Musical Classification Using Linear Discriminant Analysis

Maxwell Weil

Abstract

In this homework, we will be demonstrating the use of linear discriminant analysis (LDA) for classifying 1D datasets. First, a review of the concepts and strategies for applying LDA will be reviewed, in order to solidify the mathematical backings of this investigation. Next, we will show how LDA was used within this assignment using MATLAB software. Finally, we will analyze the results of this study and reflect on how the uses and limitations of LDA. Ultimately, we will seek to explain and present the practical insights that LDA can provide about data.

I. Introduction

As data becomes ever more ingrained in our society, there is a growing need for companies, researchers, and individuals to use this data to drive technological progress. Processing information and making fast, data-driven decisions have become essential. In order to accomplish this, we can look towards a variety of machine learning techniques. Machine learning allows for the automated detection of patterns within data, which can then be applied to new information to extrapolate context and significance. In this assignment, we will be covering a simple machine learning technique known as LDA and its uses in classifying 1D data.

LDA is a statistical technique which provides a space that is most suitable for discerning the differences between two or more classes within a given data set. For example, given a set of images of cats and dogs, LDA can provide a space that best separates the qualities in each image between dogs and cats. Rather than sorting each image manually, an algorithm then be applied to determine whether an image is statistically more like a dog or a cat, based on where it falls on the LDA space. This powerful method enables us to classify data in an automated and efficient way.

In this assignment, we will be using LDA to classify various samples of music. Several tests will be done to determine how well our process works under different levels of specificity. However, an in-depth coverage will first be explored to provide a theoretical basis for the strategies used.

II. Theoretical Background

Before LDA is computed, a dataset of interest is decomposed using SVD and PCA. This is done in order to project the data onto an axis that best captures variation. While we will not focus on all of the mathematics behind these methods, the projection used for PCA is shown in equation 1 below. All equations have been adapted from [1].

$$f(x,y) \approx \sum_{j=1}^{N} a_j(y)\,\phi_j(x) \qquad \text{EQ. 1}$$

In this equation, N is the number of modes, a_j is the scaling coefficients, and φ_j is an orthonormal basis. Briefly, this expansion converts a given function into an orthonormal basis of vectors which maximize the variation of the data along the bases. This adjustment is relevant for the applications of LDA.

Broadly, LDA is a method for separating multiple classes of data. LDA works by creating a lower dimensional projection of the PCA projection, where different classes within the data are separated. This requires the data to be assigned a class before computing LDA. This is known as

supervised machine learning, where the data being fed into the algorithm is given with their class labels. This trains the LDA to be able to differentiate between the given classes.

In order to differentiate between classes, LDA relies on using statistical methods to cluster each class. To do this, we can project the data onto a new vector that allows us to easily discern classes. Below, equation 2 shows how the optimal projection vector can be represented.

$$\boldsymbol{w} = \arg\max\left(\frac{\boldsymbol{w}^T S_b \boldsymbol{w}}{\boldsymbol{w}^T S_w \boldsymbol{w}}\right) \quad \text{EQ. 2}$$

In this equation, w is the projection vector, S_b is the variance between classes, and S_w is the variance within classes. Put simply, w aims to create a projection that minimizing the within class variation but maximizing the between class variation. This results in tight cluster of data for each class, that are far apart from each other. Within single class, the variation is fairly simple to calculate.

$$S_w = \sum_{j=1}^{C} \sum_x (\boldsymbol{x} - \mu_j)(\boldsymbol{x} - \mu_j)^T \quad \text{EQ. 3}$$

Equation 3 shows the calculation for S_w, where C is the number of classes, μ_j is the mean for the j^{th} class, and x is a given row of data. While this calculation is straightforward, the calculation of between class variation can be more complex. Equation 4 illustrates how to calculate S_b, given that more than 2 classes are used.

$$S_b = \sum_{j=1}^{C} (\mu_j - \mu_0)(\mu_j - \mu_0)^T \quad \text{EQ. 4}$$

In this expression, all the previous variables remain the same, but μ_0 is added to represent the mean across all of the classes. This means that the variance is not truly calculated between classes, but rather between a given class and the overall data spread. With only two classes, equation 5 can be used.

$$S_b = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \quad \text{EQ. 5}$$

This equation is truer to our original definition of Sb, where the variation between the two classes is computed. However, in this assignment, we will make use of equation 4, as three classes are used in each test. With this in mind, using PCA and LDA together can reduce the dimensionality of our data by first projecting onto N features (from PCA, equation 1) and then projecting this onto C-1 **w** vectors (from LDA, equation 2). Essentially, this allows for the best separation of classes within a dataset based on the features first extracted by PCA.

III. Algorithm Implementation and Development

In this work, we will be analyzing 1D audio data taken from a variety of musical artists and genres. We aim to create a program that can take in different audio samples, train an LDA classifier, and then accurately discriminate the proper categories for test data. The concepts previously discussed will be used to accomplish this, though an overview of the data preprocessing is necessary to fully appreciate these efforts.

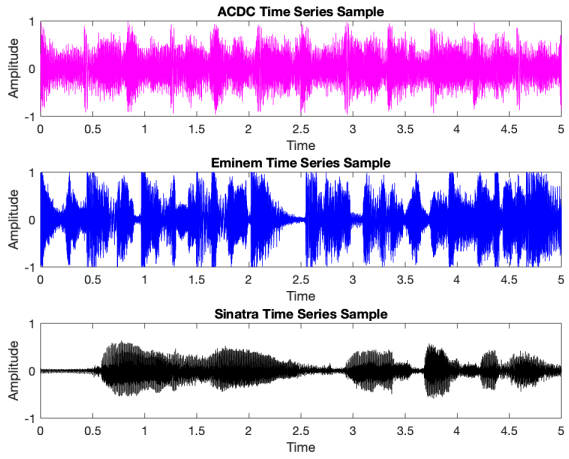

Figure 1. Time series plots of 5 second audio clips from the first test. Clear distinctions can be seen between each class, even without frequency information. However, without the frequency information, distinctions between various instruments and overtones would be challenging.

For each given test (three total tests were conducted), various songs were imported into MATLAB. First, the filenames were requested from the user (lines 7-25), and then each file was imported and downsampled to improved computation speed (lines 37-42). Next, each file was divided into random, non-overlapping 5 second segments (lines 46-50). Each of these segments was then assigned as being training data or test data, with twice as much training data as test data (lines 53-62).

Following this initial processing, each audio segment was transformed into the time-frequency domain, giving a spectrogram for each clip (lines 70-82). As in the previous assignment, the spectrogram can help show distinguishable characteristics between different types of audio signals (such as with overtones when viewing the piano and recorder). Figures 1 and 2 show 5 seconds clips from both the time and time-frequency domains for sample audio data. While both domains show marked difference between each class of data, the spectrogram provides frequency and time information that ultimately gives more useful information to our classifier.

With the spectrogram data calculated, PCA was computed on all of the training data (lines 85-90). However, it is important to note that the feature number used (the number of PCA modes) was maximized, meaning the overall dimensionality was not reduced in this step. This was done in an attempt to preserve all the possible features of the spectrogram information, allowing for a more accurate classifier.

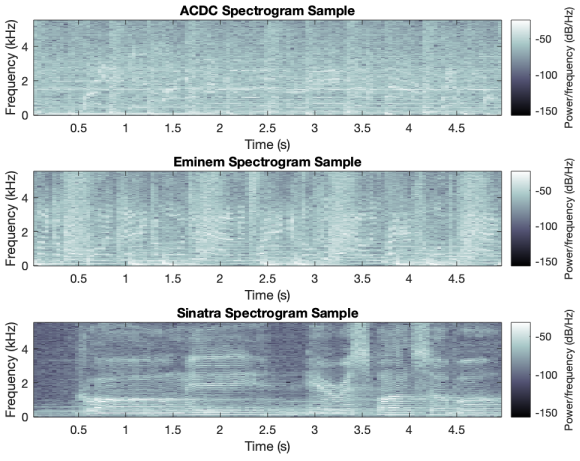

Figure 2. Time-frequency domain plots of 5 second audio clips from the first test. The same signals from Figure 1 are used here. Using information from both the time and frequency domains allows for additional distinctions between signals. Note the clear overtones in the Sinatra spectrogram, and alternating patterns of high frequencies in the ACDC and Eminem spectrograms (most likely due to higher pitch percussion). This data grants a more wholistic view of the audio signals.

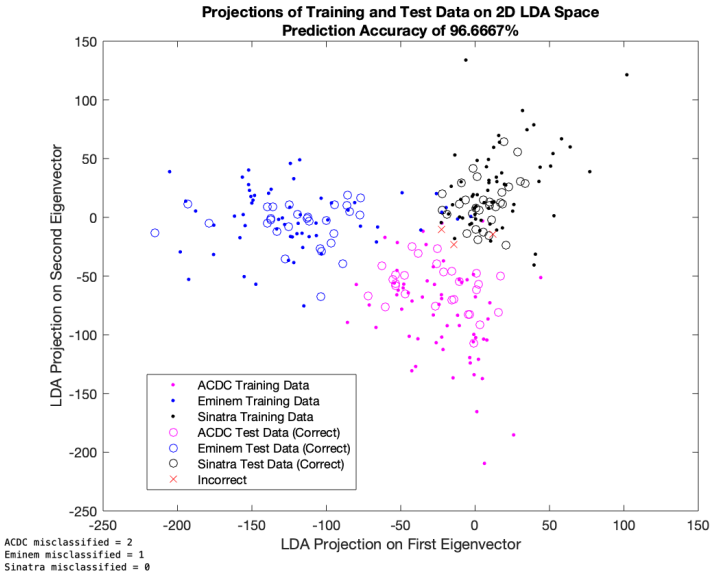

Figure 3. Results from the first classification test are shown. Each axis represents one of the two **w** *vectors obtained from LDA. Training and test data points were projected onto this space, resulting in the plot shown. Note the clear clusters of data for each artist, though some overlap is evident with Eminem and ACDC, leading to the misclassified points. Misclassified points are listed in the bottom left, listed under their true class (predicted class not shown).*

The S_b and S_w were computed, and LDA was done to find the optimal vectors to project on to (lines 98-129). Because there were 3 classes in each test, 2 vectors were found (lines 132-134). This allowed for the projection of the data onto 2D space (lines 137-139). In order to determine how to classifier each cluster of data, the average position of for each class's training data in the LDA space (class center) was calculated (lines 142-144).

The previously stored test data was then projected onto the PCA and LDA spaces (lines 150-151). Then, the Euclidean distance between the training data and each class center was found (lines 157-172). Intuitively, the closest class center was predicted to be the most likely class for a given training data point. These predictions were then compared to the true values, and the error was computed (lines 175 -182).

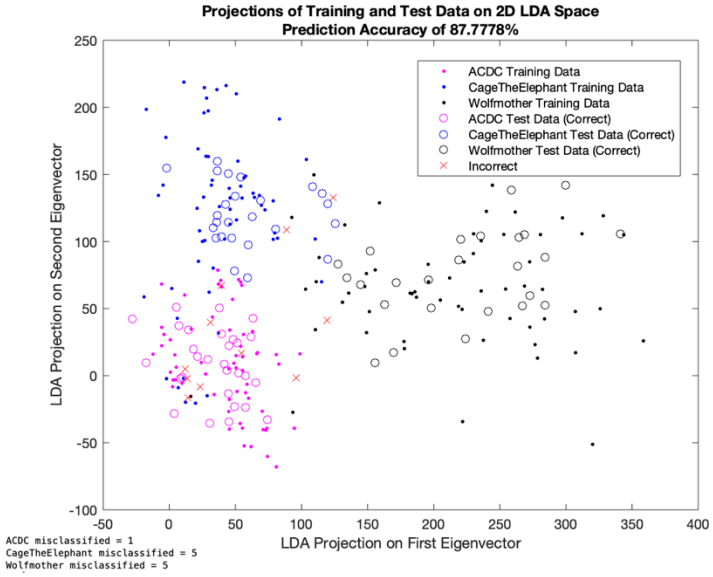

Figure 4. Results from the second test are shown. As before, each axis represents one of two LDA projections vectors. This test clearly struggled to differentiate clusters, as there are a number of overlapping data points between classes. This is most likely due to the similar qualities between the artists, as all belong to the rock genre. Misclassified points are listed in the bottom left, listed under their true class (predicted class not shown).

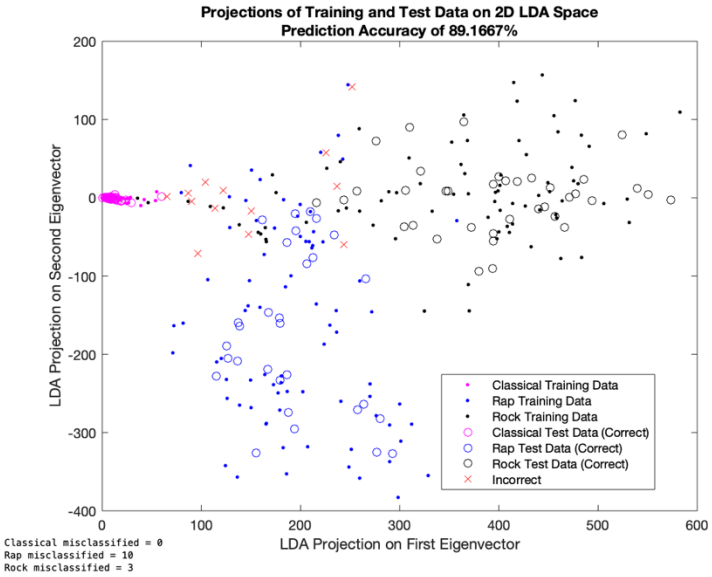

Figure 5. Results from the third test are shown. Interestingly, the classifier did very well at clustering classical music, and did not misclassify any test data from this class. However, the rock and rap genres overlap quite a bit, owing to the relatively low prediction accuracy. Misclassified points are listed in the bottom left, listed under their true class (predicted class not shown).

IV. Computational Results

Using the aforementioned methods, we were able to make interesting observation on the performance of LDA and our classifier. In the first test, audio samples from three different artists were used to train and test our classifier. ACDC, Eminem, and Frank Sinatra songs were used to give a variety of musical information, with 180 training and 90 test samples used (20 and 10 samples taken from each of three songs per artist, respectively). Figure 3 shows the overall performance of our classifier, which correctly classified 96.6% of the test data. From this figure, this is most likely due to the relatively clear separation between classes, though samples from ACDC and Eminem overlap somewhat. Generally, we can attribute the strong performance of our classifier to the distinctly different musical styles of the chosen artists.

In the second test, we presented our classifier with a more difficult task. Audio samples were again taken from 3 artists, though all fall broadly into the same genre of music. Songs from ACDC, Cage the Elephant, and Wolfmother were used, again with 180 training and 90 test samples used (20 and 10 samples taken from each of three songs per artist, respectively). Figure 4 shows the results from this test, showing how the LDA projection had a more challenging time in differentiating between the three bands. While distinct clusters are still visible, there are a number of outliers from each class that fall into the territory of another. Similar to before, this is most likely due to the similarity between the musical styles of these artists, meaning that features extracted from the spectrograms challenging to tell apart.

Finally, the third test pitted our algorithm against more broad classes. Audio samples were taken from a variety of artists, each belonging to one of three musical genres, classical, rap, or rock, with 240 training and 120 test samples used (20 and 10 samples taken from each of four songs per genre, respectively). Figure 5 illustrates that, while the classifier performed better than in the second test, it's prediction accuracy only improved by 1.39%. Interestingly, the classifier did excellent at determining whether a sample was classical, with no misclassified test points of this genre. This is probably due to a combination of the low volume and instrumental simplicity of the classical music used (piano and cello pieces). Had louder, orchestral works been tested, the classifier may not have done so well. It is also peculiar to note that most rap was misclassified than rock. This may owe to much of rap music being lyrical rather than instrumental. Because vocals are often much quieter and difficult to perceive than drumbeats or other instruments, they

may easily get lost in the spectrograms. This is evident in Figure 2, where the Frank Sinatra clip used is primarily singing, while both the ACDC and Eminem clips consist of singing and instruments.

V. Summary and Conclusions

Through this assignment, we were able to explore how LDA can be applied to classify large amounts of audio data in a fast and efficient way. Through this, we illustrated the limitations of the technique as well, and suggested possible alterations for circumventing these issues. Overall, LDA is a powerful tool with many practical applications to real life problems. Hopefully, demonstrating the underlying principles and uses of this algorithm provides insight to its many possible applications.

References

[1] J. Nathan Kutz, *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data.* Oxford University Press, 2013.

Appendix A. MATLAB Functions Used

`input(prompt)` – Displays prompt and requests user input in the command window. Can also be given 's' argument (prompt, 's') to accept input as a string.

`convertCharsToStrings(chars)` – Converts contents of chars to a string vector.

`repelem(v,n)` – Returns a vector with each element in the vector v repeated n times.

`repmat(A,n1,n2)` – Returns a matrix with matrix A being repeated n1 times in the first dimension and n2 in the second dimension.

`audioread(file)` – Reads in file as an audio file, and outputs the signal and sampling rate.

`downsample(y,n)` – Resamples signal y at every nth point (decreases sampling by a factor of n)

`randperm(n,k)`– Creates a vector of k unique integers between 1 and n.

`spectrogram(x,window(size),overlap,samples,fs)` – Computes the spectrogram of x using the specified window with a given size, the amount of overlap between each window, the number of sampling points, and the sampling rate fs.

`norm(v,n)` – Computes and returns the n-norm of a given vector v.

`sort(A, 'direction')` – Sorts the elements of a matrix A in order of size, either in ascending or descending order, as specified by the direction.

Appendix B. MATLAB Code

```
% Maxwell Weil
% AMATH 482
% HW 4

clear all; close all; clc;

% Requesting file numbers and names, assuming that song
% Files are listed as ACDC1, ACDC2, Eminem1, Eminem2, etc.
num_files = input('How many files for each class (must be equal)?   ');
class1_name = input('First class file name? ', 's');
class2_name = input('Second class file name? ', 's');
class3_name = input('Third class file name? ', 's');

% Converting names to string arrays
class1_name = convertCharsToStrings(class1_name);
class2_name = convertCharsToStrings(class2_name);
class3_name = convertCharsToStrings(class3_name);

% Setting file names to read in
class_names = repelem([class1_name,class2_name,class3_name],num_files);
file_id = repmat(1:num_files,1,num_files);
file_names = strings(size(class_names));
for i=1:length(class_names)
    file_names(i) = strcat(class_names(i), num2str(file_id(i)));
end

% Setting number of training samples and test samples taken from each file,
% As well as duration of each sample in seconds
n_train = 20;
n_test = 10;
seconds = 5;

% Looping through files
for k = 1:length(file_names)

    % Reading in current file
    current_file = strcat(file_names(k),'.mp3');
    [y,fs] = audioread(current_file);

    % Downsampling audio and sampling rate
    y = downsample(y,4);
    fs = fs/4;

    % Calculating duration of song, and total possible samples that can be
    % Taken without any overlapping data
    duration = length(y)/fs;
    num_samples = floor(duration/seconds);

    % Randomly selecting nonoverlapping start points for samples
    rand_samples = randperm(num_samples-1,n_train+n_test);

    % Creating matrix of training data
    for i = 1:n_train
        train_data(:,i,k) = y(rand_samples(i)*seconds*fs: ...
            (rand_samples(i)+1)*seconds*fs-1);
    end

    % Creating matrix of test data
    for i = 1:n_test
```

```
60            test_data(:,i,k) = y(rand_samples(i+n_train)*seconds*fs: ...
61                (rand_samples(i+n_train)+1)*seconds*fs-1);
62        end
63    end
64
65    % Reshaping data for spectrogram
66    train_data = reshape(train_data,size(train_data,1),[],1);
67    test_data = reshape(test_data,size(test_data,1),[],1);
68
69    % Looping through training data and computing spectrogram
70    for j = 1:size(train_data,2)
71        spec = spectrogram(train_data(:,j)',gausswin(1000), ...
72            560, 246, fs, 'yaxis');
73        train_spec(:,j) = reshape(abs(spec),[],1);
74    end
75
76    % Computing spectrogram for test data, parameters were chosen for optimal
77    % Performance given the type and size of data
78    for j = 1:size(test_data,2)
79        spec = spectrogram(test_data(:,j)',gausswin(1000), ...
80            560, 246, fs, 'yaxis');
81        test_spec(:,j) = reshape(abs(spec),[],1);
82    end
83
84    % Setting number of features to look at in training data
85    feature = 20;
86
87    % Computing SVD of training data, making feature space
88    [U,S,V] = svd(train_spec, 'econ');
89    music = S*V';
90    U = U(:,1:feature);
91
92    % Finding number of training data for each class (should be equal)
93    n_class1 = num_files*n_train;
94    n_class2 = num_files*n_train;
95    n_class3 = num_files*n_train;
96
97    % Finding columns of music matrix pertaining to each class
98    class1 = music(1:feature,1:n_class1);
99    class2 = music(1:feature,n_class1+1:n_class1+n_class2);
100   class3 = music(1:feature,n_class1+n_class2+1:n_class1+n_class2+n_class3);
101
102   % Finding average values of PCA projection for each class, and overall
103   avg_class1 = mean(class1,2);
104   avg_class2 = mean(class2,2);
105   avg_class3 = mean(class3,2);
106   avg_tot = mean(music(1:feature,:),2);
107
108   % Computing in class variances
109   Sw = 0;
110   for k=1:n_class1
111       Sw = Sw + (class1(:,k)-avg_class1)*(class1(:,k)-avg_class1)';
112   end
113
114   for k=1:n_class2
115       Sw = Sw + (class2(:,k)-avg_class2)*(class2(:,k)-avg_class2)';
116   end
117
118   for k=1:n_class2
119       Sw = Sw + (class3(:,k)-avg_class3)*(class3(:,k)-avg_class3)';
120   end
```

```
121
122   % Computing between class variances
123   Sb_class1 = (avg_class1-avg_tot)*(avg_class1-avg_tot)';
124   Sb_class2 = (avg_class2-avg_tot)*(avg_class2-avg_tot)';
125   Sb_class3 = (avg_class3-avg_tot)*(avg_class3-avg_tot)';
126   Sb = Sb_class3 + Sb_class2 +Sb_class1;
127
128   % Computing LDA
129   [V2,D] = eig(Sb,Sw);
130
131   % Finding the two nonzero eigenvalues and their corresponding vectors
132   [~, ind] = sort(abs(diag(D)), 'descend');
133   w = V2(:,ind(1:2));
134   w = w/norm(w,2);
135
136   % Projecting each class onto both eigenvectors
137   proj_class1 = w'*class1;
138   proj_class2 = w'*class2;
139   proj_class3 = w'*class3;
140
141   % Finding the center of each class cluster
142   centroids = [mean(proj_class1(1,:)), mean(proj_class1(2,:));
143       mean(proj_class2(1,:)), mean(proj_class2(2,:));
144       mean(proj_class3(1,:)), mean(proj_class3(2,:))];
145
146   % Establishing number of test samples for each class
147   n_test_samp = n_test*num_files;
148
149   % Projecting test data onto trained PCA and LDA found previously
150   test_proj_PCA = U'*test_spec;
151   test_proj_LDA = w'*test_proj_PCA;
152
153   % Initializing prediction vector
154   predictions = zeros(1,n_test*length(class_names));
155
156   % Looping through LDA projections for test sample
157   for i = 1:length(test_proj_LDA)
158
159       % Initializing distance vector
160       dist = zeros(1,3);
161
162       % Looping through each class's center
163       for j = 1:size(centroids,1)
164
165           % Finding distance of LDA projection from each class center
166           dist(j) = pdist2(test_proj_LDA(:,i)',centroids(j,:));
167       end
168
169       % Computing the closest class center and saving as prediction
170       [M,I] = min(dist);
171       predictions(i) = I;
172   end
173
174   % Creating vector of true classes for each test sample
175   truths = repelem([1:3],n_test_samp);
176
177   % Finding correct and incorrect predictions
178   correct = predictions==truths;
179   incorrect = ~correct;
180
181   % Computing percent correct
```

```
percent_correct = sum(correct)/length(correct)*100;
%%
% Creating plotting vectors for correct prediction in each class
% Each row is 1 for all correct predictions for that class,
% And 0 for incorrect predictions or other classes
correct_plot = logical([correct(1:n_test_samp) zeros(1,2*n_test_samp);
    zeros(1,n_test_samp) correct(n_test_samp+1:2*n_test_samp) ...
        zeros(1,n_test_samp);
    zeros(1,2*n_test_samp) correct(2*n_test_samp+1:3*n_test_samp)]);

% Plotting training and test data
% As well aslong with correct and incorrect predictions
plot(proj_class1(1,:), proj_class1(2,:), '.m')
hold on
plot(proj_class2(1,:), proj_class2(2,:), '.b')
plot(proj_class3(1,:), proj_class3(2,:), '.k')
plot(test_proj_LDA(1,correct_plot(1,:)), ...
    test_proj_LDA(2,correct_plot(1,:)), 'mo')
plot(test_proj_LDA(1,correct_plot(2,:)), ...
    test_proj_LDA(2,correct_plot(2,:)), 'bo')
plot(test_proj_LDA(1,correct_plot(3,:)), ...
    test_proj_LDA(2,correct_plot(3,:)), 'ko')
plot(test_proj_LDA(1,incorrect),test_proj_LDA(2,incorrect), 'rx')

xlabel('LDA Projection on First Eigenvector')
ylabel('LDA Projection on Second Eigenvector')
title(["Projections of Training and Test Data on 2D LDA Space", ...
    strcat("Prediction Accuracy of ", num2str(percent_correct), "%")])
legend(strcat(class1_name," Training Data"), ...
    strcat(class2_name," Training Data"), ...
    strcat(class3_name," Training Data"), ...
    strcat(class1_name," Test Data (Correct)"), ...
    strcat(class2_name," Test Data (Correct)"), ...
    strcat(class3_name," Test Data (Correct)"), ...
    "Incorrect",'Location','best')

% Computing and printing number of misclassified points from each class
class1_in = sum(~correct_plot(1,:))-2*n_test_samp;
class2_in = sum(~correct_plot(2,:))-2*n_test_samp;
class3_in = sum(~correct_plot(3,:))-2*n_test_samp;
disp(strcat(class1_name, " misclassified = ", num2str(class1_in)))
disp(strcat(class2_name, " misclassified = ", num2str(class2_in)))
disp(strcat(class3_name, " misclassified = ", num2str(class3_in)))

%% Producing sample plots and spectrograms

% Creating time vector for time series plotting
time = 1/fs:1/fs:seconds;

% Choosing last sample from each class
class1_samp = train_data(:,n_class1);
class2_samp = train_data(:,n_class1+n_class2);
class3_samp = train_data(:,n_class1+n_class2+n_class3);

% Plotting time series of sample from each class
subplot(3,1,1)
plot(time, class1_samp, 'm')
title(strcat(class1_name, " Time Series Sample"))
axis([0 5 -1 1])
xlabel('Time')
ylabel('Amplitude')
```

```
243   subplot(3,1,2)
244   plot(time, class2_samp, 'b')
245   title(strcat(class2_name, " Time Series Sample"))
246   axis([0 5 -1 1])
247   xlabel('Time')
248   ylabel('Amplitude')
249   subplot(3,1,3)
250   plot(time, class3_samp, 'k')
251   title(strcat(class3_name, " Time Series Sample"))
252   axis([0 5 -1 1])
253   xlabel('Time')
254   ylabel('Amplitude')
255
256   % Plotting spectrogram sample from each class
257   figure
258   subplot(3,1,1)
259   spectrogram(class1_samp',gausswin(1000), 560, 246, fs, 'yaxis');
260   title(strcat(class1_name, " Spectrogram Sample"))
261   subplot(3,1,2)
262   spectrogram(class2_samp',gausswin(1000), 560, 246, fs, 'yaxis');
263   title(strcat(class2_name, " Spectrogram Sample"))
264   subplot(3,1,3)
265   spectrogram(class3_samp',gausswin(1000), 560, 246, fs, 'yaxis');
266   title(strcat(class3_name, " Spectrogram Sample"))
267   colormap bone
```