Music Classification

Alyssa Mell

Abstract

Using Spectrograms, Principal Component Analysis (PCA), and Linear Discriminant Analysis (LDA) we build a classifier to differentiate between several music pieces. We compare how well our classifier works when using it with three artists from separate genres (test 1), three artists from the same genre (test 2), and multiple artists from three different genres (test 3). We take a variety of 5 second training clips for each test and then test our classifier with songs that are not used as part of the training data. Our purpose is to analyse how well our classifier holds up under these three tests.

Sec. I. Introduction and Overview

When listening to music, humans are able to hear clear distinctions between different songs, and instantly recognize artists or genres. However, this process is so natural that it can be difficult to discern exactly how or why we know when a song is from a certain artist or genre. By building a music classifier, we allow a computer to have the ability to recognize various songs and classify them for us. Since this ability is not an innate quality of a computer, we must build this classifier using certain properties of the songs.

We first use the Gábor Transform to take the spectrogram of each song clip and gather the frequencies within each piece. Then, we do PCA by taking the Singular Value Decomposition (SVD) of the transformed data and projecting each song onto its principal components. Next, we find the variance between and within each group of songs for a given artist/genre. Finally, we do LDA to determine the two significant eigenvectors of the two variances and then project our songs onto each of them. This projection is plotted in two dimensions in order to show the location of each song clip relative to the others. We are able to use these spatial orientations to determine which set of songs our test data is closest to and therefore "classify" that song as within that group.

Sec. II. Theoretical Background

A couple of important aspects of this paper are the Gábor Transform and the SVD. However, since both of these are mentioned in detail in previous papers, we will not cover their theoretical background in this paper.

Instead, in this paper, we will cover the LDA and the variances that are found and used for this part of the analysis. The first important equation to note is the equation for the variances between groups, S_b, as shown in equation 1.

$$S_b = \Sigma_{j=1}^{n}(\bar{\mu}_j - \bar{\mu})(\bar{\mu}_j - \bar{\mu})^T \tag{1}$$

In equation 1, n gives the number of groups, j specifies the current group, $\bar{\mu}_j$ gives the row-wise mean of the j^{th} group, and $\bar{\mu}$ gives the row-wise mean across all groups. In order to create the most spread between all groups and effectively classify new test songs we want to maximize this S_b value.

Another important equation is given by S_w, or the variance within groups, as shown in equation 2.

$$S_w = \Sigma_{j=1}^{n}\Sigma_{\bar{x}}(\bar{x} - \bar{\mu}_j)(\bar{x} - \bar{\mu}_j)^T \tag{2}$$

In equation 2, n gives the number of groups, j specifies the current group, $\bar{x}$ gives the corresponding generalized eigenvector of each song, and $\bar{\mu}_j$ gives the row-wise mean of the j^{th} group. Unlike S_b, we want to minimize S_w because this will cause each group to occupy a more compact area on the plot and therefore allow for clearer distinctions to be made between each group.

Equations 1 and 2 are then used in the LDA in order to find the projection onto which our groups have the largest variance between each other and the smallest variances within themselves. This is given by equation 3, defined as a generalized eigenvalue problem.

$$S_b\bar{w} = \lambda S_w\bar{w} \tag{3}$$

In equation 3, λ gives the eigenvalues and $\bar{w}$ gives the eigenvectors. We solve this equation for the eigenvector that corresponds to the largest eigenvalue. This ensures that we are maximizing the distance between classes, minimizing the distance within classes, and therefore finding the projection onto which we will have the best chance of accurately classifying songs.

Sec. III. Algorithm Implementation and Development

In this section there will often be references to the MATLAB code which is located in Appendix B. The specific line of the code that is being referenced will be stated at the end of the sentence. All references refer to the main code. Although we built three separate classifiers in our code, they were all created in the same way, so we will explain the algorithm implementation for all of these classifiers in general.

In order to create a classifier, we began by gathering a large number of 5 second clips for each artist. Each test included three groups, and 18 clips were gathered for each groups, summing to a total of 54 clips per test. After loading all of these clips into MATLAB, we began by setting a feature number to 10 (line 28). This feature number represented the number of details we wanted to gather from each song clip in order to differentiate them from each other. It was chosen through methods of trial and error in order to find the best-fitting value. Before using these clips to perform any calculations, we first resampled them, taking out only every other point, in order to reduce the amount of time needed to run the code. This resampling caused the maximum possible frequency observed to decrease by a factor of two. However, this did not affect our results because none of the clips were at high enough frequencies for this to matter. We then found the spectrogram of all song clips using the function `get_spec`(), which converted our data into frequency values over time (line 38). The return data from the spectrogram was reshaped into a column and all 54 clips were set as individual columns in the matrix `spec_songs`() (line 39).

Next, we put this transformed data into a function called `song_trainer`() which found the two significant eigenvectors corresponding to the components that maximized S_b and minimized S_w (line 43). This was found by first taking the SVD of the all song clips and then multiplying $S * V$ in order to get the projection of the songs onto their principal components. Equations 1 and 2, S_w and S_b, were then calculated for the number of specified features and the generalized eigenvalue problem (equation 3) was solved. The solution to this equation provided us with the eigenvectors in V that we projected our data onto, which correspond to the two largest eigenvalues in D.

After projecting onto these eigenvectors, we were ready to test our classifier with new clips that were not involved in the training data. 30 new clips were loaded into MATLAB for tests 1 and 2, and 27 clips were used for trial 3. All of the same steps that were followed for the training data were applied to the testing data as well. We first took the spectrogram of the test data, then multiplied it by U from the SVD, and finally found the projection of this data onto the two significant eigenvectors (lines 74-78). In order to determine which group the test data belonged to, we took the taxicab distances between the location of the center of each group and the test song on the two dimensional scale given by the eigenvector projections. The value corresponding the the first eigenvector was much larger than that given by the second eigenvector so we divided the first distance by 1000 in order to ensure the two values carried a similar weight (lines 79-81). Whichever distance ended up being the smallest was the group that the test song was classified under. All classifications were compared against the correct classification in order to give an accuracy rating for each test.

Sec. IV. Computational Results

For all figures in this section, the red, blue, and black asterisks represent the center of each corresponding group of songs. Test one involved three artists from different genres. These three artists were Billy Joel (rock), Jack Johnson (folk), and Keith Urban (country). For this test, 56.67% of the 30 test songs were correctly classified. Of these, 4 were Billy Joel's, 4 were Jack Johnson's, and 9 were Keith Urban's. Looking at figures 1-3 we see the distribution of the three artists training data as the blue, red, and black open points. All of the test songs are plotted as green asterisks. The test data for Billy Joel is in figure 1, Jack Johnson is in figure 2, and Keith Urban is in figure 3. When looking at these three figures one might guess that Jack Johnson's songs would be best classified since the red points are the most separated from the blue and black ones. However, this was not the case. As can be seen by the green asterisks in each of the figures, many of Jack Johnson's test songs ended up being closer to the blue and black asterisks than to the red one.

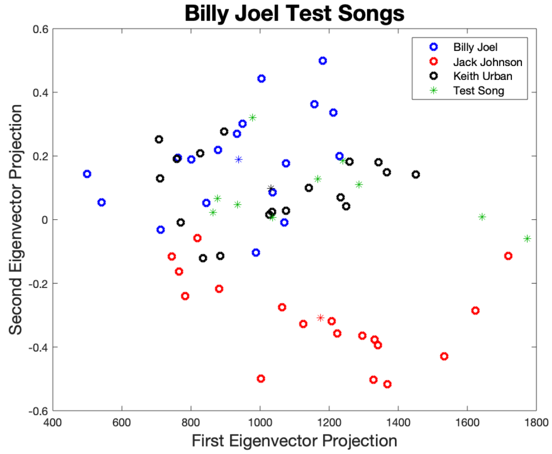

Figure 1: Test 1 - Band Classification. All test songs for Billy Joel. Groups centers are denoted by correspondingly colored asterisks.

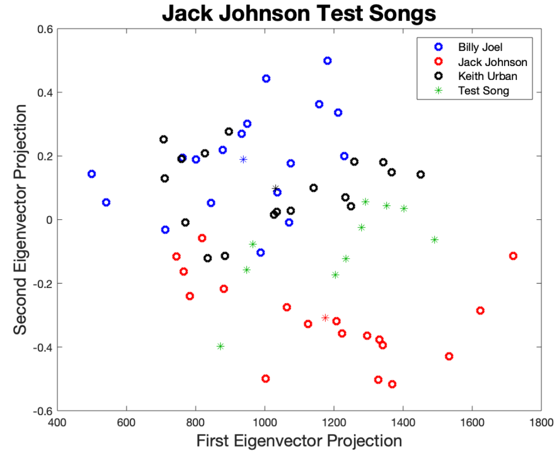

Figure 2: Test 1 - Band Classification. All test songs for Jack Johnson. Groups centers are denoted by correspondingly colored asterisks.

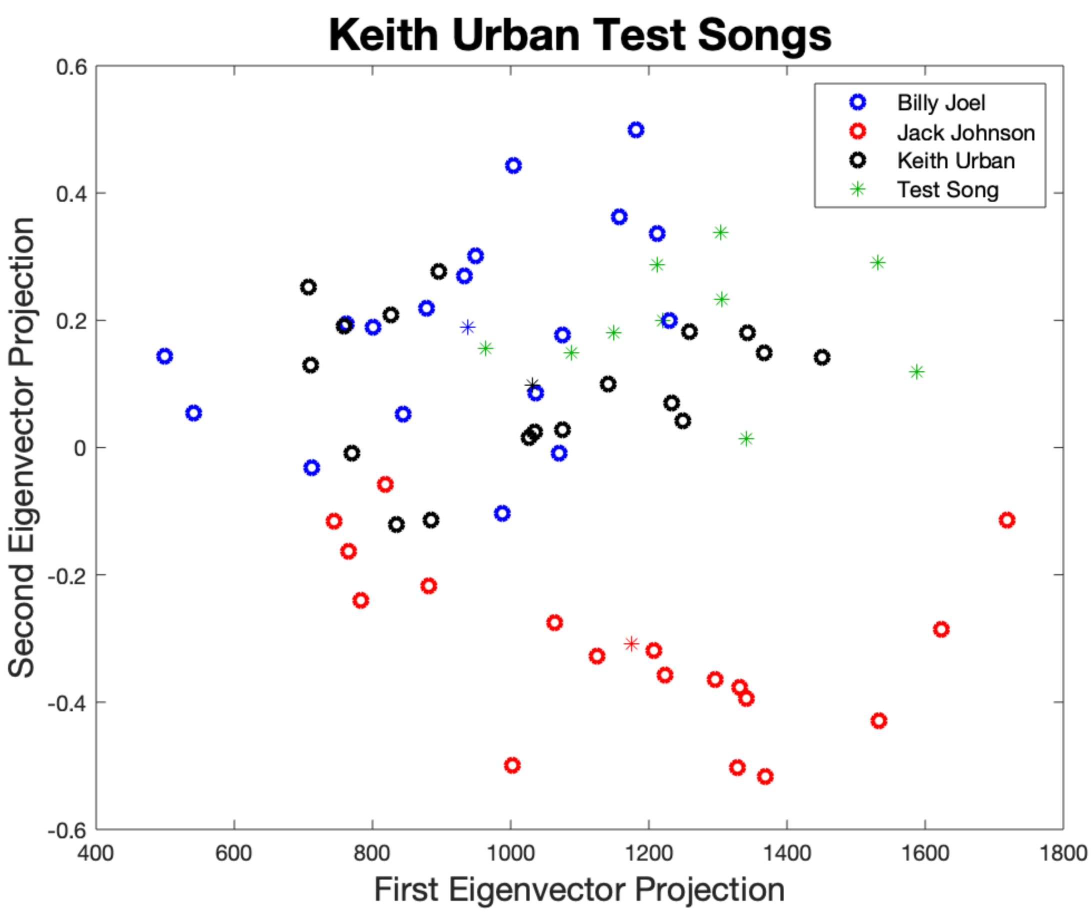

Figure 3: Test 1 - Band Classification. All test songs for Keith Urban. Groups centers are denoted by correspondingly colored asterisks.

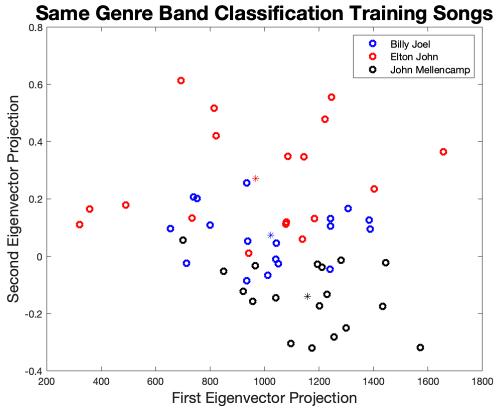

Figure 4: Test 2 - Contains all training songs for same genre band classification test. Groups centers are denoted by correspondingly colored asterisks.

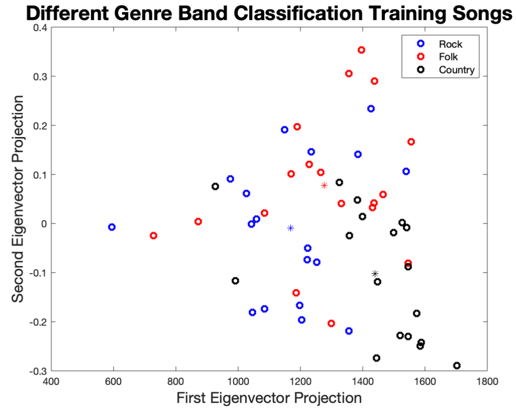

Figure 5: Test 3 - Contains all training songs for genre classification test. Groups centers are denoted by correspondingly colored asterisks.

This is most likely due to not having enough songs in the training data. The qualities of training songs used were not able to fully encapsulate all of Jack Johnson's songs, therefore causing many of the test songs to be incorrectly classified. In contrast, figure 3 shows how a majority of Keith Urban's songs are correctly classified, with only one song being mistaken as Billy Joel's.

Test two involved three artists from the same genres. These three artists were Billy Joel, Elton John, and John Mellencamp. For this test, 53.33% of the 30 test songs were correctly classified. Of these, 8 were Billy Joel's, 3 were Elton John's, and 5 were John Mellencamp's. In comparison to test one, the accuracy for this test is slightly lower. Figure 4 shows all of the songs used in the training data for this test. Again, the fact that there is so much overlap in each of these groups is likely due to an insufficient amount of training data. It is interesting to note that the best classified artist, Billy Joel, was actually in the center of all the artists. The data points associated with Billy Joel tend to have less variance within their group when compared to Elton John and John Mellencamp. This suggests that Billy Joel's songs may, in general, tend to be more similar to each other, therefore allowing them to be classified better.

Test three involved multiple artists from different genres. For this test, three different artists were chosen from the genres of rock, folk, and country. For rock, these artists were Billy Joel, Elton John, and John Mellencamp. For folk, these artists were Jack Johnson, Jason Mraz, and John Mayer. For country, these artists were Keith Urban, Blake Shelton, and Tim McGraw. For this test, 40.74% of the 27 test songs were correctly classified. Of these, 4 were rock, 2 were folk, and 5 were country. Figure 5 shows how there is a large amount of overlap between each group, so it's no surprise that this classification test does not perform was well as the previous two. In this case, it is hard to tell which group has the smallest S_w since all of groups seem to span a similar region. In order to improve this classification test, a much larger number of training songs should be used.

Sec. V. Summary and Conclusions

By using spectrograms, the SVD, and LDA we were able to extract the most important features of songs from various artists and genres. In doing so, we were then able to build a classifier that attempted to correctly classify each song into one of the three training groups. Although none of our tests had a very high success rate, the success could be improved by training the algorithm with more songs before performing the testing. From our analysis of three different groups of training and testing data, we can also conclude that it is more difficult to classify songs when they belong to the same genre or when multiple artists are used to train each genre. In these cases, it's especially important to have a large training set.

Appendix A. MATLAB Functions

Listed below are the important MATLAB functions used in this paper, along with a brief explanation of how they are implemented.

- [Y, FS] = `audioread`(FILENAME) : reads an audiofile and return the sampled data in Y and the sample rate in FS.
- [V, D] = `eig`(A) : produces a diagonal matrix D of eigenvalues and a matrix V with corresponding eigenvectors.
- [U,S,V] = `svd`(X, 'econ') : finds the "economy size" singular value decomposition of X. If x is m-by-n, where m < n, only the first m columns of V are computed and S is m-by-m.
- `diag`(X) : returns the main diagonal values of X as a vector.
- `reshape`(X,NxM,1) : converts the NxM matrix X into a column vector with NxM values.

Appendix B. MATLAB Code

Main Code

```matlab
%% Test 1 Traning - Band Classification
clc; clear all; close all

test = 1; % specify test number for plotting at bottom

% Contains all training songs for test 1
song_titles = {
    'vienna1.wav', 'vienna2.wav', 'vienna3.wav', ...
    'piano_man1.wav', 'piano_man2.wav', 'piano_man3.wav', ...
    'uptown1.wav', 'uptown2.wav', 'uptown3.wav', ...
    'my_life1.wav', 'my_life2.wav', 'my_life3.wav', ...
    'still_rock1.wav', 'still_rock2.wav', 'still_rock3.wav', ...
    'good_die1.wav', 'good_die2.wav', 'good_die3.wav', ...
    'better1.wav', 'better2.wav', 'better3.wav', ...
    'mind_sale1.wav', 'mind_sale2.wav', 'mind_sale3.wav', ...
    'upside1.wav', 'upside2.wav', 'upside3.wav', ...
    'banana1.wav', 'banana2.wav', 'banana3.wav', ...
    'sunsets1.wav', 'sunsets2.wav', 'sunsets3.wav', ...
    'remember1.wav', 'remember2.wav', 'remember3.wav', ...
    'we_were1.wav', 'we_were2.wav', 'we_were3.wav', ...
    'cop_car1.wav', 'cop_car2.wav', 'cop_car3.wav', ...
    'fighter1.wav', 'fighter2.wav', 'fighter3.wav', ...
    'blue1.wav', 'blue2.wav', 'blue3.wav', ...
    'parallel1.wav', 'parallel2.wav', 'parallel3.wav', ...
    'female1.wav', 'female2.wav', 'female3.wav'
    };

feature = 10; % set feature number
num_songs = length(song_titles); % number of training songs
spec_songs = zeros(5622750, num_songs); % initialize spectrogram data

% Get all transformed data for each song clip and set each clip as a column
% in the spec_songs matrix
for ii = 1:num_songs
    [y,Fs] = audioread(song_titles{ii});
    Fs = Fs/2;
    y = y(1:2:length(y));
    [spec] = get_spec(y, Fs);
    spec_songs(:,ii) = reshape(spec,5622750,1);
```

```
end

% Take the SVD of all transformed song clips
[U,w,w2,mid1,mid2,mid3,v,v_2] = song_trainer(spec_songs, feature);


%% Test 1 Testing - Band Classification

% Contains all testing songs for test 1
test_songs = {
    'billy_woman.wav', 'billy_longest.wav', 'billy_innocent.wav', ...
    'billy_shot.wav', 'billy_dreams.wav', 'billy_lullabye.wav', ...
    'billy_right.wav', 'billy_york.wav', 'billy_alexa.wav', ...
    'billy_extremes.wav', ...
    'jack_constellations.wav', 'jack_flake.wav', 'jack_angel.wav', ...
    'jack_tape.wav', 'jack_belle.wav', 'jack_fade.wav', ...
    'jack_staple.wav', 'jack_believe.wav', 'jack_radiate.wav', ...
    'jack_ones.wav', ...
    'keith_john.wav', 'keith_texas.wav', 'keith_thunder.wav', ...
    'keith_habit.wav', 'keith_worry.wav', 'keith_shame.wav', ...
    'keith_camaro.wav', 'keith_everything.wav', 'keith_georgia.wav', ...
    'keith_winning.wav'
    };

test_num = length(test_songs); % total tested songs
num_correct = zeros(1,test_num); % stores if song was correctly classified

% Tests each song clip to determine which category it should be placed in
% based on training data. Saves whether or not the correct value is given
% for each clip.
for ii = 1:test_num
    [y, Fs] = audioread(test_songs{ii});
    Fs = Fs/2;
    y = y(1:2:length(y));
    [spec] = get_spec(y,Fs);
    test_spec(:,1) = reshape(spec,5622750,1);
    TestMat = U'*test_spec;
    x_pos = w'*TestMat;
```

```
    y_pos = w2'*TestMat;
    dist1 = sqrt((abs(mid1(1)-x_pos)/1000) + abs(mid1(2)-y_pos));
    dist2 = sqrt((abs(mid2(1)-x_pos)/1000) + abs(mid2(2)-y_pos));
    dist3 = sqrt((abs(mid3(1)-x_pos)/1000) + abs(mid3(2)-y_pos));
    dist = [dist1 dist2 dist3];
    if find(dist == min(dist)) == 1 && ii <= 10
        num_correct(ii) = 1;
    elseif find(dist == min(dist)) == 2 && ii > 10 && ii <= 20
        num_correct(ii) = 1;
    elseif find(dist == min(dist)) == 3 && ii > 20 && ii <= 30
        num_correct(ii) = 1;
    end
end

% Gives percentage of correctly classified songs
percent_correct = sum(num_correct)/length(num_correct);

%% Test 2 Training - Same Genre
clc; clear all; close all

test = 2; % specify test number for plotting at bottom

% Contains all training songs for test 2
song_titles = {
    'vienna1.wav', 'vienna2.wav', 'vienna3.wav', ...
    'piano_man1.wav', 'piano_man2.wav', 'piano_man3.wav', ...
    'uptown1.wav', 'uptown2.wav', 'uptown3.wav', ...
    'my_life1.wav', 'my_life2.wav', 'my_life3.wav', ...
    'still_rock1.wav', 'still_rock2.wav', 'still_rock3.wav', ...
    'good_die1.wav', 'good_die2.wav', 'good_die3.wav', ...
    'tiny_dancer1.wav', 'tiny_dancer2.wav', 'tiny_dancer3.wav',
        ...
    'rocket1.wav', 'rocket2.wav', 'rocket3.wav', ...
    'bennie1.wav', 'bennie2.wav', 'bennie3.wav', ...
    'your_song1.wav', 'your_song2.wav', 'your_song3.wav', ...
    'standing1.wav', 'standing2.wav', 'standing3.wav', ...
    'daniel1.wav', 'daniel2.wav', 'daniel3.wav', ...
    'jack_diane1.wav', 'jack_diane2.wav', 'jack_diane3.wav', ...
    'small_town1.wav', 'small_town2.wav', 'small_town3.wav', ...
    'aint_done1.wav', 'aint_done2.wav', 'aint_done3.wav', ...
    'hurts_good1.wav', 'hurts_good2.wav', 'hurts_good3.wav', ...
    'cherry1.wav', 'cherry2.wav', 'cherry3.wav', ...
    'life_now1.wav', 'life_now2.wav', 'life_now3.wav'
    };
```

```
feature = 10; % set feature number
num_songs = length(song_titles); % number of training songs
spec_songs = zeros(5622750, num_songs); % initialize spectrogram data

% Get all transformed data for each song clip and set each clip as a column
% in the spec_songs matrix
for ii = 1:num_songs
    [y,Fs] = audioread(song_titles{ii});
    Fs = Fs/2;
    y = y(1:2:length(y));
    [spec] = get_spec(y, Fs);
    spec_songs(:,ii) = reshape(spec,5622750,1);
end

% Take the SVD of all transformed song clips
[U,w,w2,mid1,mid2,mid3,v,v_2] = song_trainer(spec_songs, feature);

%% Test 2 Testing - Same Genre

% Contains all testing songs for test 2
test_songs = {
    'billy_woman.wav', 'billy_longest.wav', 'billy_innocent.wav', ...
    'billy_shot.wav', 'billy_dreams.wav', 'billy_lullabye.wav', ...
    'billy_right.wav', 'billy_york.wav', 'billy_alexa.wav', ...
    'billy_extremes.wav', ...
    'elton_candle.wav', 'elton_blues.wav', 'elton_crocodile.wav', ...
    'elton_sacrifice.wav', 'elton_sorry.wav', 'elton_lucy.wav', ...
    'elton_breaking.wav', 'elton_honky.wav', 'elton_train.wav', ...
    'elton_wcn.wav', ...
    'john_crumbling.wav', 'john_not_running.wav', 'john_paper_fire.wav', ...
    'john_wild_night.wav', 'john_rumbleseat.wav', 'john_dance.wav', ...
    'john_minutes.wav', 'john_china.wav', 'john_brass.wav', ...
    'john_troubled.wav'
    };

test_num = length(test_songs); % total tested songs
```

```
num_correct = zeros(1,test_num); % stores if song was correctly
    classified

% Tests each song clip to determine which category it should be
    placed in
% based on training data. Saves whether or not the correct value
    is given
% for each clip.
for ii = 1:test_num
    [y, Fs] = audioread(test_songs{ii});
    Fs = Fs/2;
    y = y(1:2:length(y));
    [spec] = get_spec(y,Fs);
    test_spec(:,1) = reshape(spec,5622750,1);
    TestMat = U'*test_spec;
    x_pos = w'*TestMat;
    y_pos = w2'*TestMat;
    dist1 = sqrt((abs(mid1(1)-x_pos)/1000) + abs(mid1(2)-y_pos));
    dist2 = sqrt((abs(mid2(1)-x_pos)/1000) + abs(mid2(2)-y_pos));
    dist3 = sqrt((abs(mid3(1)-x_pos)/1000) + abs(mid3(2)-y_pos));
    dist = [dist1 dist2 dist3];
    if find(dist == min(dist)) == 1 && ii <= 10
        num_correct(ii) = 1;
    elseif find(dist == min(dist)) == 2 && ii > 10 && ii <= 20
        num_correct(ii) = 1;
    elseif find(dist == min(dist)) == 3 && ii > 20 && ii <= 30
        num_correct(ii) = 1;
    end
end

% Gives percentage of correctly classified songs
percent_correct = sum(num_correct)/length(num_correct);

%% Test 3 Training - Genre Classification
clc; clear all; close all

test = 3; % specify test number for plotting at bottom

% Contains all training songs for test 3
song_titles = {
    'vienna2.wav', 'piano_man2.wav', 'uptown2.wav', ...
    'my_life2.wav', 'still_rock2.wav', 'good_die2.wav', ...
    'tiny_dancer2.wav', 'rocket2.wav', 'bennie2.wav'...
    'your_song2.wav', 'standing2.wav', 'daniel2.wav', ...
    'jack_diane2.wav', 'small_town2.wav', 'aint_done2.wav', ...
```

```
    'hurts_good2.wav', 'cherry2.wav', 'life_now2.wav', ...
    'better2.wav', 'mind_sale2.wav', 'upside2.wav', ...
    'banana2.wav', 'sunsets2.wav', 'remember2.wav', ...
    'new_light.wav', 'wonderland.wav', 'burning_room.wav', ...
    'gravity.wav', 'carry_me.wav', 'daughters.wav', ...
    'give_up.wav', 'have_it.wav', 'yours.wav', ...
    'unlonely.wav', 'million_miles.wav', 'lucky.wav', ...
    'we_were2.wav', 'cop_car2.wav', 'fighter2.wav', ...
    'blue2.wav', 'parallel2.wav', 'female2.wav', ...
    'god_country.wav', 'sangria.wav', 'guy_girl.wav', ...
    'hell_right.wav', 'name_dogs.wav', 'honey_bee.wav', ...
    'neon_church.wav', 'thought.wav', 'humble.wav', ...
    'like_dying.wav', 'something_like_that.wav', 'smile.wav'
    };

feature = 10; % set feature number
num_songs = length(song_titles); % number of training songs
spec_songs = zeros(5622750, num_songs); % initialize spectrogram data

% Get all transformed data for each song clip and set each clip as a column
% in the spec_songs matrix
for ii = 1:num_songs
    [y,Fs] = audioread(song_titles{ii});
    Fs = Fs/2;
    y = y(1:2:length(y));
    [spec] = get_spec(y, Fs);
    spec_songs(:,ii) = reshape(spec,5622750,1);
end

% Take the SVD of all transformed song clips
[U,w,w2,mid1,mid2,mid3,v,v_2] = song_trainer(spec_songs, feature);

%% Test 3 Testing - Genre Classification

% Contains all testing songs for test 3
test_songs = {
    'billy_woman.wav', 'billy_longest.wav', 'billy_innocent.wav', ...
    'elton_candle.wav', 'elton_blues.wav', 'elton_crocodile.wav', ...
    'john_crumbling.wav', 'john_not_running.wav', 'john_paper_fire.wav', ...
    'jack_constellations.wav', 'jack_flake.wav', 'jack_angel.wav',
```

```
    ...
    'john_who_says.wav', 'john_georgia.wav', 'john_wildfire.wav', ...
    'jason_night.wav', 'jason_dance.wav', 'jason_remedy.wav', ...
    'keith_john.wav', 'keith_texas.wav', 'keith_thunder.wav', ...
    'blake_cool.wav', 'blake_beach.wav', 'blake_fool.wav', ...
    'like_love.wav', 'tim_speak.wav', 'tim_southern.wav'
    };

test_num = length(test_songs); % total tested songs
num_correct = zeros(1,test_num); % stores if song was correctly classified

% Tests each song clip to determine which category it should be placed in
% based on training data. Saves whether or not the correct value is given
% for each clip.
for ii = 1:test_num
    [y, Fs] = audioread(test_songs{ii});
    Fs = Fs/2;
    y = y(1:2:length(y));
    [spec] = get_spec(y,Fs);
    test_spec(:,1) = reshape(spec,5622750,1);
    TestMat = U'*test_spec;
    x_pos = w'*TestMat;
    y_pos = w2'*TestMat;
    dist1 = sqrt((abs(mid1(1)-x_pos)/1000) + abs(mid1(2)-y_pos));
    dist2 = sqrt((abs(mid2(1)-x_pos)/1000) + abs(mid2(2)-y_pos));
    dist3 = sqrt((abs(mid3(1)-x_pos)/1000) + abs(mid3(2)-y_pos));
    dist = [dist1 dist2 dist3];
    if find(dist == min(dist)) == 1 && ii <= 9
        num_correct(ii) = 1;
    elseif find(dist == min(dist)) == 2 && ii > 9 && ii <= 18
        num_correct(ii) = 1;
    elseif find(dist == min(dist)) == 3 && ii > 18 && ii <= 27
        num_correct(ii) = 1;
    end
end

% Gives percentage of correctly classified songs
percent_correct = sum(num_correct)/length(num_correct);

%% Plot Single Song
close all; % close figure
```

```
% Get data for test song
[y, Fs] = audioread('john_crumbling.wav'); % specify song to test
Fs = Fs/2;
y = y(1:2:length(y));
[spec] = get_spec(y,Fs);
test_spec(:,1) = reshape(spec,5622750,1);

% Calculate position of test song
TestMat = U'*test_spec;
x_pos = w'*TestMat;
y_pos = w2'*TestMat;

% Calculate distances between test song and center of each traning
    set
dist1 = sqrt((abs(mid1(1)-x_pos)/1000) + abs(mid1(2)-y_pos));
dist2 = sqrt((abs(mid2(1)-x_pos)/1000) + abs(mid2(2)-y_pos));
dist3 = sqrt((abs(mid3(1)-x_pos)/1000) + abs(mid3(2)-y_pos));
dist = [dist1 dist2 dist3];

% Blue: Test 1 - Billy Joel, Test 2 - Billy Joel, Test 3 - Rock
p1 = plot(v(1,:),v_2(1,:),'ob','Linewidth',2);
hold on

% Red: Test 1 - Jack Johnson, Test 2 - Elton John, Test 3 - Folk
p2 = plot(v(2,:),v_2(2,:),'or','Linewidth',2);

% Black: Test 1 - Keith Urban, Test 2 - John Mellencamp, Test 3 -
    Country
p3 = plot(v(3,:),v_2(3,:), 'ok','Linewidth',2);

p4 = plot(mid1(1),mid1(2),'*b'); % plots center of blue points
p5 = plot(mid2(1),mid2(2),'*r'); % plots center of red points
p6 = plot(mid3(1),mid3(2),'*k'); % plots center of black points
p7 = plot(x_pos,y_pos,'*c'); % plots test song in cyan

xlabel('First Eigenvector Projection');
ylabel('Second Eigenvector Projection');

% Creates legend and title depending on test number
if test == 1
    legend([p1 p2 p3 p7],'Billy Joel', 'Jack Johnson', 'Keith
        Urban', 'Test Song');
    title('Projection of Song Clips onto Eigenvectors Test 1');
elseif test == 2
```

```
    legend([p1 p2 p3 p7],'Billy Joel', 'Elton John', 'John
       Mellencamp', 'Test Song');
    title('Projection of Song Clips onto Eigenvectors Test 2');
else
    legend([p1 p2 p3 p7],'Rock','Folk','Country','Test Song');
    title('Projection of Song Clips onto Eigenvectors Test 3');
end
```

Spectrogram Code

```
% This function takes in values for the sampled data and sampling
    rate for
% a given song clip. It returns the transformed data of this clip
    by
% finding the frequencies associated with the spectrogram.
function [spec] = get_spec(y, Fs)
v = y.';
L = 5;
n = Fs*L;
t2 = linspace(0,L,n+1); t = t2(1:n);
a = 1;
tslide=0:0.1:L;
spec = zeros(length(tslide),n);

for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2);
    vg=g.*v;
    vgt=fft(vg);
    spec(j,:) = fftshift(abs(vgt));
end

end
```

Song Trainer Code

```
% This function takes in values for the transformed data of all
    training
% song clips and the number of features. It returns the U matrix
    from the
% SVD of this data along with its two largest eigenvectors (w, w2)
    , the
% location of each song clip in two dimensional space (mid1, mid2,
    mid3),
% and the projection of each song onto the two largest
    eigenvectors (v,
% v_2).
function [U,w,w2,mid1,mid2,mid3,v,v_2] = song_trainer(spec_songs,
    feature)

    % Separate out songs
    song1_0 = spec_songs(:,1:18);
    song2_0 = spec_songs(:,19:36);
    song3_0 = spec_songs(:,37:54);

    % length of Songs
    n1 = size(song1_0,2); n2 = size(song2_0,2); n3 = size(song3_0
        ,2);

    [U,S,V] = svd([song1_0 song2_0 song3_0],'econ'); % SVD
    songs = S*V'; % projection onto principal components

    % Extracts data corresponding to specified feature number
    U = U(:,1:feature);
    song1 = songs(1:feature,1:n1);
    song2 = songs(1:feature,n1+1:2*n1);
    song3 = songs(1:feature,2*n1+1:3*n1);

    % Finds mean value for each song
    m = mean(songs(1:feature),2);
    m1 = mean(song1,2);
    m2 = mean(song2,2);
    m3 = mean(song3,2);

    % Calculates within class variance
    Sw = 0;
    for k=1:n1
        Sw = Sw + (song1(:,k)-m1)*(song1(:,k)-m1)';
    end
```

```
    for k=1:n2
        Sw = Sw + (song2(:,k)-m2)*(song2(:,k)-m2)';
    end
    for k=1:n3
        Sw = Sw + (song3(:,k)-m3)*(song3(:,k)-m3)';
    end

    % Calculates between class variance
    Sb = (m1-m)*(m1-m)' + (m2-m)*(m2-m)' + (m3-m)*(m3-m)';

    % Linear discriminant analysis
    [V2,D] = eig(Sb,Sw);

    % Finds two largest eigenvectors and projects data onto each of them
    [~,ind] = max(abs(diag(D)));
    w = V2(:,ind); w = w/norm(w,2);
    D(ind,ind) = 0; % remove largest singular value
    [~,ind] = max(abs(diag(D)));
    w2 = V2(:,ind); w = w/norm(w,2);

    % Projects songs onto two largest eigenvectors
    v1 = w'*song1;
    v2 = w'*song2;
    v3 = w'*song3;
    v = [v1; v2; v3];
    v1_2 = w2'*song1;
    v2_2 = w2'*song2;
    v3_2 = w2'*song3;
    v_2 = [v1_2; v2_2; v3_2];

    % Finds approximate two dimensional position of each song
    mid1 = [mean(v1), mean(v1_2)];
    mid2 = [mean(v2),mean(v2_2)];
    mid3 = [mean(v3),mean(v3_2)];
end
```