r = length(find(res1==1)) + length(find(res2==2)) + length(find(res3==3));
right(k) = r;
msg = sprintf('%d features, %d modes, %d correct of %d', f, l, r, len);
disp(msg)
```

```
end

% select the "best" fit number of features
r = features(right == max(right));
f = r(1);
msg = sprintf('feature count: %d', f);
disp(msg);

% create low-dimensional representation matrix
U = U(:,1:f);
% prepare data for LDA
tunes = S * V';
tunes = tunes(1:f,:);
vec1 = tunes(:,1:len1);
vec2 = tunes(:,1+len1:len1+len2);
vec3 = tunes(:,1+len1+len2:len1+len2+len3);
mean1 = mean(vec1,2);
mean2 = mean(vec2,2);
mean3 = mean(vec3,2);
mall = mean(tunes,2);

% calculate S_w, scatter within classes
Sw = zeros(f,f);
for j=1:len1
    Sw = Sw + (vec1(:,j)-mean1)*(vec1(:,j)-mean1)';
end
for j=1:len2
    Sw = Sw + (vec2(:,j)-mean2)*(vec2(:,j)-mean2)';
end
for j=1:len3
    Sw = Sw + (vec2(:,j)-mean3)*(vec3(:,j)-mean3)';
end

% calculate S_b, scatter between classes
Sb = (mean1-mall)*(mean1-mall)';
Sb = Sb + (mean2-mall)*(mean2-mall)';
Sb = Sb + (mean3-mall)*(mean3-mall)';

% solve eigenvector problem
[eV, D] = eig(Sb, Sw);

% find the non-trivial solutions
```

```
d = abs(diag(D));
modes = find(d>0.001);
l = length(modes);

% construct the projection matrix
w = zeros(f,l);
for j=1:l
    w(:,j) = eV(:,modes(j));
end

% calculate the centers of the classes
pt1 = w'*vec1;
pt2 = w'*vec2;
pt3 = w'*vec3;

c1 = mean(pt1,2);
c2 = mean(pt2,2);
c3 = mean(pt3,2);

centers = [c1 c2 c3];
```

B.9 Code for the Classification Algorithm

```
% classify the data points
% based on normals to the midpoints of the centrolds
% file: hw5classify.m

function results = hw5classify(centers, pts)

% separate the centers
c1 = centers(:,1);
c2 = centers(:,2);
c3 = centers(:,3);

l = length(pts(1,:));
counts = zeros(3,l);

% compare c1 and c2
slope = (c1(1)-c2(1))/(c2(2)-c1(2));
center = mean([c1 c2], 2);
ref1 = slope*(c1(1) - center(1)) + center(2) - c1(2);
for j=1:l
    val = slope*(pts(1,j) - center(1)) + center(2) - pts(2,j);
    if val*ref1 > 0
        counts(1,j) = counts(1,j)+1;
    else
        counts(2,j) = counts(2,j)+1;
    end
end

% compare c1 and c3
slope = (c1(1)-c3(1))/(c3(2)-c1(2));
center = mean([c1 c3], 2);
ref1 = slope*(c1(1) - center(1)) + center(2) - c1(2);
for j=1:l
    val = slope*(pts(1,j) - center(1)) + center(2) - pts(2,j);
    if val*ref1 > 0
        counts(1,j) = counts(1,j)+1;
    else
        counts(3,j) = counts(3,j)+1;
    end
end

% compare c2 and c3
slope = (c2(1)-c3(1))/(c3(2)-c2(2));
center = mean([c2 c3], 2);
ref1 = slope*(c2(1) - center(1)) + center(2) - c2(2);
for j=1:l
    val = slope*(pts(1,j) - center(1)) + center(2) - pts(2,j);
    if val*ref1 > 0
        counts(2,j) = counts(2,j)+1;
    else
        counts(3,j) = counts(3,j)+1;
    end
end

% assign the class based on the counts
results = 1*(counts(1,:)==2) + 2*(counts(2,:)==2) + 3*(counts(3,:)==2);
```

B.10 Code for Experiment 1: Band Classification

```
% experiment 1: band classification
% file: hw5exp1.m

clear all; close all; clc;
```

```
% load the data files (spectrograms)
tic
load joplin
train1 = [ joplin2 joplin3 joplin6 joplin7 joplin8 ];
test1a = joplin4 ;
test1b = joplin5 ;
load calix
train2 = [ calix1 calix3 calix4 calix5 calix7 ];
test2a = calix6 ;
test2b = calix2 ;
load alice
train3 = [ alice1 alice2 alice4 alice5 alice6 ];
test3a = alice7 ;
test3b = alice3 ;
timeload = toc;
msg = sprintf('loading data took %.2f seconds', timeload);
disp(msg);

% generate the classifier
tic
[w, U, centers] = hw5genclass(train1, train2, train3);
timesvd = toc;
msg = sprintf('svd/lda took %.2f seconds', timesvd);
disp(msg);

% report on accuracy of the training data
disp('Accuracy of training data');
res1 = hw5classify(centers, w'*(U'*train1));
res2 = hw5classify(centers, w'*(U'*train2));
res3 = hw5classify(centers, w'*(U'*train3));
len1 = length(train1(1,:));
len2 = length(train2(1,:));
len3 = length(train3(1,:));
right1 = length(find(res1==1));
right2 = length(find(res2==2));
right3 = length(find(res3==3));

fmt = 'Class 1: %d / %d, Class 2: %d / %d, Class 3: %d / %d';
msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);

% report on accuracy of test set a
```

```
disp('Accuracy of test set a, subjectively similar');
res1 = hw5classify(centers, w'*(U'*test1a));
res2 = hw5classify(centers, w'*(U'*test2a));
res3 = hw5classify(centers, w'*(U'*test3a));
len1 = length(test1a(1,:));
len2 = length(test2a(1,:));
len3 = length(test3a(1,:));
right1 = length(find(res1==1));
right2 = length(find(res2==2));
right3 = length(find(res3==3));

msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);

% report on accuracy of test set b
disp('Accuracy of test set b, subjectively a bit different');
res1 = hw5classify(centers, w'*(U'*test1b));
res2 = hw5classify(centers, w'*(U'*test2b));
res3 = hw5classify(centers, w'*(U'*test3b));
len1 = length(test1b(1,:));
len2 = length(test2b(1,:));
len3 = length(test3b(1,:));
right1 = length(find(res1==1));
right2 = length(find(res2==2));
right3 = length(find(res3==3));

msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);
```

B.11 Code for Experiment 2: The Case of Seattle

```
% experiment 2: The Case of Seattle
% file: hw5exp2.m

clear all; close all; clc;

% load the data files (spectrograms)
tic
load sg
train1 = [ sg4 sg5 sg6 sg7 sg8 ];
test1a = sg3 ;
test1b = sg1 ;
load nirvana
```

```
train2 = [ nirvana1 nirvana2 nirvana3 nirvana5 nirvana7 ];
test2a = nirvana4 ;
test2b = nirvana8 ;
load alice
train3 = [ alice1 alice2 alice4 alice5 alice6 ];
test3a = alice3 ;
test3b = alice7 ;
timeload = toc;
msg = sprintf('loading data took %.2f seconds', timeload);
disp(msg);

% generate the classifier
tic
[w, U, centers] = hw5genclass(train1, train2, train3);
timesvd = toc;
msg = sprintf('svd/lda took %.2f seconds', timesvd);
disp(msg);

% report on accuracy of the training data
disp('Accuracy of training data');
res1 = hw5classify(centers, w'*(U'*train1));
res2 = hw5classify(centers, w'*(U'*train2));
res3 = hw5classify(centers, w'*(U'*train3));
len1 = length(train1(1,:));
len2 = length(train2(1,:));
len3 = length(train3(1,:));
right1 = length(find(res1==1));
right2 = length(find(res2==2));
right3 = length(find(res3==3));

fmt = 'Class 1: %d / %d, Class 2: %d / %d, Class 3: %d / %d';
msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);

% report on accuracy of test set a
disp('Accuracy of test set a, subjectively similar');
res1 = hw5classify(centers, w'*(U'*test1a));
res2 = hw5classify(centers, w'*(U'*test2a));
res3 = hw5classify(centers, w'*(U'*test3a));
len1 = length(test1a(1,:));
len2 = length(test2a(1,:));
len3 = length(test3a(1,:));
```

```
right1 = length(find(res1==1));
right2 = length(find(res2==2));
right3 = length(find(res3==3));

msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);

% report on accuracy of test set b
disp('Accuracy of test set b, subjectively a bit different');
res1 = hw5classify(centers, w'*(U'*test1b));
res2 = hw5classify(centers, w'*(U'*test2b));
res3 = hw5classify(centers, w'*(U'*test3b));
len1 = length(test1b(1,:));
len2 = length(test2b(1,:));
len3 = length(test3b(1,:));
right1 = length(find(res1==1));
right2 = length(find(res2==2));
right3 = length(find(res3==3));

msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);
```

B.12 Code for Experiment 3: Genre Classification

```
% experiment 3: genre classification
% file: hw5exp1.m

clear all; close all; clc;

% load the training data files (spectrograms)
tic
load joplin
train1 = [ joplin2 joplin3 joplin6 joplin7 joplin8 ];
load calix
train2 = [ calix1 calix3 calix4 calix5 calix7 ];
load alice
train3 = [ alice1 alice2 alice4 alice5 alice6 ];
timeload = toc;
msg = sprintf('loading data took %.2f seconds', timeload);
disp(msg);

% generate the classifier
tic
```

```
[w, U, centers] = hw5genclass(train1, train2, train3);
timesvd = toc;
msg = sprintf('svd/lda took %.2f seconds', timesvd);
disp(msg);

% report on accuracy of the training data
disp('Accuracy of training data');
res1 = hw5classify(centers, w'*(U'*train1));
res2 = hw5classify(centers, w'*(U'*train2));
res3 = hw5classify(centers, w'*(U'*train3));
len1 = length(train1(1,:));
len2 = length(train2(1,:));
len3 = length(train3(1,:));
right1 = length(find(res1==1));
right2 = length(find(res2==2));
right3 = length(find(res3==3));

fmt = 'Class 1: %d / %d, Class 2: %d / %d, Class 3: %d / %d';
msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);

% report on accuracy of rag-time test data
disp('Accuracy of rag-time test data');
load rag
res1 = hw5classify(centers, w'*(U'*rag1));
res2 = hw5classify(centers, w'*(U'*rag2));
res3 = hw5classify(centers, w'*(U'*rag3));
len1 = length(rag1(1,:));
len2 = length(rag2(1,:));
len3 = length(rag3(1,:));
right1 = length(find(res1==1));
right2 = length(find(res2==1));
right3 = length(find(res3==1));

fmt = 'Case 1: %d / %d, Case 2: %d / %d, Case 3: %d / %d';
msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);

% report on accuracy of early polyphonic test data
disp('Accuracy of early polyphony test data');
load poly
res1 = hw5classify(centers, w'*(U'*poly1));
```

```
res2 = hw5classify(centers, w'*(U'*poly2));
res3 = hw5classify(centers, w'*(U'*poly3));
len1 = length(poly1(1,:));
len2 = length(poly2(1,:));
len3 = length(poly3(1,:));
right1 = length(find(res1==2));
right2 = length(find(res2==2));
right3 = length(find(res3==2));

msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);

% report on accuracy of grunge test data
disp('Accuracy of grunge test data');
load sg
load nirvana
res1 = hw5classify(centers, w'*(U'*sg4));
res2 = hw5classify(centers, w'*(U'*nirvana4));
res3 = hw5classify(centers, w'*(U'*sg3));
len1 = length(sg4(1,:));
len2 = length(nirvana4(1,:));
len3 = length(sg3(1,:));
right1 = length(find(res1==3));
right2 = length(find(res2==3));
right3 = length(find(res3==3));

msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);

% report on accuracy of player piano test data
disp('Accuracy of player piano test data');
load piano
res1 = hw5classify(centers, w'*(U'*piano1));
res2 = hw5classify(centers, w'*(U'*piano2));
res3 = hw5classify(centers, w'*(U'*piano3));
len1 = length(piano1(1,:));
len2 = length(piano2(1,:));
len3 = length(piano3(1,:));
right1 = length(find(res1==1));
right2 = length(find(res2==1));
right3 = length(find(res3==1));
```

```
msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);

% report on accuracy of acapella voice test data
disp('Accuracy of acapella voice test data');
load voice
res1 = hw5classify(centers, w'*(U'*voice1));
res2 = hw5classify(centers, w'*(U'*voice2));
res3 = hw5classify(centers, w'*(U'*voice3));
len1 = length(voice1(1,:));
len2 = length(voice2(1,:));
len3 = length(voice3(1,:));
right1 = length(find(res1==2));
right2 = length(find(res2==2));
right3 = length(find(res3==2));

msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);

% report on accuracy of rock test data
disp('Accuracy of rock test data');
load rock
res1 = hw5classify(centers, w'*(U'*rock1));
res2 = hw5classify(centers, w'*(U'*rock2));
res3 = hw5classify(centers, w'*(U'*rock3));
len1 = length(rock1(1,:));
len2 = length(rock2(1,:));
len3 = length(rock3(1,:));
right1 = length(find(res1==3));
right2 = length(find(res2==3));
right3 = length(find(res3==3));

msg = sprintf(fmt, right1, len1, right2, len2, right3, len3);
disp(msg);

% report on accuracy of additional test data
disp('Accuracy of "additional" test data, Tsoy and Bruckner');
load rock
res1 = hw5classify(centers, w'*(U'*rock4));
res2 = hw5classify(centers, w'*(U'*voice4));
len1 = length(rock4(1,:));
len2 = length(voice4(1,:));
```

```
right1 = length(find(res1==3));
right2 = length(find(res2==2));

fmt = 'Case 1: %d / %d, Case 2: %d / %d';
msg = sprintf(fmt, right1, len1, right2, len2);
disp(msg);
```

B.13 Output for Experiment 1: Band Classification

```
loading data took 1.05 seconds
svd took 5.63 seconds
5 features, 2 modes, 281 correct of 300
10 features, 2 modes, 294 correct of 300
15 features, 2 modes, 294 correct of 300
20 features, 2 modes, 290 correct of 300
25 features, 2 modes, 289 correct of 300
30 features, 2 modes, 292 correct of 300
35 features, 2 modes, 294 correct of 300
40 features, 2 modes, 295 correct of 300
45 features, 2 modes, 294 correct of 300
50 features, 2 modes, 293 correct of 300
55 features, 2 modes, 291 correct of 300
60 features, 2 modes, 293 correct of 300
65 features, 2 modes, 282 correct of 300
70 features, 2 modes, 284 correct of 300
75 features, 2 modes, 287 correct of 300
80 features, 2 modes, 246 correct of 300
85 features, 2 modes, 87 correct of 300
90 features, 2 modes, 243 correct of 300
95 features, 2 modes, 267 correct of 300
100 features, 2 modes, 250 correct of 300
feature count: 40
svd/lda took 7.16 seconds
Accuracy of training data
Class 1: 98 / 100, Class 2: 97 / 100, Class 3: 100 / 100
Accuracy of test set a, subjectively similar
Class 1: 17 / 20, Class 2: 19 / 20, Class 3: 20 / 20
Accuracy of test set b, subjectively a bit different
Class 1: 8 / 20, Class 2: 18 / 20, Class 3: 18 / 20
```

B.14 Output for Experiment 2: The Case of Seattle

```
loading data took 1.00 seconds
```

```
svd took 4.69 seconds
5 features, 2 modes, 254 correct of 300
10 features, 2 modes, 271 correct of 300
15 features, 2 modes, 272 correct of 300
20 features, 2 modes, 277 correct of 300
25 features, 2 modes, 285 correct of 300
30 features, 2 modes, 278 correct of 300
35 features, 2 modes, 279 correct of 300
40 features, 2 modes, 275 correct of 300
45 features, 2 modes, 283 correct of 300
50 features, 2 modes, 279 correct of 300
55 features, 2 modes, 276 correct of 300
60 features, 2 modes, 81 correct of 300
65 features, 2 modes, 100 correct of 300
70 features, 2 modes, 249 correct of 300
75 features, 2 modes, 81 correct of 300
80 features, 2 modes, 105 correct of 300
85 features, 2 modes, 207 correct of 300
90 features, 2 modes, 217 correct of 300
95 features, 2 modes, 159 correct of 300
100 features, 2 modes, 202 correct of 300
feature count: 25
svd/lda took 5.84 seconds
Accuracy of training data
Class 1: 96 / 100, Class 2: 94 / 100, Class 3: 95 / 100
Accuracy of test set a, subjectively similar
Class 1: 17 / 20, Class 2: 17 / 20, Class 3: 11 / 20
Accuracy of test set b, subjectively a bit different
Class 1: 8 / 20, Class 2: 9 / 20, Class 3: 5 / 20
```

B.15 Output for Experiment 3: Genre Classification

```
loading data took 0.97 seconds
svd took 5.85 seconds
5 features, 2 modes, 281 correct of 300
10 features, 2 modes, 294 correct of 300
15 features, 2 modes, 294 correct of 300
20 features, 2 modes, 290 correct of 300
25 features, 2 modes, 289 correct of 300
30 features, 2 modes, 292 correct of 300
35 features, 2 modes, 294 correct of 300
40 features, 2 modes, 295 correct of 300
45 features, 2 modes, 294 correct of 300
```

```
50 features, 2 modes, 293 correct of 300
55 features, 2 modes, 291 correct of 300
60 features, 2 modes, 293 correct of 300
65 features, 2 modes, 282 correct of 300
70 features, 2 modes, 284 correct of 300
75 features, 2 modes, 287 correct of 300
80 features, 2 modes, 246 correct of 300
85 features, 2 modes, 87 correct of 300
90 features, 2 modes, 243 correct of 300
95 features, 2 modes, 267 correct of 300
100 features, 2 modes, 250 correct of 300
feature count: 40
svd/lda took 6.99 seconds
Accuracy of training data
Class 1: 98 / 100, Class 2: 97 / 100, Class 3: 100 / 100
Accuracy of rag-time test data
Case 1: 5 / 20, Case 2: 0 / 20, Case 3: 0 / 20
Accuracy of early polyphony test data
Case 1: 20 / 20, Case 2: 19 / 20, Case 3: 20 / 20
Accuracy of grunge test data
Case 1: 20 / 20, Case 2: 17 / 20, Case 3: 20 / 20
Accuracy of player piano test data
Case 1: 20 / 20, Case 2: 11 / 20, Case 3: 16 / 20
Accuracy of acapella voice test data
Case 1: 17 / 20, Case 2: 19 / 20, Case 3: 2 / 20
Accuracy of rock test data
Case 1: 13 / 20, Case 2: 2 / 20, Case 3: 14 / 20
Accuracy of "additional" test data
Case 1: 0 / 20, Case 2: 19 / 20
```

C Sound Sources

C.1 Sound Clips: joplin

Composer	Work	Code
Joplin	Bethena	joplin1
Joplin	Gladiolus Rag	joplin2
Joplin	Maple Leaf Rag	joplin3
Joplin	Pineapple Rag	joplin4
Joplin	Solace	joplin5
Joplin	The Entertainer	joplin6
Joplin	Fig Leaf Rag	joplin7
Joplin	Swipesy Cakewalk	joplin8

C.2 Sound Clips: calix

Source	Work	Code
Codex Calixtinus	Benedicamus Domino	calix1
Codex Calixtinus	Congaudeant catholici	calix2
Codex Calixtinus	Dum Pater Familias	calix3
Codex Calixtinus	Graduale	calix4
Codex Calixtinus	Misit Herodes	calix5
Codex Calixtinus	Cunctipotens genitor Deus	calix6
Codex Calixtinus	Congaudeant catholici	calix7

C.3 Sound Clips: alice

Group	Work	Code
Alice in Chains	Man in the Box	alice1
Alice in Chains	No Excuses	alice2
Alice in Chains	Rooster	alice3
Alice in Chains	Them Bones	alice4
Alice in Chains	We Die Young	alice5
Alice in Chains	Angry Chairs	alice6
Alice in Chains	Grind	alice7

C.4 Sound Clips: nirvana

Group	Work	Code
Nirvana	Heart Shaped Box	nirvana1
Nirvana	In Bloom	nirvana2
Nirvana	Lithium	nirvana3
Nirvana	Rape me	nirvana4
Nirvana	Smells Like Teen Spirit	nirvana5
Nirvana	Come As You Are	nirvana6
Nirvana	Stay Away	nirvana7
Nirvana	Lake of Fire	nirvana8
Nirvana	Aneurysm	nirvana9

C.5 Sound Clips: sg

Group	Work	Code
Soundgarden	Blow Up the Outside World	sg1
Soundgarden	Burden In My Hand	sg2
Soundgarden	My Wave	sg3
Soundgarden	Rusty Case	sg4
Soundgarden	The Day I Tried to Live	sg5
Soundgarden	Black Hole Sun	sg6
Soundgarden	Fell on Black Days	sg7
Soundgarden	Outshined	sg8

C.6 Sound Clips: rag

Composer/Artist	Work	Code
Lamb	Champagne Rag	rag1
Williams	Star Wars, Cantina Band	rag2
Original Dixieland Jazz Band	Jazz me Blues	rag3

C.7 Sound Clips: poly

Composer	Work	Code
Dufay	Missa L'homme armé, Kyrie	poly1
Pérotin	Viderunt omnes	poly2
Anon	Worldes blis ne last no throwe (England ca 1265)	poly3

C.8 Sound Clips: piano

Artist	Work	Code
Pianola Player Piano	Beer Barrel Polka	piano1
Pianola Player Piano	New York New York	piano2
Pianola Player Piano	Oliver musical Medley	piano3

C.9 Sound Clips: voice

Composer/Artist	Work	Code
Bortnyansky	Cherubic Hymn	voice1
Straight No Chaser	Lion Sleeps Tonight	voice2
Chanticleer	Loch Lomond	voice3
Bruckner	Locus Iste	voice4

C.10 Sound Clips: rock

Artist	Work	Code
Van Halen	Runnin' with the Devil	rock1
The Beatles	Back in the USSR	rock2
Bruce Springsteen	Born in the USA	rock3
Viktor Tsoy	Zvezda po imeni solntse	rock4