Music Classification using Principal Component Analysis and Linear Discriminant Analysis

Seong Hyun Han

March 6, 2020

Abstract

In this assignment, spectrogram generation, principal component analysis, and linear discriminant analysis is used to develop a machine learning algorithm that can classify songs based on their band group or genre. Training and testing data sets will be made in order to develop this program. Afterwards, the data will be transformed into spectrogram data containing frequency information of the band/genre. Singular value decomposition is used to project the data onto a number of principal components. These projected data will then be classified into classes by using linear discriminant analysis. The accuracy of the classifier will be tested and compared.

Sec. I Introduction and Overview

The main goal for this assignment is to develop a basic machine learning algorithm that can correctly classify new 5 second music clips to their corresponding band or genre. This algorithm will be developed by using principal component analysis (PCA) and linear discriminant analysis (LDA). As such, PCA and LDA will be explored in this process. In order to carry out this task, 5 second music recordings (converted to .wav files) of varying bands and genres will be sampled using a software called Audacity. From the music recordings, a training data set of 90 known songs (30 songs per band/genre) and a test data set of 30 songs (10 songs per band/genre) will be made for each test cases – this is also called supervised learning. The test cases will be: 1. Band classification of three different bands from three different genres, 2. Band classification of three different bands from the same genre, and 3. Genre classification of three songs from three different genres. Furthermore, the accuracies for each of the test cases will be compared and analyzed. Before the songs are used for analysis, spectrograms of each of the songs will be made in order to extract out their respective frequency signatures that can correlate to a specific band or genre when compiled. After compiling the music data matrices, PCA will be carried out using singular value decomposition (SVD). The data will then be projected onto the main principal components (i.e., 'features'), creating a lower dimensional space containing relevant information for each individual band or genre. Afterwards, LDA will be used to find a linear combination of features that is able to differentiate multiple classes.

Sec. II. Theoretical Background

In order to approach this assignment, several key mathematical concepts must be understood. The first concept is spectrogram generation. As previously discussed in the assignment "*Audio Signal Time-Frequency Analysis Using Gábor Transforms*," a spectrogram can be made with a sliding window (e.g., Gaussian window) that takes a time domain signal at specific times points (τ) along the entirety of the signal and takes the Fourier transform of each window. This allows for the extraction of frequency data at each time frame.

Another main concept that must be understood is singular value decomposition (SVD). Although previously discussed in the assignment "*Principal Component Analysis of Video Recordings*," information pertaining to this concept is reproduced due to its importance for this assignment. The cardinal concept of SVD is that it factorizes a given matrix into key constitutive components which all have a specific meaning in applications. The SVD is essentially a transformation that scales and rotates a set of vectors. The scaling and rotation of this transformation can be controlled through the proper construction of a matrix A [1]. The full SVD of this matrix A can be factorized into the compact matrix notation of

$$\mathrm{A} = \mathrm{U}\Sigma\mathrm{V}^* \qquad \text{(Eqn. 1)}$$

$$\mathrm{U} \in \mathbb{C}^{\mathrm{m}\times\mathrm{m}} \text{ is unitary} \qquad \text{(Eqn. 2)}$$

$$\Sigma \in \mathbb{R}^{m\times n} \text{ is diagonal} \qquad \text{(Eqn. 3)}$$
$$V \in \mathbb{C}^{n\times n} \text{ is unitary.} \qquad \text{(Eqn. 4)}$$

The full SVD is generated by adding additional “silent” columns to the U and Σ matrices to make them unitary and allow for compatibility with rank deficient matrices. The matrix U has orthonormal columns containing the left singular vectors of A and the matrix V^* contains the right singular vectors of A. The matrix Σ has r (rank) positive diagonal singular values (σ_j) ordered with the greatest value first and then in descending order by convention (the remaining n - r columns are all zeros) [1].

The SVD can be computed by

$$\begin{aligned} A^T A &= (U\Sigma V^*)^T (U\Sigma V^*) \\ &= V\Sigma U^* U\Sigma V^* \\ &= V\Sigma^2 V^* \end{aligned} \qquad \text{(Eqn. 5)}$$

$$\begin{aligned} AA^T &= (U\Sigma V^*)(U\Sigma V^*)^T \\ &= U\Sigma V^* V\Sigma U^* \\ &= U\Sigma^2 U^* \end{aligned} \qquad \text{(Eqn. 6)}$$

$$A^T AV = V\Sigma^2 \qquad \text{(Eqn. 7)}$$
$$AA^T U = U\Sigma^2. \qquad \text{(Eqn. 8)}$$

By computing the normalized eigenvectors for Eqn. 7 and Eqn. 8, the orthonormal basis vectors for U and V are produced. Moreover, the singular values can be produced by taking the square root of the eigenvalues of these equations. Furthermore, the SVD makes it possible for every matrix to be diagonalized if the proper bases for the domain and range are used. Additionally, this decomposition method allows for the projection of the data onto lower dimensional representations in an algorithmic manner. One key application of SVD is to carry out PCA, where the ideal or simplified behavior of a real data set that is noisy, potentially redundant, and unoptimized can be extracted. This data set can be constructed as a matrix with $X \in \mathbb{R}^{m\times n}$ where m is the number of measurements and n is the number of collected data points. In order to remove redundancy and identify maximal variance in the data set, the covariance matrix can be considered, where covariance shows the dependencies between two variables. The covariance matrix $C_X = \frac{1}{n-1} XX^T$, where 1/(n - 1) is the normalization constant of an unbiased estimator and C_X is a $m \times m$ matrix with the diagonal representing the variance (dynamic of interest) and the off-diagonal terms representing the covariance (redundancy) of the system. Therefore, we want to order the variance from greatest to least and covariances equal to zero. Thus, we want to diagonalize the covariance matrix $C_X = V\Lambda V^{-1}$ where the basis of eigenvectors in V are called principal components and the eigenvalues in Λ gives the variance. This is exactly carried out by SVD, where the meaning of $U\Sigma V^*$ are conserved – U: principal components and Σ: eigenvalues of the scaled data matrix (Eqn. 9 and Eqn. 10) [1].

$$A = \frac{1}{\sqrt{n-1}} X \qquad \text{(Eqn. 9)}$$

$$\begin{aligned} AA^T &= \left(\frac{1}{\sqrt{n-1}} X\right)\left(\frac{1}{\sqrt{n-1}} X\right)^T = C_x \\ &= U\Sigma^2 U^* \end{aligned} \qquad \text{(Eqn. 10)}$$

The last main concept that must be realized is a statistical decision-based classification method called linear discriminant analysis (LDA). LDA is a widely used method in machine learning algorithms to find linear combination of features that can differentiate multiple classes. The goal of LDA is to find a suitable projection basis (w) (Eqn. 11) that maximizes the distance between the inter-class data while minimizing the intra-class data.

$$\mathrm{w} = \arg\max_{w} \frac{w^T S_B w}{w^T S_W w} \quad \text{(Eqn. 11)}$$

where S_B is the inter-class variation and S_W is the intra-class variation given by the equations

$$\mathrm{S_B} = \frac{1}{\mathrm{C}} \sum_{i=1}^{C} (\mu_i - \mu)(\mu_\mathrm{i} - \mu)^\mathrm{T} \quad \text{(Eqn. 12)}$$

$$\mathrm{S}_W = \sum_{i=1}^{C} \sum_{x} (\mathrm{x} - \mu_\mathrm{i})(\mathrm{x} - \mu_\mathrm{i})^\mathrm{T} \quad \text{(Eqn. 13)}$$

where C is the number of classes, μ_i is the class mean, μ is the mean of the class means, and x is the data projections on to the desired features [2]. The LDA projection basis is calculated by finding the solution of the generalized eigenvalue problem (Eqn. 14),

$$\mathrm{S_B w} = \lambda \mathrm{S_W w} \quad \text{(Eqn. 14)}$$

where the maximum eigenvalue λ and its correlated eigenvector gives the LDA projection basis [1].

Sec. III. Algorithm Implementation and Development

In order to make the training and testing song data sets, a software called Audacity was used in order to generate 5 second music clips with a sampling frequency of 44100 Hz. Music was played within the computer's sound card input and sampled back from its output as a dual channel (stereo) recording. A macro key was made and used that selects 5 seconds of the recorded song/project, trims the rest of the music, and exports the clip as a .wav file. For Test 1, the bands chosen were Flume (Electronica), Hall & Oats (Pop Rock), and Nat King Cole (Jazz). For Test 2, the selected genre was K-pop and the bands were BIGBANG, BTS, and Girl's Generation. For Test 3, the three chosen genres were Classical, Rap, and Punk Rock.

Starting with **Test 1** in **Appendix B. MATLAB codes**, the music data matrices for each band/genre/testing set were made through the developed function called *song_compiler()*. This function down samples the music clip, to quicken computational time, using the function *downsample()* by a factor defined by the variable *ds* (every fourth point was used for this assignment). It then row-wise concatenates each song of a band or genre and outputs it as a matrix (30 songs x 5*Fs/4).

In order to produce the spectrograms of each song in the data set, another function called *spectro()* was developed. Before running this function, the computational time domain was defined as *L = 5 seconds*. The number of Fourier modes *n* equal the total number of samples (5*Fs/4). The time domain is defined from 0 to L with *n* + 1 intervals. This time interval is then converted into a frequency interval with n Fourier modes. This frequency interval must be scaled by a factor of 1/*L*, in order to have frequency in terms of hertz and not the usual 2π periodic angular frequency (ω). The frequency components are adjusted to account for n being odd. Afterwards, they are shifted using *fftshift()* to have the value zero at the center and stored in the variable *ks*. The *spectro()* function makes a spectrogram of all of the 30 songs using a Gabor/Gaussian window with width parameter a = 50 and $\tau = 0.1$, going through the aforementioned process described in the previous section. The output of *spectro()* is a spectrogram data matrix with size 30 songs x 51 time frames * n.

After all spectrogram data for the three songs and test data were created, the three song data were inputted into a developed function called *trainer()* along with a feature number, which is the number of principal components used for projecting the data onto – 40 out of the 90 principal components was selected given that around half of the singular values showed a relatively high value and showed sufficient accuracy when the code was later optimized. The *trainer()* function carries out 1. SVD decomposition, 2. LDA and Threshold generation. After column-wise concatenating the three songs, SVD decomposition was performed using the function *svd()* with the concatenated data matrix. For LDA, each song was projected onto the first 40 principal components ($S * V^T$). Afterwards, each projected song data was used to calculate $S_W, S_B,$ and w by using Eqn. 12, 13, and 14. In order to generate thresholds for each class (bands of different genres), the principal component projected values were projected onto the LDA projection basis w. Then, the mean of each projections were calculated and the projections were sorted both in acceding order and by the mean value using the *mean()* and *sort()* functions, respectively. One threshold was defined between the projection with the smallest mean value and the projection with the medium mean value. The point in which the highest value of the lowest mean projection becomes lower than the lowest value of the medium mean projection was calculated. The mean of the projection values at theses points were taken as the threshold value. This same method was used to calculate the threshold between the medium mean and high mean projections.

After finding the thresholds and song projections onto the LDA projection basis, the accuracy of the test data was calculated using this classification system. The test song data that was turned into a spectrogram format was projected onto the first 40 principal components outputted by the *trainer()* function. These projections were then projected on the LDA projection basis w, and the song positions on w were stored in the vector *pval*. The first ten, middle ten, and last ten values in *pval* correlate to the bands Flume, Hall & Oats, and Nat King Cole, respectively. Looking at the means of each of the LDA projected songs of the training data, it was determined that means from lowest to highest was Flume, Hall & Oats, and Nat King Cole, in that order. Therefore, the accuracy was calculated by counting the number of the first ten songs of *pval* correlated to the band Flume (i.e., below threshold 1) and divided by the total number of songs (i.e., 10). The number of the middle ten songs of *pval* correlating to the band Hall & Oats (i.e., between threshold 1 and 2) was counted and divided by the total number of songs. The number of the last ten songs of *pval* correlating to the band Nat King Cole (i.e., greater than threshold 2) was counted and divided by the total number of songs. The total accuracy of the test data set classification was given by the mean of the individual accuracies. This same procedure was carried out for Test 2 and Test 3.

Sec. IV. Computational Results

Test 1) From Figure 1, (left) it is evident that the singular value energies show a sigmoidal pattern. Therefore, a little less than half of the principal values/feature was estimated to capture most of the energy of the song data. We also found that this led to higher accuracies as you would assume by increasing the feature number but don't want to overfit the data. The middle subfigure shows that there is

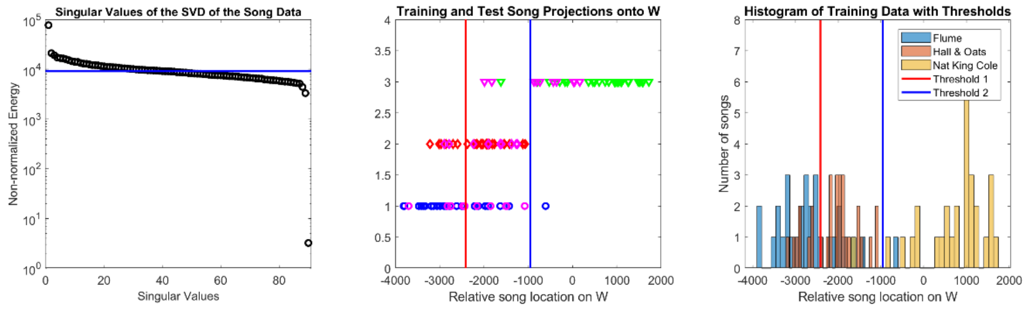

Figure 1. (*left*) Plot of the singular values (90) of the song data and their respective non-normalized energies on semilog axes – blue horizontal line at y = value of the 40th singular value. (*middle*) Plot of training and test song projections onto the LDA projection basis – (y=1: Flume, y=2: Hall & Oats, y=3: Nat King Cole) – RGB colors = training data songs, magenta = test data songs. (*right*) Histogram of the statistics of the training data projected onto the LDA projection basis with thresholds.

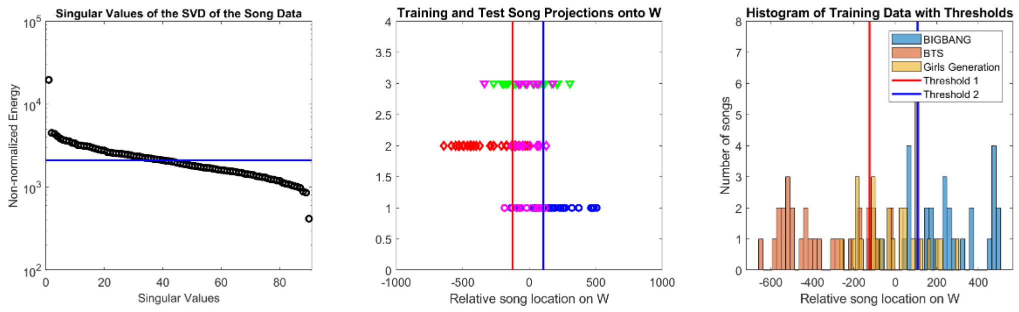

Figure 2. (*left*) Plot of the singular values (90) of the song data and their respective non-normalized energies on semilog axes – blue horizontal line at y = value of the 40th singular value. (*middle*) Plot of training and test song projections onto the LDA projection basis – (y=1: BIGBANG, y=2: BTS, y=3: Girl's Generation) – RGB colors = training data songs, magenta = test data songs. (*right*) Histogram of the statistics of the training data projected onto the LDA projection basis with thresholds.

a moderate amount of intra-class variance for each band in the training data set and a distinct separation (inter-class variance) between the bands Hall & Oats and Nat King Cole. However, there was only little separation between the bands Flume and Hall & Oats. The points in magenta show that that test data songs roughly fall inside their respective classes. This similar information regarding the training data is also shown by the histogram in the right subfigure. All in all, the accuracy (ratios = number of correct classification/total number of songs) for each band and for the total test data set was as follows: **Flume: 0.5 | Hall & Oats: 0.8 | Nat King Cole: 0.8 | Total: 0.7.**

Test 2) From Figure 2, the left subfigure shows similar information as Test 1. The middle subfigure shows that there is a relatively little amount of intra-class variance for each band in the training data set and low separation (inter-class variance) between all three bands. The points in magenta show that that test data songs roughly fall inside their respective classes. This similar information regarding the training data is also shown by the histogram in the right subfigure. All in all, the accuracy (ratios = number of correct classification/total number of songs) for each band and for the total test data set was as follows: **BIGBANG: 0.1 | BTS: 0.1 | Girl's Generation: 0.8 | Total: 0.33.**

Test 3) From Figure 3, the left subfigure shows similar information as Test 2. The middle subfigure shows that there is a moderate amount of intra-class variance for each band in the training data set and moderate separation (inter-class variance) between all three bands. The points in magenta show that that only the classical songs from the test data roughly fall inside its respective genre class. This similar information regarding the training data is also shown by the histogram in the right subfigure. All in all, the song accuracy (ratios = number of correct classification/total number of songs) for each genre and for the total test data set was as follows: **Punk: 0.1 | Classical: 1.0 | Rap: 0.0 | Total: 0.37.**

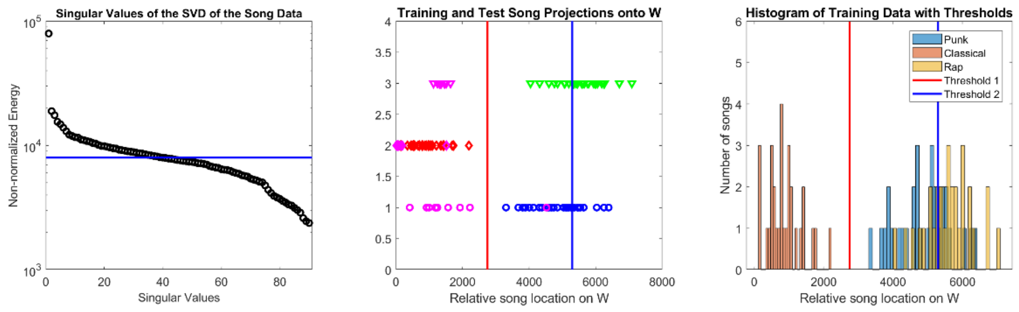

Figure 3. (*left*) Plot of the singular values (90) of the song data and their respective non-normalized energies on semilog axes – blue horizontal line at y = value of the 40th singular value. (*middle*) Plot of training and test song projections onto the LDA projection basis – (y=1: Punk, y=2: Classical, y=3: Rap) – RGB colors = training data songs, magenta = test data songs. (*right*) Histogram of the statistics of the training data projected onto the LDA projection basis with thresholds.

Sec. V. Summary and Conclusions

All in all, spectrogram generation, principal component analysis, and a multi-class linear discriminant analysis carried out using singular value decomposition was used to classify songs based on their band group or genre. Looking at the computational results, the total classifier accuracy of Test 1 - three different bands from three different genres – was the highest (0.70). This is reasonable since selecting one band will likely lead to small intra-class variation and bands from different genres should likely result in larger inter-class variations. Test 2 – three different bands from one genre – led to the lowest classifier accuracy of 0.33. Although there is small intra-class variation due to the songs being from a specific band, there is little inter-class variation as all the bands were from the same genre (K-pop); Girl's Generation may have had a higher accuracy as it was girl group with very different song style versus the two boy groups. Test 3 – songs from three different genres – had the middle accuracy of 0.37. This may be due to having a moderate inter- and intra- class variations. However, in this case, the test data songs did not fall into their respective classes except for classical music. This may be due to audio sampling error from creating the music clips on different audio streaming platforms with different volume/Audacity settings. Through this assignment, a relatively simple machine learning algorithm could be made to classify various songs to their correlated band/genre. Moreover, with more training data made with the same sampling settings, better test data, and a better classifying mechanism such as 2D K-means clustering, the algorithm could be improved to achieve higher accuracy.

Appendix A. MATLAB functions used and brief implementation explanation

- **svd():** take the singular value decomposition of a data matrix.
- **downsample():** downsample input signal.
- **sort():** sort in ascending or descending order.

Appendix B. MATLAB codes

```
%% AMATH 482 Homework 4
% Seong Hyun Han
% 3/06/20

%% Test 1
clear all; close all; clc
% bg_name = band/genre name ('flume'), num_song = number of songs (30),
% ds = downsample number (4),
num_song = 30;
ds = 4;
[flume_data] = song_compiler('flume', num_song, ds);
[hall_data] = song_compiler('hall', num_song, ds);
[nat_data] = song_compiler('nat', num_song, ds);
[test1_data] = song_compiler('test1', num_song, ds);

%% Create Gabor transform spectrogram
music_time = 5; % 5 seconds for each audio clip
fs = 44100;
fs_d = 44100/ds;
L = music_time;
tslide=0:0.1:L; % tau = 0.1
n = fs_d * music_time; % Number of Fourier modes (2^n)
t2=linspace(0,L,n+1); % Define the time domain discretization
t=t2(1:n); % Only consider the first n points for periodicity
% Fourier components rescaled to have frequency in hertz
k=(1/L)*[0:(n-1)/2 (-n+1)/2:-1]; % account for n being odd
ks=fftshift(k); % Fourier components with zero at the center

%% Spectrogram data matrix for flume, hall, nat
% a = 50, tau = 0.1, L = music duration, n = number of datapoints
% file_num = number of songs per data set, t = discrete time domain
[flume_sg] = spectro(flume_data, 50, 0.1, L, n, num_song, t);
[hall_sg] = spectro(hall_data, 50, 0.1, L, n, num_song, t);
[nat_sg] = spectro(nat_data, 50, 0.1, L, n, num_song, t);
[test1_sg] = spectro(test1_data, 50, 0.1, L, n, num_song, t);

%% SVD + LDA
[U,S,V,w,threshold_1,threshold_2,sort_bg_1,sort_bg_2,sort_bg_3,~,~,~] =
trainer(flume_sg,hall_sg,nat_sg,40);

%% Classify and Plots
% Plot singular values
figure(1)
subplot(1,3,1)
semilogy(diag(S),'ko','Linewidth',2)
hold on
diag_s = diag(S);
plot([0 length(diag(S))+1],[diag_s(40), diag_s(40)],'b','LineWidth',2)
set(gca,'Xlim',[0,length(diag(S))+1],'Fontsize',12)
title('Singular Values of the SVD of the Song Data','Fontsize',12)
xlabel('Singular Values','Fontsize',12)
ylabel('Non-normalized Energy','Fontsize',12)

% Plot song projections onto w
% figure(2)
% plot(sort_bg_1,ones(length(sort_bg_1)),'ob','Linewidth',2)
% hold on
% plot(sort_bg_2,2.*ones(length(sort_bg_2)),'dr','Linewidth',2)
% plot(sort_bg_3,3.*ones(length(sort_bg_3)),'vg','Linewidth',2)
% ylim([0 4])
% title('Song Projections onto W','Fontsize',12)
% xlabel('Relative song location on W','Fontsize',12)

% Plot histogram of training data with thresholds
subplot(1,3,3)
histogram(sort_bg_1,length(sort_bg_1))
hold on
histogram(sort_bg_2,length(sort_bg_2))
```

```matlab
histogram(sort_bg_3,length(sort_bg_3))
plot([threshold_1 threshold_1],[0 8],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 8],'b','LineWidth',2)
title('Histogram of Training Data with Thresholds','Fontsize',12)
legend('Flume', 'Hall & Oats','Nat King Cole','Threshold 1','Threshold 2')
xlabel('Relative song location on W')
ylabel('Number of songs')
set(gca,'Fontsize',12)

% Classify
TestNum = size(test1_sg,2); % 30
TestMat = U'*test1_sg;  % PCA projection
pval = w'*TestMat;  % LDA projection

% Plot test songs onto song data projections onto w
subplot(1,3,2)
plot(sort_bg_1,ones(length(sort_bg_1)),'ob','Linewidth',2)
hold on
plot(pval(1:10),ones(length(pval(1:10))),'om','Linewidth',2)
plot(sort_bg_2,2.*ones(length(sort_bg_2)),'dr','Linewidth',2)
plot(pval(11:20),2.*ones(length(pval(11:20))),'dm','Linewidth',2)
plot(sort_bg_3, 3.*ones(length(sort_bg_2)),'vg','Linewidth',2)
plot(pval(21:30),3.*ones(length(pval(21:30))),'vm','Linewidth',2)
plot([threshold_1 threshold_1],[0 8],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 8],'b','LineWidth',2)
ylim([0 4])
title('Training and Test Song Projections onto W')
xlabel('Relative song location on W')
set(gca,'Fontsize',12)

% Find accuracy
flume_test = pval(1:10);
hall_test = pval(11:20);
nat_test = pval(21:30);

% Accuracy of Nat King Cole
true_high = 0;
for i = 1:10
    if nat_test(i)>threshold_2
        true_high = true_high + 1;
    end
end
accuracy_nat = true_high/length(nat_test);

% Accuracy of Hall & Oats
true_mid = 0;
for i = 1:10
    if hall_test(i)>threshold_1 && hall_test(i)<threshold_2
        true_mid = true_mid + 1;
    end
end
accuracy_hall = true_mid/length(hall_test);

% Accuracy of Flume
true_low = 0;
for i = 1:10
    if flume_test(i)<threshold_1
        true_low = true_low + 1;
    end
end
accuracy_flume = true_low/length(flume_test);

accuracy_total = (accuracy_nat + accuracy_hall + accuracy_flume)/3;
%% Test 2
clear all; close all; clc
num_song = 30;
ds = 4;
[bb_data] = song_compiler('bb', num_song, ds);
[bts_data] = song_compiler('bts', num_song, ds);
[gg_data] = song_compiler('gg', num_song, ds);
[test2_data] = song_compiler('test2', num_song, ds);
```

```
%% Create Gabor transform spectrogram
music_time = 5; % 5 seconds for each audio clip
fs = 44100;
fs_d = 44100/ds;
L = music_time;
tslide=0:0.1:L; % tau = 0.1
n = fs_d * music_time; % Number of Fourier modes (2^n)
t2=linspace(0,L,n+1); % Define the time domain discretization
t=t2(1:n); % Only consider the first n points for periodicity
% Fourier components rescaled to have frequency in hertz
k=(1/L)*[0:(n-1)/2 (-n+1)/2:-1]; % account for n being odd
ks=fftshift(k); % Fourier components with zero at the center

%% Spectrogram data matrix for flume, hall, nat
% a = 50, tau = 0.1, L = music duration, n = number of datapoints
% file_num = number of songs per data set, t = discrete time domain
[bb_sg] = spectro(bb_data, 50, 0.1, L, n, num_song, t);
[bts_sg] = spectro(bts_data, 50, 0.1, L, n, num_song, t);
[gg_sg] = spectro(gg_data, 50, 0.1, L, n, num_song, t);
[test2_sg] = spectro(test2_data, 50, 0.1, L, n, num_song, t);

%% SVD + LDA
[U,S,V,w,threshold_1,threshold_2,sort_bg_1,sort_bg_2,sort_bg_3,~,~,~] =
trainer(bb_sg,bts_sg,gg_sg,40);

%% Classify and Plots
% Plot singular values
figure(1)
subplot(1,3,1)
semilogy(diag(S),'ko','Linewidth',2)
hold on
diag_s = diag(S);
plot([0 length(diag(S))+1],[diag_s(40), diag_s(40)],'b','LineWidth',2)
set(gca,'Xlim',[0,length(diag(S))+1],'Fontsize',12)
title('Singular Values of the SVD of the Song Data','Fontsize',12)
xlabel('Singular Values','Fontsize',12)
ylabel('Non-normalized Energy','Fontsize',12)

% Plot song projections onto w
% figure(2)
% plot(sort_bg_1,ones(length(sort_bg_1)),'ob','Linewidth',2)
% hold on
% plot(sort_bg_2,2.*ones(length(sort_bg_2)),'dr','Linewidth',2)
% plot(sort_bg_3,3.*ones(length(sort_bg_3)),'vg','Linewidth',2)
% ylim([0 4])
% title('Song Projections onto W','Fontsize',12)
% xlabel('Relative song location on W','Fontsize',12)

% Plot histogram of training data with thresholds
subplot(1,3,3)
histogram(sort_bg_1,length(sort_bg_1))
hold on
histogram(sort_bg_2,length(sort_bg_2))
histogram(sort_bg_3,length(sort_bg_3))
plot([threshold_1 threshold_1],[0 8],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 8],'b','LineWidth',2)
title('Histogram of Training Data with Thresholds','Fontsize',12)
legend('BIGBANG', 'BTS','Girls Generation','Threshold 1','Threshold 2')
xlabel('Relative song location on W')
ylabel('Number of songs')
set(gca,'Fontsize',12)

% Classify
TestNum = size(test2_sg,2); % 30
TestMat = U'*test2_sg;  % PCA projection
pval = w'*TestMat;  % LDA projection

% Plot test songs onto song data projections onto w
subplot(1,3,2)
plot(sort_bg_1,ones(length(sort_bg_1)),'ob','Linewidth',2)
```

```
hold on
plot(pval(1:10),ones(length(pval(1:10))),'om','Linewidth',2)
plot(sort_bg_2,2.*ones(length(sort_bg_2)),'dr','Linewidth',2)
plot(pval(11:20),2.*ones(length(pval(11:20))),'dm','Linewidth',2)
plot(sort_bg_3, 3.*ones(length(sort_bg_2)),'vg','Linewidth',2)
plot(pval(21:30),3.*ones(length(pval(21:30))),'vm','Linewidth',2)
plot([threshold_1 threshold_1],[0 8],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 8],'b','LineWidth',2)
ylim([0 4])
title('Training and Test Song Projections onto W')
xlabel('Relative song location on W')
set(gca,'Fontsize',12)

% Find accuracy
bb_test = pval(1:10);
bts_test = pval(11:20);
gg_test = pval(21:30);

% Accuracy of BIGBANG
true_high = 0;
for i = 1:10
    if bb_test(i)>threshold_2
        true_high = true_high + 1;
    end
end
accuracy_bb = true_high/length(bb_test);

% Accuracy of Girls Generation
true_mid = 0;
for i = 1:10
    if gg_test(i)>threshold_1 && gg_test(i)<threshold_2
        true_mid = true_mid + 1;
    end
end
accuracy_gg = true_mid/length(gg_test);

% Accuracy of BTS
true_low = 0;
for i = 1:10
    if bts_test(i)<threshold_1
        true_low = true_low + 1;
    end
end
accuracy_bts = true_low/length(bts_test);

accuracy_total = (accuracy_bb + accuracy_gg + accuracy_bts)/3;

%% Test 3
clear all; close all; clc
num_song = 30;
ds = 4;
[classical_data] = song_compiler('classical', num_song, ds);
[punk_data] = song_compiler('punk', num_song, ds);
[rap_data] = song_compiler('rap', num_song, ds);
[test3_data] = song_compiler('test3', num_song, ds);

%% Create Gabor transform spectrogram
music_time = 5; % 5 seconds for each audio clip
fs = 44100;
fs_d = 44100/ds;
L = music_time;
tslide=0:0.1:L; % tau = 0.1
n = fs_d * music_time; % Number of Fourier modes (2^n)
t2=linspace(0,L,n+1); % Define the time domain discretization
t=t2(1:n); % Only consider the first n points for periodicity
% Fourier components rescaled to have frequency in hertz
k=(1/L)*[0:(n-1)/2 (-n+1)/2:-1]; % account for n being odd
ks=fftshift(k); % Fourier components with zero at the center

%% Spectrogram data matrix for flume, hall, nat
% a = 50, tau = 0.1, L = music duration, n = number of datapoints
```

```matlab
% file_num = number of songs per data set, t = discrete time domain
[classical_sg] = spectro(classical_data, 50, 0.1, L, n, num_song, t);
[punk_sg] = spectro(punk_data, 50, 0.1, L, n, num_song, t);
[rap_sg] = spectro(rap_data, 50, 0.1, L, n, num_song, t);
[test3_sg] = spectro(test3_data, 50, 0.1, L, n, num_song, t);

%% SVD + LDA
[U,S,V,w,threshold_1,threshold_2,sort_bg_1,sort_bg_2,sort_bg_3,~,~,~] =
trainer(punk_sg,classical_sg,rap_sg,40);

%% Plots

% Plot singular values
figure(1)
semilogy(diag(S),'ko','Linewidth',2)
set(gca,'Xlim',[0,length(diag(S))+1],'Fontsize',12)
title('Singular Values of the SVD of the Song Data')
xlabel('Singular Values')
ylabel('Non-normalized Energy')

% Plot song projections onto w
figure(2)
plot(sort_bg_1,ones(length(sort_bg_1)),'ob','Linewidth',2)
hold on
plot(sort_bg_2,2.*ones(length(sort_bg_2)),'dr','Linewidth',2)
plot(sort_bg_3,3.*ones(length(sort_bg_3)),'*g','Linewidth',2)
ylim([0 4])
title('Song Projections onto W')
xlabel('Relative song location on W')

% Plot histogram of training data with thresholds
figure(3)
histogram(sort_bg_1,length(sort_bg_1))
hold on
histogram(sort_bg_2,length(sort_bg_2))
histogram(sort_bg_3,length(sort_bg_3))
plot([threshold_1 threshold_1],[0 8],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 8],'b','LineWidth',2)
title('Classical, Punk, Rap')
set(gca,'Fontsize',12)
legend('Punk','Classical','Rap','Threshold 1','Threshold 2')
title('Histogram of Training Data with Thresholds')
ylabel('Number of songs')
xlabel('Relative song location on W')

%% Classify and Plots
% Plot singular values
figure(1)
subplot(1,3,1)
semilogy(diag(S),'ko','Linewidth',2)
hold on
diag_s = diag(S);
plot([0 length(diag(S))+1],[diag_s(40), diag_s(40)],'b','LineWidth',2)
set(gca,'Xlim',[0,length(diag(S))+1],'Fontsize',12)
title('Singular Values of the SVD of the Song Data','Fontsize',12)
xlabel('Singular Values','Fontsize',12)
ylabel('Non-normalized Energy','Fontsize',12)

% Plot song projections onto w
% figure(2)
% plot(sort_bg_1,ones(length(sort_bg_1)),'ob','Linewidth',2)
% hold on
% plot(sort_bg_2,2.*ones(length(sort_bg_2)),'dr','Linewidth',2)
% plot(sort_bg_3,3.*ones(length(sort_bg_3)),'vg','Linewidth',2)
% ylim([0 4])
% title('Song Projections onto W','Fontsize',12)
% xlabel('Relative song location on W','Fontsize',12)

% Plot histogram of training data with thresholds
subplot(1,3,3)
histogram(sort_bg_1,length(sort_bg_1))
```

```
hold on
histogram(sort_bg_2,length(sort_bg_2))
histogram(sort_bg_3,length(sort_bg_3))
plot([threshold_1 threshold_1],[0 6],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 6],'b','LineWidth',2)
title('Histogram of Training Data with Thresholds','Fontsize',12)
legend('Punk','Classical','Rap','Threshold 1','Threshold 2')
xlabel('Relative song location on W')
ylabel('Number of songs')
set(gca,'Fontsize',12)

% Classify
TestNum = size(test3_sg,2); % 30
TestMat = U'*test3_sg;  % PCA projection
pval = w'*TestMat;  % LDA projection

% Plot test songs onto song data projections onto w
subplot(1,3,2)
plot(sort_bg_1,ones(length(sort_bg_1)),'ob','Linewidth',2)
hold on
plot(pval(1:10),ones(length(pval(1:10))),'om','Linewidth',2)
plot(sort_bg_2,2.*ones(length(sort_bg_2)),'dr','Linewidth',2)
plot(pval(11:20),2.*ones(length(pval(11:20))),'dm','Linewidth',2)
plot(sort_bg_3, 3.*ones(length(sort_bg_2)),'vg','Linewidth',2)
plot(pval(21:30),3.*ones(length(pval(21:30))),'vm','Linewidth',2)
plot([threshold_1 threshold_1],[0 8],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 8],'b','LineWidth',2)
ylim([0 4])
title('Training and Test Song Projections onto W')
xlabel('Relative song location on W')
set(gca,'Fontsize',12)

% Find accuracy
punk_test = pval(1:10);
classical_test = pval(11:20);
rap_test = pval(21:30);

% Accuracy of Rap
true_high = 0;
for i = 1:10
    if rap_test(i)>threshold_2
        true_high = true_high + 1;
    end
end
accuracy_rap = true_high/length(rap_test);

% Accuracy of Punk
true_mid = 0;
for i = 1:10
    if punk_test(i)>threshold_1 && punk_test(i)<threshold_2
        true_mid = true_mid + 1;
    end
end
accuracy_punk = true_mid/length(punk_test);

% Accuracy of Classical
true_low = 0;
for i = 1:10
    if classical_test(i)<threshold_1
        true_low = true_low + 1;
    end
end
accuracy_classical = true_low/length(classical_test);

accuracy_total = (accuracy_rap + accuracy_punk + accuracy_classical)/3;

%% Functions
% Audio data matrix
% bg_name = band/genre name (flume), num_song = number of songs (30),
% ds = downsample number (4)
function [bg_data] = song_compiler(bg_name, num_song, ds) % bg_name
```

```
cell = {};
for n = 1:num_song
    file_name = '%s_%d.wav';
    file_name = sprintf(file_name, bg_name, n);
    [y, ~]= audioread(file_name);
    y = downsample(y, ds);
    cell{n} = y(:, 1); % only use the channel one inputs
end
bg_data = [cell{:}]'; % each row is a song
end

% Spectrogram
% a = 50, tau = 0.1, L = music duration, n = number of datapoints
function [sg] = spectro(data, width, tau, L, n, num_song, t)
    a = width; % width
    tslide=0:tau:L;
    m_gt_g = zeros(length(tslide),floor(n)); % store filtered frequency data
    sg = []; % store spectrogram of all 30 songs a row vectors
    for song = 1:num_song
        for j=1:length(tslide)
            % Gabor filter function / window
            gabor = exp(-a*(t-tslide(j)).^2); % tau = tslide(j) = translation parameter
            m_gt = gabor.*data(song,:); % apply filter to signal (mutiplication in time domain)
            m_gt_f = fft(m_gt);
            m_gt_g(j,:) = fftshift(abs(m_gt_f)); % We don't want to scale it
        end

        row_vec = [];
        for i = 1:length(tslide)
            row_vec = [row_vec m_gt_g(i, :)];
        end
        sg =[sg; row_vec];
    end
    sg = sg';
end

% Trainer
% bg_sg_n = band/genre spectrogram data, feature = # of principal components

function
[U,S,V,w,threshold_1,threshold_2,sort_bg_1,sort_bg_2,sort_bg_3,sorted_high,sorted_mid,sorted_low]
= trainer(bg_sg_1,bg_sg_2,bg_sg_3,feature)
    n1 = size(bg_sg_1,2); % number of columns in each spectrogram data set
    n2 = size(bg_sg_2,2);
    n3 = size(bg_sg_3,2);

    % SVD Decomposition
    % data matrix = n x (3 * num_song)
    [U,S,V] = svd([bg_sg_1, bg_sg_2, bg_sg_3],'econ');
    % U = 2811375x90
    % S = 90x90
    % V = 90x90

    % LDA
    bg_proj = S*V'; % projection onto principal components
    U = U(:,1:feature);
    % U = 2811375xfeature
    bg_proj_1 = bg_proj(1:feature,1:n1);
    bg_proj_2 = bg_proj(1:feature,n1+1:n1+n2);
    bg_proj_3 = bg_proj(1:feature,n1+n2+1:n1+n2+n3);

    m1 = mean(bg_proj_1,2); % (10x1) mean of all columns for each row
    m2 = mean(bg_proj_2,2);
    m3 = mean(bg_proj_3,2);

    Sw = 0; % within class variances
    for k=1:n1 %(30)
        Sw = Sw + (bg_proj_1(:,k)-m1)*(bg_proj_1(:,k)-m1)'; % sigma * sigma = variance
    end
    for k=1:n2
        Sw = Sw + (bg_proj_2(:,k)-m2)*(bg_proj_2(:,k)-m2)';
    end
    for k=1:n3
        Sw = Sw + (bg_proj_3(:,k)-m3)*(bg_proj_3(:,k)-m3)';
    end

    num_class = 3;
%     m_total = mean([bg_proj_1, bg_proj_2, bg_proj_3]);
    m_total = (m1+m2+m3)/3;
    Sb1 = (m1-m_total)*(m1-m_total)'; % between class
    Sb2 = (m2-m_total)*(m2-m_total)'; % between class
    Sb3 = (m3-m_total)*(m3-m_total)'; % between class
    Sb = (Sb1+Sb2+Sb3)/num_class;

    [V2,D] = eig(Sb,Sw); % linear discriminant analysis
    [~,ind] = max(abs(diag(D)));
    w = V2(:,ind); % maximum eigenvalue and its associated eigenvector
    w = w/norm(w,2);

    v_proj_1 = w'*bg_proj_1;
    v_proj_2 = w'*bg_proj_2;
    v_proj_3 = w'*bg_proj_3;

    sort_bg_1 = sort(v_proj_1);
    sort_bg_2 = sort(v_proj_2);
    sort_bg_3 = sort(v_proj_3);

    sort_mean_1 = mean(sort_bg_1);
    sort_mean_2 = mean(sort_bg_2);
    sort_mean_3 = mean(sort_bg_3);

    [~, sort_mean_ind] = sort([sort_mean_1, sort_mean_2, sort_mean_3]);
    sorted_high_ind = sort_mean_ind(3);
    sorted_mid_ind = sort_mean_ind(2);
    sorted_low_ind = sort_mean_ind(1);

    sort_bg = [sort_bg_1; sort_bg_2; sort_bg_3];
    sorted_high = sort_bg(sorted_high_ind,:);
    sorted_mid = sort_bg(sorted_mid_ind,:);
    sorted_low = sort_bg(sorted_low_ind,:);

    t1 = length(sorted_low);
    t2 = 1;
    while sorted_low(t1)>sorted_mid(t2)
        t1 = t1-1;
        t2 = t2+1;
    end
    threshold_1 = (sorted_low(t1)+sorted_mid(t2))/2;

    t2 = length(sorted_mid);
    t3 = 1;
    while sorted_mid(t2)>sorted_high(t3)
        t2 = t2-1;
        t3 = t3+1;
    end
    threshold_2 = (sorted_mid(t2)+sorted_high(t3))/2;
end
```

References

1. Kutz, Jose Nathan. *Data-Driven Modeling &amp; Scientific Computation: Methods for Complex Systems &amp; Big Data*. Oxford University Press, 2013.
2. “Linear Discriminant Analysis.” *Wikipedia*, Wikimedia Foundation, 23 Feb. 2020, en.wikipedia.org/wiki/Linear_discriminant_analysis.