Music Genre Recognition using Short-time Fourier Transform and Principal Component Analysis

Jiaqi Zhang
University of Washington
zjq95@uw.edu

Abstract

Music genres such as rock, pop, classic etc., are instantly recognizable to human ears. This implies that our brain is capable of processing the audio signals and characterize them into genres that we are familiar of. The objective of this report is to produce an algorithm that can classify a given piece of music into different genres.

I. Introduction

This paper will first introduce the theoretical background of the algorithm using *Short Time Fourier Transform*, *Principal Component Analysis* and *Discrimination Analysis*. Followed by the implementation and development of the algorithm. The algorithm will be applied to three different tests. In the first test we will build training sets and test sets. We will use the algorithm to classify the song clips from three different bands of various genres. In the second test, we will use the same technique to see if it is able to distinguish between songs within the same genre. Finally in the third test, we will use the algorithm for general classification of the music genres such as Rock, Pop and Classics.

II. Theoretical Background

This algorithm implements three main features: the *Short-Time Fourier Transform*(STFT), *Principal Component Analysis*(PCA) and *Discriminant Analysis* (DA). STFT is used to capture songs' features. First of all, the signal to be transformed is multiplied by a window function, which is non-zero only for a short period of time. Then Fourier Transform is applied to the resulting signal as the window slid along the time axis[1]. Mathematically, it is written as:

$$\mathbf{STFT}\{x(t)\}(\tau,\omega) \equiv X(\tau,\omega) = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-j\omega t}\,dt \tag{1}$$

w(t) is the window function, which is chosen to be the Gaussian Window Function centered at 0. *x(t)* is the signal to be transformed. $x(\tau,\ \omega)$ is the fourier transform of $x(t)w(t-\tau)$. In Figure 1, we can see that as τ changes over time, ω sweeps through the signal and captures features from separate points of time. STFT investigates both time and frequency information of a given signal.

In order to identify a song's genre, we extract the key features of the transformed signal using the PCA/SVD technique. SVD decomposes a matrix into three matrices:

$$X = U\Sigma V^* \tag{2}$$

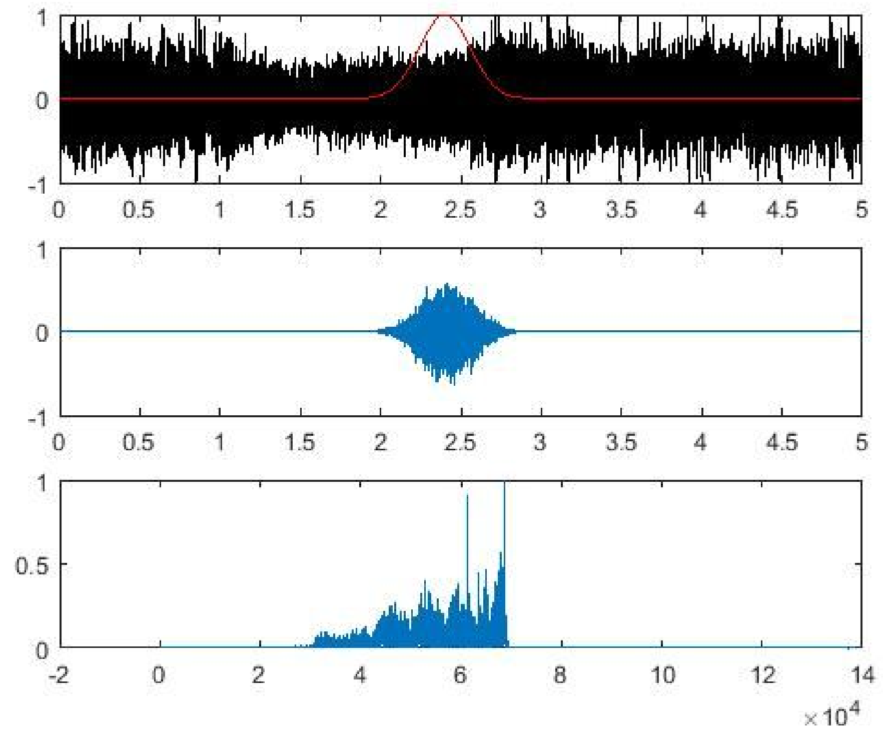

Figure 1. (a) The original signal and the overlap of the windowed function (red line). (b) The filtered signal at approximately 2.4s. (c)the FFT of the filtered signal.

U and *V* are the left and right singular vevtors of ***X***. The diagonal of Σ are the corresponding singular values. We perform PCA to get the principal modes of the matrix ***X***. First of all, we transform ***X*** into the principal component basis:

$$Y = U * X \tag{3}$$

In this new basis, columns of *U* are the principal components of ***X***. We can use these principal components of features to recognize each song. For a specific music genre, we uses multiple songs of the same genre and repeat the STFT and PCA to construct a representation matrix.

Then we use *Discriminant Analysis* on this representation matrix to predict membership in a cluster or category based on observed values of several variables[2]. Given a new observation, DA classifies the new observed data into one of the already observed groups.

III. Algorithm Implementation and Development

1. Obtain Data

We start by constructing training sets. After loading a song to Matlab, we obtain a *n*2* matrix and the frequency which is typically 44100 frames per song. *n* is the total number of frames in the song and 2 means two channels. In order to decrease the amount of data, we evaluate the average of the two channels. In this paper, we selected 5 songs for each genre and took two 5s samples from each song. This would give us 10 samples for each genre. The 5s section can be taken from any part of song. For this paper, it is arbitrarily chosen to be from 60s to 65s and from 65s to 70s. We put the row vector representing each sample together to form the signal matrix ***a***. Each row of ***a*** represents a sample and the columns represent the frames during the 5s period.

2. Define Parameters

Now that we have the signal matrix ***a***, we need to define some parameters for STFT. First of all, we define a dummy matrices ***aSpec***. We fill ***aSpec*** with the frequency information during the STFT loop. Then we define the sampling rate. Note that a sampling rate that is too large will result in large amount of data while a sampling rate that is too small cannot capture all the information in the 5s portion. Therefore, in this paper, a rate of 0.1s is used. Therefore the signal matrix will be sampled 51 times. The resulting frequency matrix would have manageable size as well as enough information.

3. Perform Short Time Fourier Transform

To perform STFT, we use a for loop that sweeps from the beginning of the sample to the end and stops at each sample spot. At each spot, we calculate the *Gaussian Filter* and multiply the filter with the signal. Then we Fourier transform the filtered signal to transform it into range of frequencies and shift the center to 0. We also chose to resample the signals, i.e. selecting 1 out of every 10 frequencies to reduce the size of the data by 1/10. Then we append the information to the end of the frequency matrix ***aSpec***. So that the first 51 rows of ***aSpec*** represent the frequency information for the first song, rows 52 to 102 represent the info for the second song and so on.

4.Constructing Metadata Matrix X

After repeating the steps 1-3 for each song of the same genre, we put all the frequency information matrices together to form a single matrix ***X***. The first 1/10 columns of ***X*** represents the info for the first song, the 1/10+1 to 2/10 columns of ***X*** represent the info for the second song and so on.

5. Perform PCA on X

Matrix ***X*** holds the information for the band, so we perform PCA on ***X*** to obtain the principal modes that can represent the genre of the band. First of all, we subtract the mean from ***X*** to normalize each songs. Then we perform economy SVD on the normalized matrix to get the unitary matrix ***U***. Then we calculate the principal component basis ***Y*** by multiplying transpose of ***U*** with normalized ***X***. The principal modes of ***X*** are stored in the diagonal of ***Y*** from largest to smallest.

6. Constructing Classification Matrix

After repeating steps 1-4 for each band, we obtain three matrices ***xa***, ***xb*** and ***xc***. We align the three matrices together and perform PCA on this overall matrix. Then obtain matrix ***Y*** just as step 5 mentioned. The diagonal of ***Y*** contains 51 principal modes but in this paper, we only use the first 14 modes to reduce data size. 14 modes are able to capture enough genre's information.

We have decided to use the first 14 modes of ***Y*** for the classification algorithm. Therefore, we extract the first 10 columns of ***Y*** to form matrix ***x1***, the 11 to 20 columns of ***Y*** to form matrix ***x2***, the 21 to 30 columns of ***Y*** to form matrix ***x3***. ***x1, x2*** and ***x3*** are the information matrices for each band. Each column of ***x1*** represents the 14 principal modes for one song from the first band. Each column of ***x2*** represents the 14 principal modes for one song from the second band and so on.

8.Obtain Training Set and Test Set
We randomly select 10 numbers, each number corresponds to one column of a classification matrix. Then we choose the first 6 random columns to be the training set the the last 4 random columns to be the test set. Then we put the training set and test set for each genre together to form the overall training set ***xtrain*** and overall testing set ***xtest***. We also define the ***Group*** set by assigning 1s to the first genre, 2s to the second genre and 3s to the third genre. When plotting the result, a song classifies as the first genre should have height 1.

9. Classification
After obtaining the training set, test set and the group to train to, we simply use 'classify' to perform discriminant analysis. The 'classify' function will compare the principle modes of the songs in the test set to the characteristics of the principal modes of the songs in the training set and determine which genre it belongs to. Then we cross validate the result and use an average accuracy to evaluate the performance of the algorithm.

10. Variation in Test 2
The methodology is identical to the steps 1-9 for test 1 and only song selection changed. In test 1, three different bands from three different genres are tested, but in test 2, three different bands from the same genre are tested.

11. Variation in Test 3
The methodology is identical to the steps 1-9 for test 1 and only song selections changed.In this test, songs from different bands within the three selected genres are tested, regardless of the bands they are from.

IV. Computational Results

Test 1. Band Classification

In test 1, we aim to differentiate the songs of each band of different genres. In this paper, songs of artists **Amy Winehouse** (jazz), **Hans Zimmer** (classic), and **MUSE** (rock) are used. 6 songs of each band are used to train and 4 songs from each band are tested. Therefore, a total of 12 songs are used to test the accuracy of this algorithm. 6 results are obtained for cross validation. The cross validated result is as following:

Overall accuracy: 73.94%
Accuracy for Amy Winehouse: 79.16%
Accuracy for Hans Zimmer: 62.5%
Accuracy for MUSE: 79.16%

Comparing with the principal modes of each band shown in Figure 2, we can see that accuracy is higher for the bands with principal modes that maintain high energy and worse for the bands with principal modes that maintain middle energy.

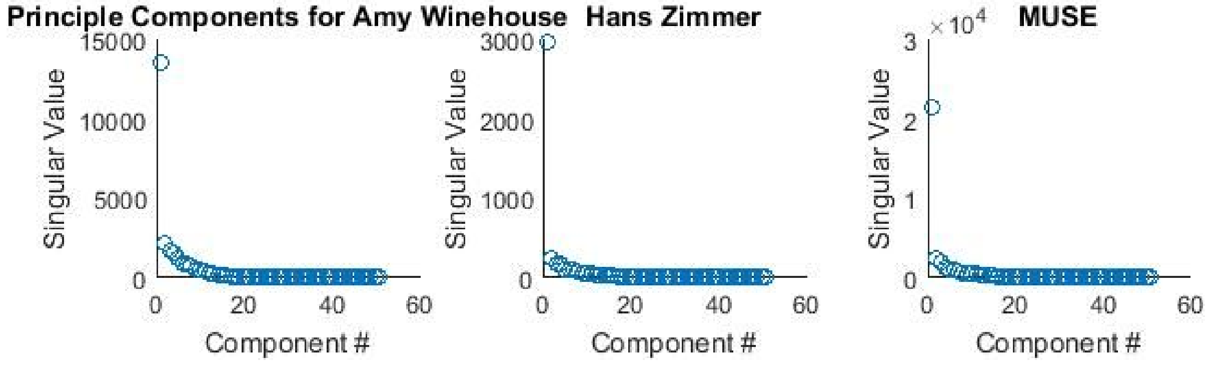

Figure 2. Principal modes for three bands from different genres.

Test 2. The Case for Seattle

In this test, three bands from the same genre are tested, thus making the test more challenging. In this paper, three bands: **MUSE**, **Arctic Monkeys** and **Radiohead** that play rock musics are selected. Same as test 1, training set is constructed with a total of 18 songs, 6 songs from each band. Total of 12 songs are used to test. 6 results are carried out for cross validation. The cross validated result is as following:

Overall accuracy: 52.78%
Accuracy for MUSE: 50%
Accuracy for Arctic Monkeys: 41.6%
Accuracy for Radiohead: 66.67%

Notice that the overall accuracy is lower in test 2 because the features for the principal components are similar to each other (Figure 3), making it harder for the algorithm to characterize using existing groups. Accuracy for the bands MUSE and Arctic Monkeys are lower than the other one because these two bands share very similar principal modes. The reason for a poorer performance of the algorithm is that there are less difference between the style of these three bands.

Test 3. Genre Classification

This test aims to classify each music by genre. In this paper, three genres: Alternative, Classical and Hip-Hop are chosen to be the genres tested. Each genre contains 10 samples of songs from different artists. Again, 6 results are used to cross validate accuracy. The cross validated result is as follows:

Overall accuracy: 77.78%
Accuracy for Alternative: 79.17%
Accuracy for Classical: 75%
Accuracy for Hip Hop: 79.17%

As we can see in Figure 4, the principal modes for each genre has pretty distinct features. Therefore the performance of the algorithm is better than in test 2. Similarly, the genre with principal modes dominating the middle energy range is harder to classify than the ones dominating high energy range.

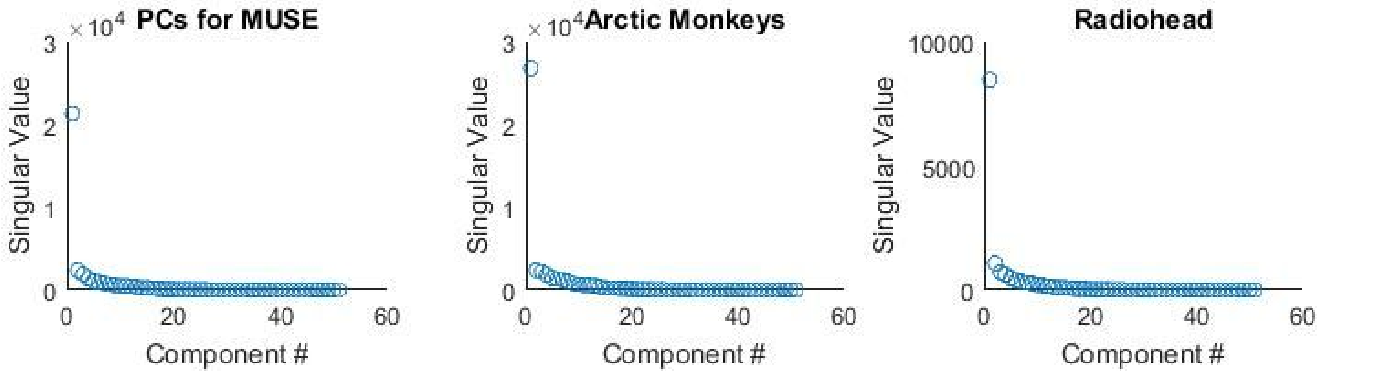

Figure 3. Principal modes for three rock bands.

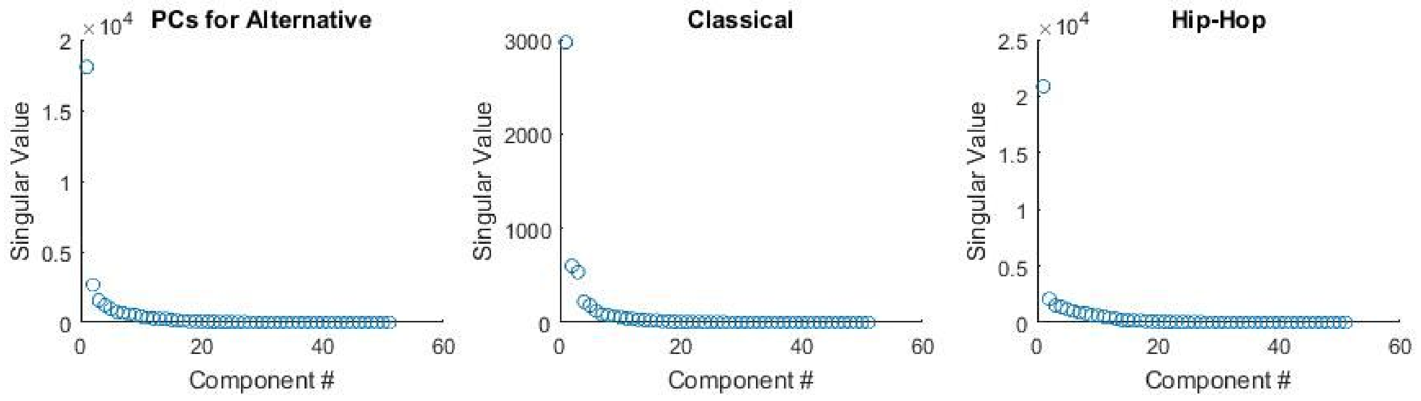

Figure 4.Principal modes for three different genres.

V. Summary and Conclusion

In this paper, the theoretical background and implementation of a music classification algorithm are explained. This algorithm utilized the Short Time Fourier Transform technique, Principal Component Analysis and Discriminant Analysis to classify music genres. This paper also evaluated the performance of the algorithm. For test 1, songs from three bands of different genres are tested, resulting in an accuracy of 73.94%. For test 2, songs from three bands of same genre are tested, resulting in an accuracy of 52.78%. For test 3, songs of different genres are tested and classified as different genres, resulting in an accuracy of 77.78%. We conclude that accuracy increases when the training sets are more distinct and independent from each other. It is also higher for the genres with principal modes that dominates high energy range.

VI. Bibliorgraphy

[1] E. Jacobsen and R. Lyons, The sliding DFT, *Signal Processing Magazine* vol. 20, issue 2, pp. 74–80 (March 2003).

[2] David W. Stockburger. *Multivariate Statistics: Concepts, Models, and Applications.* 1996.

Appendix A. Matlab Functions

audioread - Reads in audio files.
fft - fourier transform.
fftshift - shift the frequency domain to center at 0.
abs - takes the absolute value of the variable.
resample - altematically selects a portion of the variable to form a smaller matrix.
svd - singular value decomposition.
diag - takes the diagoal of the variable and put it in a column vector.
classify - classify the given test set into groups based on the features observed in the principal components of the training set.
bar - draw the result vector as a bar graph.

Appendix B. Matlab Codes

```
%% Test 1
[a1, aFs1] = audioread('AW1.mp3');
[a2, aFs2] = audioread('AW2.mp3');
[a3, aFs3] = audioread('AW3.mp3');
[a4, aFs4] = audioread('AW4.mp3');
[a5, aFs5] = audioread('AW5.mp3');

[b1, bFs1] = audioread('HZ1.mp3');
[b2, bFs2] = audioread('HZ2.mp3');
[b3, bFs3] = audioread('HZ3.mp3');
[b4, bFs4] = audioread('HZ4.mp3');
[b5, bFs5] = audioread('HZ5.mp3');

[c1, cFs1] = audioread('MUSE1.mp3');
[c2, cFs2] = audioread('MUSE2.mp3');
[c3, cFs3] = audioread('MUSE3.mp3');
[c4, cFs4] = audioread('MUSE4.mp3');
[c5, cFs5] = audioread('MUSE5.mp3');
%% Average Channels
Fs = aFs1;

a1com = (a1(:, 1) + a1(:, 2))/2;
a1Seg = a1com(Fs*60 + 1:Fs*65);
a2com = (a2(:, 1) + a2(:, 2))/2;
a2Seg = a2com(Fs*60 + 1:Fs*65);
a3com = (a3(:, 1) + a3(:, 2))/2;
a3Seg = a3com(Fs*60 + 1:Fs*65);
a4com = (a4(:, 1) + a4(:, 2))/2;
a4Seg = a4com(Fs*60 + 1:Fs*65);
a5com = (a5(:, 1) + a5(:, 2))/2;
a5Seg = a5com(Fs*60 + 1:Fs*65);
a6Seg = a1com(Fs*66 + 1:Fs*71);
a7Seg = a2com(Fs*66 + 1:Fs*71);
a8Seg = a3com(Fs*66 + 1:Fs*71);
a9Seg = a4com(Fs*66 + 1:Fs*71);
a10Seg = a5com(Fs*66 + 1:Fs*71);

b1com = (b1(:, 1) + b1(:, 2))/2;
b1Seg = b1com(Fs*60 + 1:Fs*65);
b2com = (b2(:, 1) + b2(:, 2))/2;
b2Seg = b2com(Fs*60 + 1:Fs*65);
b3com = (b3(:, 1) + b3(:, 2))/2;
b3Seg = b3com(Fs*60 + 1:Fs*65);
b4com = (b4(:, 1) + b4(:, 2))/2;
b4Seg = b4com(Fs*60 + 1:Fs*65);
b5com = (b5(:, 1) + b5(:, 2))/2;
b5Seg = b5com(Fs*60 + 1:Fs*65);
b6Seg = b1com(Fs*66 + 1:Fs*71);
b7Seg = b2com(Fs*66 + 1:Fs*71);
```

```
b8Seg = b3com(Fs*66 + 1:Fs*71);
b9Seg = b4com(Fs*66 + 1:Fs*71);
b10Seg = b5com(Fs*66 + 1:Fs*71);


c1com = (c1(:, 1) + c1(:, 2))/2;
c1Seg = c1com(Fs*60 + 1:Fs*65);
c2com = (c2(:, 1) + c2(:, 2))/2;
c2Seg = c2com(Fs*60 + 1:Fs*65);
c3com = (c3(:, 1) + c3(:, 2))/2;
c3Seg = c3com(Fs*60 + 1:Fs*65);
c4com = (c4(:, 1) + c4(:, 2))/2;
c4Seg = c4com(Fs*60 + 1:Fs*65);
c5com = (c5(:, 1) + c5(:, 2))/2;
c5Seg = c5com(Fs*60 + 1:Fs*65);
c6Seg = c1com(Fs*66 + 1:Fs*71);
c7Seg = c2com(Fs*66 + 1:Fs*71);
c8Seg = c3com(Fs*66 + 1:Fs*71);
c9Seg = c4com(Fs*66 + 1:Fs*71);
c10Seg = c5com(Fs*66 + 1:Fs*71);
%% Combining songs
aSeg = [a1Seg' ; a2Seg'; a3Seg'; a4Seg'; a5Seg'; a6Seg'; a7Seg'; a8Seg';
a9Seg'; a10Seg'];
bSeg = [b1Seg' ; b2Seg'; b3Seg'; b4Seg'; b5Seg'; b6Seg'; b7Seg'; b8Seg';
b9Seg'; b10Seg'];
cSeg = [c1Seg' ; c2Seg'; c3Seg'; c4Seg'; c5Seg'; c6Seg'; c7Seg'; c8Seg';
c9Seg'; c10Seg'];

%%
L = 10;
n = 5*Fs;
rate = 0.1;
tslide = 0:rate:5;
window = -20;

%%
xa = [];
for i = 1:L
  ai = aSeg(i, :);
  aSpec = [];
  t = (1:length(ai))/Fs;
  for j = 1:length(tslide)
        g = exp(window*(t-tslide(j)).^2); %Guassian Filter
        aG = g.*ai;
        aG_fft = fft(aG);
        aSpec = [aSpec; resample(abs(fftshift(aG_fft)), 1, 10)];
  end
  xa = [xa, aSpec];
end
%% pca for a
[m, n] = size(xa);
```

```
mn = mean(xa, 2);
xa_sub=xa-repmat(mn, 1, n);
[u, s, v] = svd(xa_sub/sqrt(n-1), 'econ');
y = u'*xa_sub;
Cya = y*y.'/(n-1);
PCa = diag(Cya);
save PCa.mat PCa;
figure (1);
subplot(1, 3, 1);
scatter(1:length(PCa), PCa);
title('Principle Components for Amy Winehouse');
xlabel('Component #');
ylabel('Singular Value');
%%
xb = [];
for i = 1:L
  bi = bSeg(i, :);
  bSpec = [];
  t = (1:length(bi))/Fs;
  for j = 1:length(tslide)
        g = exp(window*(t-tslide(j)).^2); %Guassian Filter
        bG = g.*bi;
        bG_fft = fft(bG);
        bSpec = [bSpec; resample(abs(fftshift(bG_fft)), 1, 10)];
  end
  xb = [xb, bSpec];
end
%% pca for b
[m, n] = size(xb);
mn = mean(xb, 2);
xb_sub=xb-repmat(mn, 1, n);
[u, s, v] = svd(xb_sub/sqrt(n-1), 'econ');
y = u'*xb_sub;
Cyb = y*y.'/(n-1);
PCb = diag(Cyb);
save PCb.mat PCb;
subplot(1, 3, 2);
scatter(1:length(PCb), PCb);
title('Hans Zimmer');
xlabel('Component #');
ylabel('Singular Value');
%%
xc = [];
for i = 1:L
  ci = cSeg(i, :);
  cSpec = [];
  t = (1:length(ci))/Fs;
  for j = 1:length(tslide)
        g = exp(window*(t-tslide(j)).^2); %Guassian Filter
        cG = g.*ci;
        cG_fft = fft(cG);
        cSpec = [cSpec; resample(abs(fftshift(cG_fft)), 1, 10)];
  end
  xc = [xc, cSpec];
end
%% pca for c
[m, n] = size(xc);
mn = mean(xc, 2);
xc_sub=xc-repmat(mn, 1, n);
[u, s, v] = svd(xc_sub/sqrt(n-1), 'econ');
y = u'*xc_sub;
Cyc = y*y.'/(n-1);
PCc = diag(Cyc);
save PCc.mat PCc;
subplot(1, 3, 3);
scatter(1:length(PCc), PCc);
title('MUSE');
xlabel('Component #');
ylabel('Singular Value');
%% pca
x = [xa, xb, xc];
[m, n] = size(x);
mn = mean(x, 2);
x_sub=x-repmat(mn, 1, n);
[u, s, v] = svd(x_sub/sqrt(n-1), 'econ');
y = u'*x_sub;

%% classification
rand = randperm (L);
x1 = y(1:14, 1:L);
x2 = y(1:14, L+1:2*L);
x3 = y(1:14, 2*L+1:3*L);

xtrain = [x1(:, rand(1:6)), x2(:, rand(1:6)), x3(:, rand(1:6))];
xtest = [x1(:, rand(7:10)), x2(:,rand(7:10)), x3(:, rand(7:10))];
trainInto = [ones(1, 6), 2*ones(1, 6), 3*ones(1, 6)];

result = knnclassify(xtest', xtrain', trainInto);
figure(2);
bar(result);
title('Classification');
```

```matlab
%% Test 2
[a1, aFs1] = audioread('MUSE1.mp3');
[a2, aFs2] = audioread('MUSE2.mp3');
[a3, aFs3] = audioread('MUSE3.mp3');
[a4, aFs4] = audioread('MUSE4.mp3');
[a5, aFs5] = audioread('MUSE5.mp3');

[b1, bFs1] = audioread('AM1.mp3');
[b2, bFs2] = audioread('AM2.mp3');
[b3, bFs3] = audioread('AM3.mp3');
[b4, bFs4] = audioread('AM4.mp3');
[b5, bFs5] = audioread('AM5.mp3');

[c1, cFs1] = audioread('RH1.m4a');
[c2, cFs2] = audioread('RH2.m4a');
[c3, cFs3] = audioread('RH3.m4a');
[c4, cFs4] = audioread('RH4.mp3');
[c5, cFs5] = audioread('RH5.mp3');
%%
Fs = aFs1;

a1com = (a1(:, 1) + a1(:, 2))/2;
a1Seg = a1com(Fs*60 + 1:Fs*65);
a2com = (a2(:, 1) + a2(:, 2))/2;
a2Seg = a2com(Fs*60 + 1:Fs*65);
a3com = (a3(:, 1) + a3(:, 2))/2;
a3Seg = a3com(Fs*60 + 1:Fs*65);
a4com = (a4(:, 1) + a4(:, 2))/2;
a4Seg = a4com(Fs*60 + 1:Fs*65);
a5com = (a5(:, 1) + a5(:, 2))/2;
a5Seg = a5com(Fs*60 + 1:Fs*65);
a6Seg = a1com(Fs*66 + 1:Fs*71);
a7Seg = a2com(Fs*66 + 1:Fs*71);
a8Seg = a3com(Fs*66 + 1:Fs*71);
a9Seg = a4com(Fs*66 + 1:Fs*71);
a10Seg = a5com(Fs*66 + 1:Fs*71);

b1com = (b1(:, 1) + b1(:, 2))/2;
b1Seg = b1com(Fs*60 + 1:Fs*65);
b2com = (b2(:, 1) + b2(:, 2))/2;
b2Seg = b2com(Fs*60 + 1:Fs*65);
b3com = (b3(:, 1) + b3(:, 2))/2;
b3Seg = b3com(Fs*60 + 1:Fs*65);
b4com = (b4(:, 1) + b4(:, 2))/2;
b4Seg = b4com(Fs*60 + 1:Fs*65);
b5com = (b5(:, 1) + b5(:, 2))/2;
b5Seg = b5com(Fs*60 + 1:Fs*65);
b6Seg = b1com(Fs*66 + 1:Fs*71);
b7Seg = b2com(Fs*66 + 1:Fs*71);
b8Seg = b3com(Fs*66 + 1:Fs*71);
```

```
b9Seg = b4com(Fs*66 + 1:Fs*71);
b10Seg = b5com(Fs*66 + 1:Fs*71);


c1com = (c1(:, 1) + c1(:, 2))/2;
c1Seg = c1com(Fs*60 + 1:Fs*65);
c2com = (c2(:, 1) + c2(:, 2))/2;
c2Seg = c2com(Fs*60 + 1:Fs*65);
c3com = (c3(:, 1) + c3(:, 2))/2;
c3Seg = c3com(Fs*60 + 1:Fs*65);
c4com = (c4(:, 1) + c4(:, 2))/2;
c4Seg = c4com(Fs*60 + 1:Fs*65);
c5com = (c5(:, 1) + c5(:, 2))/2;
c5Seg = c5com(Fs*60 + 1:Fs*65);
c6Seg = c1com(Fs*66 + 1:Fs*71);
c7Seg = c2com(Fs*66 + 1:Fs*71);
c8Seg = c3com(Fs*66 + 1:Fs*71);
c9Seg = c4com(Fs*66 + 1:Fs*71);
c10Seg = c5com(Fs*66 + 1:Fs*71);
%%
aSeg = [a1Seg' ; a2Seg'; a3Seg'; a4Seg'; a5Seg'; a6Seg'; a7Seg'; a8Seg';
a9Seg'; a10Seg'];
bSeg = [b1Seg' ; b2Seg'; b3Seg'; b4Seg'; b5Seg'; b6Seg'; b7Seg'; b8Seg';
b9Seg'; b10Seg'];
cSeg = [c1Seg' ; c2Seg'; c3Seg'; c4Seg'; c5Seg'; c6Seg'; c7Seg'; c8Seg';
c9Seg'; c10Seg'];

%%
L = 10;
n = 5*Fs;
rate = 0.1;
tslide = 0:rate:5;
window = -20;
%% Windowed Fourier Transform Plot
 ai = aSeg(1, :);
 aSpec = [];
 for j = 1:length(tslide)
        t = (1:length(ai))/Fs;
        g = exp(window*(t-tslide(j)).^2); %Guassian Filter
        aG = g.*ai;
        aG_fft = fft(aG);
        aSpec = [aSpec; resample(abs(fftshift(aG_fft)), 1, 10)];
        subplot(3, 1, 1);
        plot(t, ai,'k',t, g,'r');
        subplot(3, 1, 2);
        plot(t, aG);
        n = 220500;
        k = (2*pi)/5*[0:n/2-1 -n/2:-1];
        ks = fftshift(k);
        subplot(3, 1, 3);
        aGt = aG_fft(1, n/2:n);
        plot(ks(n/2:n), abs(fftshift(aGt)/max(abs(aGt))));
        drawnow;
 end
%%
xa = [];
for i = 1:L
 ai = aSeg(i, :);
 aSpec = [];
 for j = 1:length(tslide)
        t = (1:length(ai))/Fs;
        g = exp(window*(t-tslide(j)).^2); %Guassian Filter
        aG = g.*ai;
        aG_fft = fft(aG);
        aSpec = [aSpec; resample(abs(fftshift(aG_fft)), 1, 10)];
 end
 xa = [xa, aSpec];
end
%% pca for a
[m, n] = size(xa);
mn = mean(xa, 2);
xa_sub=xa-repmat(mn, 1, n);
[u, s, v] = svd(xa_sub/sqrt(n-1), 'econ');
y = u'*xa_sub;
Cya = y*y.'/(n-1);
PCa = diag(Cya);
save PCa.mat PCa;
figure (1);
subplot(1, 3, 1);
scatter(1:length(PCa), PCa);
title('PCs for MUSE');
xlabel('Component #');
ylabel('Singular Value');
%%
xb = [];
for i = 1:L
 bi = bSeg(i, :);
 bSpec = [];
 for j = 1:length(tslide)
        t = (1:length(bi))/Fs;
        g = exp(window*(t-tslide(j)).^2); %Guassian Filter
        bG = g.*bi;
        bG_fft = fft(bG);
        bSpec = [bSpec; resample(abs(fftshift(bG_fft)), 1, 10)];
 end
 xb = [xb, bSpec];
end
%% pca for b
[m, n] = size(xb);
mn = mean(xb, 2);
xb_sub=xb-repmat(mn, 1, n);
[u, s, v] = svd(xb_sub/sqrt(n-1), 'econ');
```

```
y = u'*xb_sub;
Cyb = y*y.'/(n-1);
PCb = diag(Cyb);
save PCb.mat PCb;
subplot(1, 3, 2);
scatter(1:length(PCb), PCb);
title('Arctic Monkeys');
xlabel('Component #');
ylabel('Singular Value');
%%
xc = [];
for i = 1:L
  ci = cSeg(i, :);
  cSpec = [];
  for j = 1:length(tslide)
        t = (1:length(ci))/Fs;
        g = exp(window*(t-tslide(j)).^2); %Guassian Filter
        cG = g.*ci;
        cG_fft = fft(cG);
        cSpec = [cSpec; resample(abs(fftshift(cG_fft)), 1, 10)];
  end
  xc = [xc, cSpec];
end
%% pca for c
[m, n] = size(xc);
mn = mean(xc, 2);
xc_sub=xc-repmat(mn, 1, n);
[u, s, v] = svd(xc_sub/sqrt(n-1), 'econ');
y = u'*xc_sub;
Cyc = y*y.'/(n-1);
PCc = diag(Cyc);
save PCc.mat PCc;
subplot(1, 3, 3);
scatter(1:length(PCc), PCc);
title('Radiohead');
xlabel('Component #');
ylabel('Singular Value');
%% pca
x = [xa, xb, xc];
[m, n] = size(x);
mn = mean(x, 2);
x_sub=x-repmat(mn, 1, n);
[u, s, v] = svd(x_sub/sqrt(n-1), 'econ');
y = u'*x_sub;

%%
rand = randperm (L);
x1 = y(1:14, 1:L);
x2 = y(1:14, L+1:2*L);
x3 = y(1:14, 2*L+1:3*L);
```

```
xtrain = [x1(:, rand(1:6)), x2(:, rand(1:6)), x3(:, rand(1:6))];
xtest = [x1(:, rand(7:10)), x2(:,rand(7:10)), x3(:, rand(7:10))];
trainInto = [ones(1, 6), 2*ones(1, 6), 3*ones(1, 6)];

result = classify(xtest', xtrain', trainInto);
figure(2);
bar(result);
title('Classification');
```

```
%% Test 3

%%Alternative
[a1, aFs1] = audioread('AL1.mp3');
[a2, aFs2] = audioread('AL2.mp3');
[a3, aFs3] = audioread('AL3.m4a');
[a4, aFs4] = audioread('AL4.m4a');
[a5, aFs5] = audioread('AL5.mp3');

%%Classical
[b1, bFs1] = audioread('CL1.mp3');
[b2, bFs2] = audioread('CL2.mp3');
[b3, bFs3] = audioread('CL3.mp3');
[b4, bFs4] = audioread('CL4.mp3');
[b5, bFs5] = audioread('CL5.mp3');

%%Hip-Hop
[c1, cFs1] = audioread('HH1.mp3');
[c2, cFs2] = audioread('HH2.m4a');
[c3, cFs3] = audioread('HH3.mp3');
[c4, cFs4] = audioread('HH4.mp3');
[c5, cFs5] = audioread('HH5.mp3');
%%
Fs = aFs1;

a1com = (a1(:, 1) + a1(:, 2))/2;
a1Seg = a1com(Fs*60 + 1:Fs*65);
a2com = (a2(:, 1) + a2(:, 2))/2;
a2Seg = a2com(Fs*60 + 1:Fs*65);
a3com = (a3(:, 1) + a3(:, 2))/2;
a3Seg = a3com(Fs*60 + 1:Fs*65);
a4com = (a4(:, 1) + a4(:, 2))/2;
a4Seg = a4com(Fs*60 + 1:Fs*65);
a5com = (a5(:, 1) + a5(:, 2))/2;
a5Seg = a5com(Fs*60 + 1:Fs*65);
a6Seg = a1com(Fs*66 + 1:Fs*71);
a7Seg = a2com(Fs*66 + 1:Fs*71);
a8Seg = a3com(Fs*66 + 1:Fs*71);
a9Seg = a4com(Fs*66 + 1:Fs*71);
a10Seg = a5com(Fs*66 + 1:Fs*71);

b1com = (b1(:, 1) + b1(:, 2))/2;
b1Seg = b1com(Fs*60 + 1:Fs*65);
b2com = (b2(:, 1) + b2(:, 2))/2;
b2Seg = b2com(Fs*60 + 1:Fs*65);
b3com = (b3(:, 1) + b3(:, 2))/2;
b3Seg = b3com(Fs*60 + 1:Fs*65);
b4com = (b4(:, 1) + b4(:, 2))/2;
b4Seg = b4com(Fs*60 + 1:Fs*65);
b5com = (b5(:, 1) + b5(:, 2))/2;
b5Seg = b5com(Fs*60 + 1:Fs*65);
```

```
b6Seg = b1com(Fs*66 + 1:Fs*71);
b7Seg = b2com(Fs*66 + 1:Fs*71);
b8Seg = b3com(Fs*66 + 1:Fs*71);
b9Seg = b4com(Fs*66 + 1:Fs*71);
b10Seg = b5com(Fs*66 + 1:Fs*71);


c1com = (c1(:, 1) + c1(:, 2))/2;
c1Seg = c1com(Fs*60 + 1:Fs*65);
c2com = (c2(:, 1) + c2(:, 2))/2;
c2Seg = c2com(Fs*60 + 1:Fs*65);
c3com = (c3(:, 1) + c3(:, 2))/2;
c3Seg = c3com(Fs*60 + 1:Fs*65);
c4com = (c4(:, 1) + c4(:, 2))/2;
c4Seg = c4com(Fs*60 + 1:Fs*65);
c5com = (c5(:, 1) + c5(:, 2))/2;
c5Seg = c5com(Fs*60 + 1:Fs*65);
c6Seg = c1com(Fs*66 + 1:Fs*71);
c7Seg = c2com(Fs*66 + 1:Fs*71);
c8Seg = c3com(Fs*66 + 1:Fs*71);
c9Seg = c4com(Fs*66 + 1:Fs*71);
c10Seg = c5com(Fs*66 + 1:Fs*71);
%%
aSeg = [a1Seg' ; a2Seg'; a3Seg'; a4Seg'; a5Seg'; a6Seg'; a7Seg'; a8Seg'; a9Seg';
a10Seg'];
bSeg = [b1Seg' ; b2Seg'; b3Seg'; b4Seg'; b5Seg'; b6Seg'; b7Seg'; b8Seg'; b9Seg';
b10Seg'];
cSeg = [c1Seg' ; c2Seg'; c3Seg'; c4Seg'; c5Seg'; c6Seg'; c7Seg'; c8Seg'; c9Seg';
c10Seg'];

%%
L = 10;
n = 5*Fs;
rate = 0.1;
tslide = 0:rate:5;
window = -20;
%% Windowed Fourier Transform Plot
 ai = aSeg(1, :);
 aSpec = [];
 for j = 1:length(tslide)
      t = (1:length(ai))/Fs;
      g = exp(window*(t-tslide(j)).^2); %Guassian Filter
      aG = g.*ai;
      aG_fft = fft(aG);
      aSpec = [aSpec; resample(abs(fftshift(aG_fft)), 1, 10)];
      subplot(3, 1, 1);
      plot(t, ai,'k',t, g,'r');
      subplot(3, 1, 2);
      plot(t, aG);
      n = 220500;
      k = (2*pi)/5*[0:n/2-1 -n/2:-1];
```

```
        ks = fftshift(k);
        subplot(3, 1, 3);
        aGt = aG_fft(1, n/2:n);
        plot(ks(n/2:n), abs(fftshift(aGt)/max(abs(aGt))));
        drawnow;
        end
%%
xa = [];
for i = 1:L
        ai = aSeg(i, :);
        aSpec = [];
        for j = 1:length(tslide)
        t = (1:length(ai))/Fs;
        g = exp(window*(t-tslide(j)).^2); %Guassian Filter
        aG = g.*ai;
        aG_fft = fft(aG);
        aSpec = [aSpec; resample(abs(fftshift(aG_fft)), 1, 10)];
        end
        xa = [xa, aSpec];
end
%% pca for a
[m, n] = size(xa);
mn = mean(xa, 2);
xa_sub=xa-repmat(mn, 1, n);
[u, s, v] = svd(xa_sub/sqrt(n-1), 'econ');
y = u'*xa_sub;
Cya = y*y.'/(n-1);
PCa = diag(Cya);
save PCa.mat PCa;
figure (1);
subplot(1, 3, 1);
scatter(1:length(PCa), PCa);
title('PCs for Alternative');
xlabel('Component #');
ylabel('Singular Value');
%%
xb = [];
for i = 1:L
        bi = bSeg(i, :);
        bSpec = [];
        for j = 1:length(tslide)
        t = (1:length(bi))/Fs;
        g = exp(window*(t-tslide(j)).^2); %Guassian Filter
        bG = g.*bi;
        bG_fft = fft(bG);
        bSpec = [bSpec; resample(abs(fftshift(bG_fft)), 1, 10)];
        end
        xb = [xb, bSpec];
end
%% pca for b
[m, n] = size(xb);
```

```
mn = mean(xb, 2);
xb_sub=xb-repmat(mn, 1, n);
[u, s, v] = svd(xb_sub/sqrt(n-1), 'econ');
y = u'*xb_sub;
Cyb = y*y.'/(n-1);
PCb = diag(Cyb);
save PCb.mat PCb;
subplot(1, 3, 2);
scatter(1:length(PCb), PCb);
title('Classical');
xlabel('Component #');
ylabel('Singular Value');
%%
xc = [];
for i = 1:L
      ci = cSeg(i, :);
      cSpec = [];
      for j = 1:length(tslide)
      t = (1:length(ci))/Fs;
      g = exp(window*(t-tslide(j)).^2); %Guassian Filter
      cG = g.*ci;
      cG_fft = fft(cG);
      cSpec = [cSpec; resample(abs(fftshift(cG_fft)), 1, 10)];
      end
      xc = [xc, cSpec];
end
%% pca for c
[m, n] = size(xc);
mn = mean(xc, 2);
xc_sub=xc-repmat(mn, 1, n);
[u, s, v] = svd(xc_sub/sqrt(n-1), 'econ');
y = u'*xc_sub;
Cyc = y*y.'/(n-1);
PCc = diag(Cyc);
save PCc.mat PCc;
subplot(1, 3, 3);
scatter(1:length(PCc), PCc);
title('Hip-Hop');
xlabel('Component #');
ylabel('Singular Value');
%% pca
x = [xa, xb, xc];
[m, n] = size(x);
mn = mean(x, 2);
x_sub=x-repmat(mn, 1, n);
[u, s, v] = svd(x_sub/sqrt(n-1), 'econ');
y = u'*x_sub;

%%
rand = randperm (L);
x1 = y(1:14, 1:L);
```

```
x2 = y(1:14, L+1:2*L);
x3 = y(1:14, 2*L+1:3*L);

xtrain = [x1(:, rand(1:6)), x2(:, rand(1:6)), x3(:, rand(1:6))];
xtest = [x1(:, rand(7:10)), x2(:,rand(7:10)), x3(:, rand(7:10))];
trainInto = [ones(1, 6), 2*ones(1, 6), 3*ones(1, 6)];

result = classify(xtest', xtrain', trainInto);
figure(2);
bar(result);
title('Classification');
```