# Lab3

# Music Genre Detection

ECEN 4532 Digital Signal Processing Lab

Yiming Wang

University of Colorado Boulder

March/ 3/ 2017

## 1.Introduction

In this lab, we will implement a genre classifier. We will train the machine learning algorithm with the mfcc coefficients or NPCP coefficients of the training set of music and cross-validate it with the testing set.

## 2.Distance matrix D (assignment1,2)

Track-396 still has a strong rhythm pattern. The pattern is a sine wave. Track-370 has a strong pattern as well. It has a bunch of small spikes and three relatively larger spikes. It is consistent with the fact that the song has bas strong beats all the time. Track463 and track-547 have strong patterns too. Track-547 has the same pattern (one large spike in the middle, two smaller one aside) repeating continuously. Track-463 has single spike repeating.

### I. Import 150 songs

In order to provide 150 learning samples for the program, we need some special tricks to do that. In the program, we imported the songs from the 6 directories to workspace. We store them into a sample matrix in the order of: classical, electronic, jazz, punk, rock, world. It is important to store the same genre of songs together because we need to visualize the result of distance between genres later.

### II. MFCC only: merge 40 mfcc channels into 12

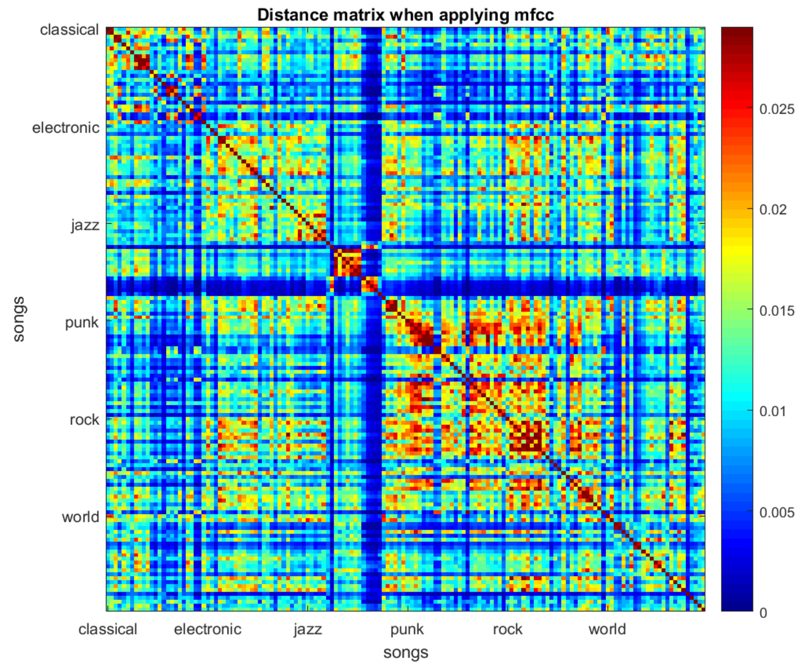
With mfcc method, we can get 40 data points in one single frame. First of all, in order to get the feature of the genre, we discard the information on the time axis, which is the difference between exact songs instead of genres. We compute the mean of each channel on the time axis.

An array of 40 mfccs can now define the feature of a song. But it would be too much for the machine learning algorithm to do. In order to make the computation faster, we merge 40 channels into 12.

### III. Find the distance

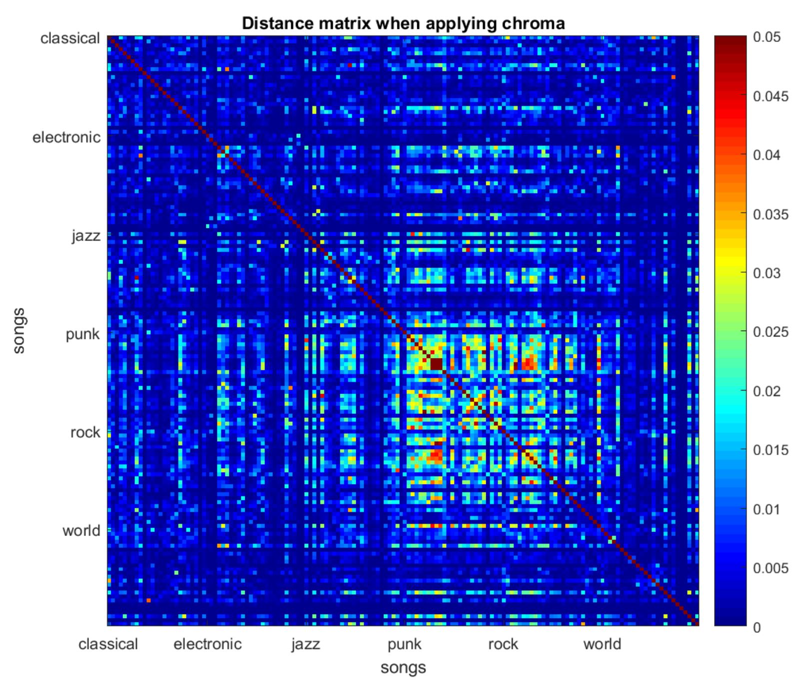
In Probability, Covariance is used to describe the error between 2 random variables. It can describe the distance between features of songs in our case. We assume that different genres have different preferences in the frequency domain. Under this assumption, it makes sense to compute how different are the mfccs or NPCPs among the songs. After we get the covariance between every 2 songs, we compute Kullback-Leibler divergence and rescale the result. We call the result ‘distance’. Here are the results:

a. d matrix when applying mfcc, gamma = 0.5




Comments: From the plot we can see that punk and rock have a very close feature. It could be hard to tell the difference between them. Electronic is also consistent with itself. Classical and Jazz have similar situation. The feature difference in the same genre seems have a relatively larger variance than punk and rock. For the world genre, the songs are like every other genres according to the analysis.

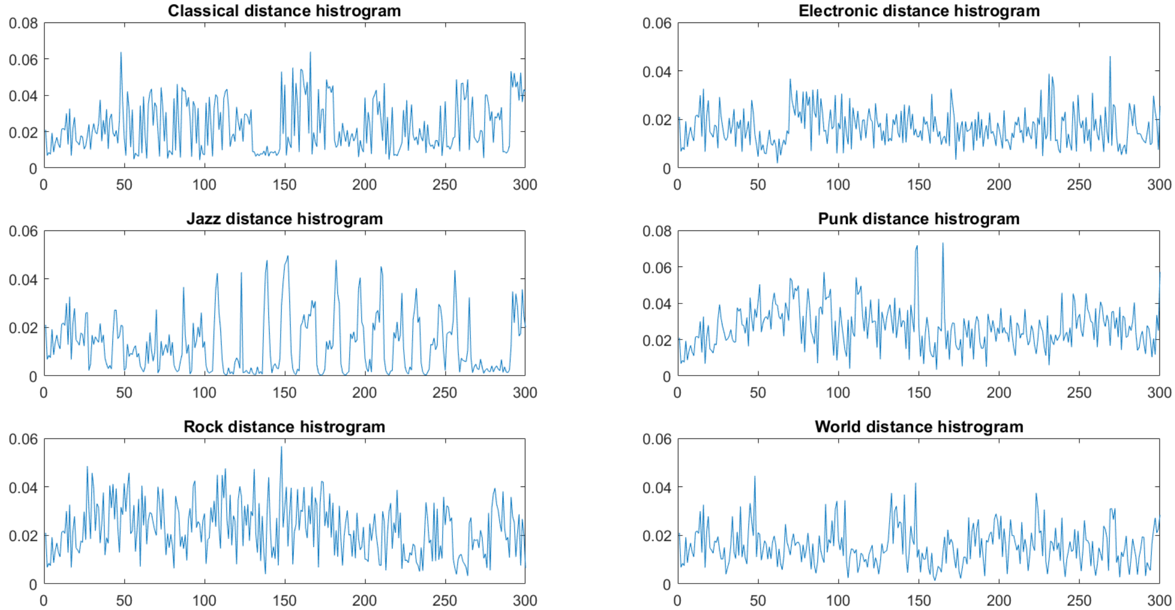
b. d matrix when applying NPCP, gamma = 1




Comments: We can still see that punk and rock songs sound like each other from this plot. Unfortunately, It is hard to distinguish classical, electronic, jazz, or world genre if we use NPCP. Meanwhile, punk songs even look more like rock songs than itself because the magnitude is larger.

### 3. Pairwise distance histogram within each genre (assignment3)

In this section we compute the pairwise distance within a single genre. The purpose is to examine the character consistency of each genre. Each genre has 25 songs. So the amount of non-overlapping pairs for each genre is $\sum_{n=1}^{n=24} n = 300$.




Comment: From the histogram we can tell that Classical, Electronic, Rock and Punk have a relatively good consistency. Classical has the largest amount of highly consistent songs although there are some low consistency songs. Punk and Rock has a nice and stable average distance plot.

### 4. Compute Average distance matrix (assignment4,5,6)

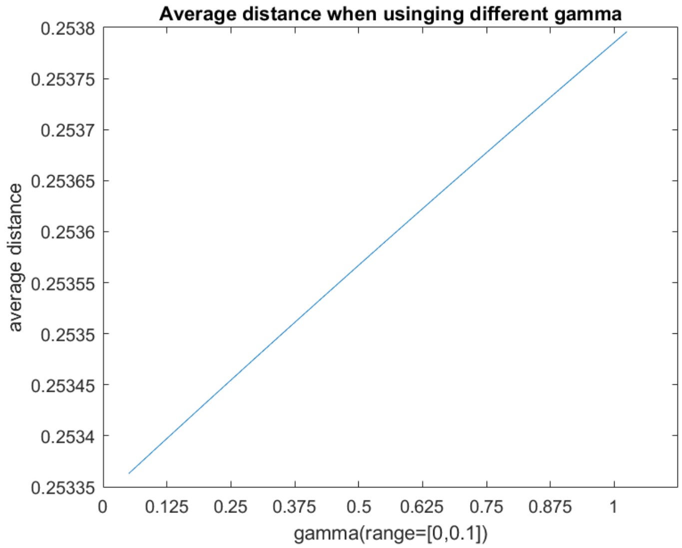
We know that each genre has 25 songs. In order to get a general conclusion from the statistical analysis of the samples, we compute the average of the distance matrix to find distance between genres.

Firstly, we sum the distance of the first song in genre A to the 25 songs in genre B. Then we do the second song, the third song till the 25th song in genre A. After that, we divide the sum by 25^2 to get the average. This is the distance between genre A and genre B.

Then, this process is iterated to compute the distance between every 2 genres.

### 4.I. Experiment with different values of γ (assignment 5)

We researched the way to find gamma which gives us the maximized divergent average distances between genres. The method we use is to take the average of the average distance matrix. We call the value 'D_mean' in the MATLAB code. To normalize D_mean, we divide it by the minimum value in the average distance matrix. Here is a plot of D_mean verses gamma:

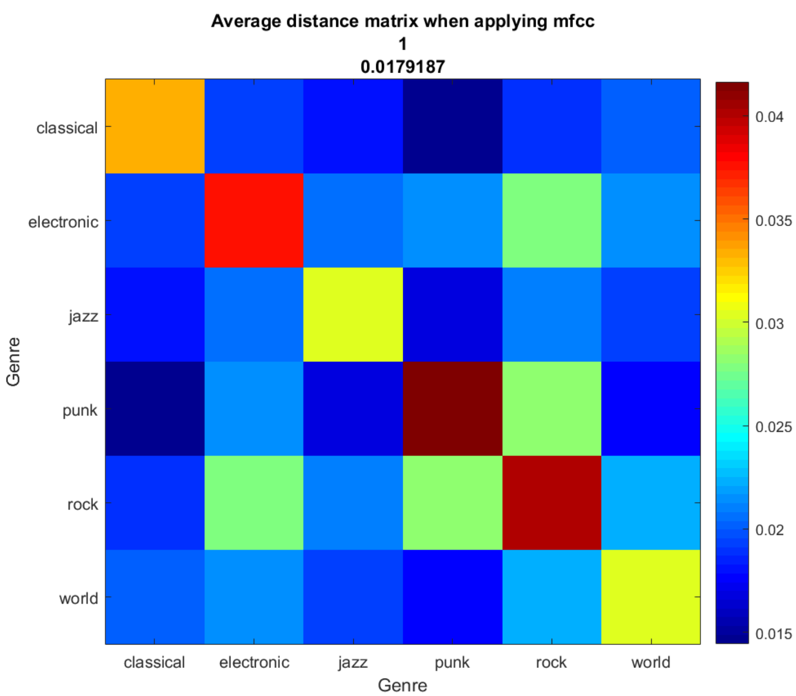



Commants: The plot shows positive correlation between gamma and D_mean. So, in order to get the most separated distances, we should set gamma to be 1.

### 4.II. Compute average distance matrix and comments (assignment 4,6)

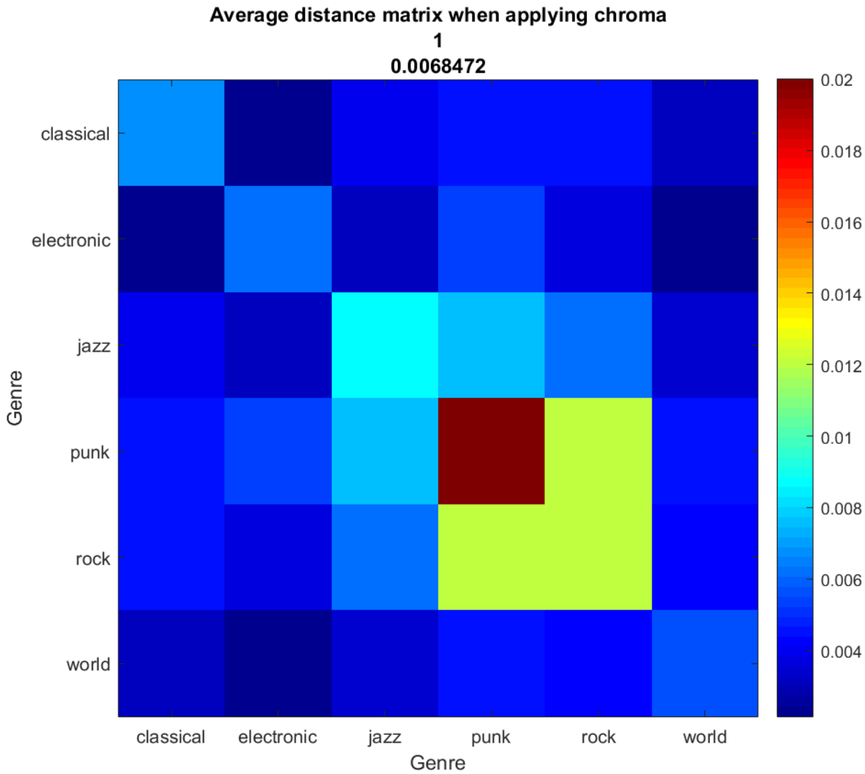
Here are the plots of the average distance matrix:

a. average distance matrix when using mfcc, gamma = 1

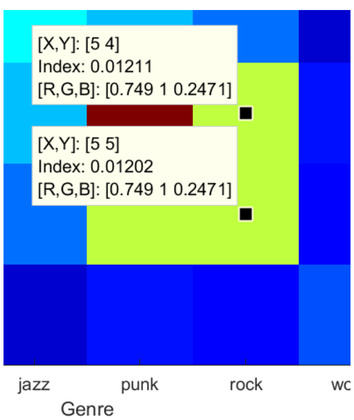



Comments: The plot indicate that mfcc coefficients is good for distinguishing the genre among the 150 songs because the axis has the major vote. Even world genre has noticeable difference compared with other genres.

b. average distance matrix when using NPCP, gamma = 1




Comments: For classical, electronic, jazz and world, the differences compared with other genres are small. That's not the only issue. According to the data, a rock song has higher probability to be recognized as punk. The evidence is as follows:




So, the conclusion is that NPCP is not preferable to determine genres.

### 5. Effect of the length to the separation of average D matrix (assignment 7)

This table shows different track length setup and the corresponding D mean value:

| | | | | |
|---|---|---|---|---|
| secs | 30 | 60 | 120 | 240 |
| D mean | 0.3102 | 0.3092 | 0.2444 | 0.2118 |

From the result we can see that the longer the track is, the smaller D mean would be. Maybe it's because the longer the track is, there are more similarity in these songs. This indicates us that we should extract shorter length to have better genre character distance result.

**6. Classification method (assignment 8,9,10,11)**

We implemented a classifier with K nearest neighbors method. This KNN method is to compute the distance of the test data with the training data set, and determine the major vote within the K amount of nearest neighbors to be the genre. The distance in this lab is the distance between the mfcc coefficients or NPCP coefficients.

In the previous part, we have already computed the pairwise distaces within the whole 150 songs. Now we will randomly select 20 songs from each genre to form the training set. The rest 5 songs are considered to be the testing set. Therefore the total amount of the training set is 120. The amount of the testing set is 30.

We use cross-validation to test the model. In order to make the result valid, we iterate the test for 10 times. In each test the selection of the testing set and the training set is different.

Here is the cross-validation result displayed as a confusion matrix when using mfcc coefficients:

T_avg =

| | Classical | Electronic | Jazz | Punk | Rock | World |
|---|---|---|---|---|---|---|
| **Classical** | 4.6 | 0 | 0.1 | 0 | 0.1 | 0.2 |
| **Electronic** | 0.9 | 1.8 | 0.4 | 0 | 1.9 | 0 |
| **Jazz** | 1.4 | 0.1 | 2.2 | 0.2 | 0.8 | 0.3 |
| **Punk** | 0.1 | 0.2 | 0 | 4.1 | 0.6 | 0 |
| **Rock** | 0.8 | 0.4 | 0.2 | 0.7 | 2.9 | 0 |
| **World** | 1.9 | 0.3 | 0.8 | 0.2 | 0.8 | 1 |

T_std =

| | Classical | Electronic | Jazz | Punk | Rock | World |
|---|---|---|---|---|---|---|
| **Classical** | 0.69921 | 0 | 0.31623 | 0 | 0.31623 | 0.42164 |
| **Electronic** | 0.56765 | 0.91894 | 0.5164 | 0 | 0.99443 | 0 |
| **Jazz** | 0.69921 | 0.31623 | 1.0328 | 0.42164 | 1.0328 | 0.48305 |
| **Punk** | 0.31623 | 0.42164 | 0 | 0.56765 | 0.69921 | 0 |
| **Rock** | 0.78881 | 0.5164 | 0.42164 | 0.82327 | 0.8756 | 0 |
| **World** | 0.99443 | 0.67495 | 0.63246 | 0.42164 | 0.78881 | 0.4714 |

Comment:

The result shows that this genre detection algorithm has 92% success rate for classical, 36% for electronic, 44% for Jazz, 82% for Punk, 58% for Rock, 20% for World. Because the standard deviations are all smaller than 1 except for Jazz (which is 1.0328), we can conclude that the behavior is stable.

The conclusion is that using mfcc coefficients as the music key genre character is good for distinguish Classical and Punk from the others. Meanwhile, this algorithm is “acceptable” for Jazz and Rock. However, it’s almost unusable for World and Electronic.

Here is the cross-validation result displayed as a confusion matrix when using NPCP coefficients:

```
T_avg =
```

| | Classical | Electronic | Jazz | Punk | Rock | World |
|---|---|---|---|---|---|---|
| **Classical** | 1.6 | 0.5 | 0.3 | 0.4 | 1.9 | 0.3 |
| **Electronic** | 0.4 | 2.3 | 0.3 | 1.4 | 0.4 | 0.2 |
| **Jazz** | 0.5 | 0.4 | 2.2 | 1.2 | 0.7 | 0 |
| **Punk** | 0.4 | 0.1 | 0 | 4.1 | 0.4 | 0 |
| **Rock** | 1.2 | 0.5 | 0.5 | 2 | 0.4 | 0.4 |
| **World** | 1.8 | 1.1 | 0.4 | 0.3 | 0.9 | 0.5 |

```
T_std =
```

| | Classical | Electronic | Jazz | Punk | Rock | World |
|---|---|---|---|---|---|---|
| **Classical** | 0.84327 | 0.70711 | 0.48305 | 0.69921 | 0.99443 | 0.48305 |
| **Electronic** | 0.96609 | 1.1595 | 0.67495 | 0.96609 | 0.69921 | 0.42164 |
| **Jazz** | 0.52705 | 0.5164 | 1.3166 | 0.78881 | 0.67495 | 0 |
| **Punk** | 0.5164 | 0.31623 | 0 | 0.56765 | 0.69921 | 0 |
| **Rock** | 1.3166 | 0.70711 | 0.70711 | 1.0541 | 0.69921 | 0.69921 |
| **World** | 1.2293 | 1.1972 | 0.69921 | 0.67495 | 0.8756 | 0.84984 |

Comment:

The success rate of this algorithm is 32% for Classical, 46% for Electronic, 44% for Jazz, 82% for Punk, 8% for Rock, 10% for World. The standard deviation for Electronic and Jazz are relatively large, which means the results are relatively unstable for these two genres.

The result shows NPCP coefficients is worse than mfcc coefficients as genre character. Only Punk has stable 82% success rate. This result is consistent with the result derived from the distance matrix of the whole 150 songs.

## Appendix. MATLAB Source Code

## I.DSP_lab3_main.m (compute distance matrix for 150 songs)

```
%------------I.Import 150 songs as training samples-----------------------%
size_files=150;                         %amount of audio files
fs = zeros(1,size_files);               %music track sampling frequency
length = 30;                            %determine the length uint=seconds
wav = zeros(length*22050,size_files);   %extracted in each song
start = 1;                              %start point (uint = seconds)
for j=1:6                               %keep 6 genres in order
    if j == 1
        file_info=dir('contest/data/classical');
    elseif j == 2
        file_info=dir('contest/data/electronic');
    elseif j == 3
        file_info=dir('contest/data/jazz');
    elseif j == 4
        file_info=dir('contest/data/punk');
    elseif j == 5
        file_info=dir('contest/data/rock');
    elseif j == 6
        file_info=dir('contest/data/world');
    end
    for i=1:25
        [song_temp,fs(1,i+(j-1)*25)]=audioread(file_info(i+2).name);
        [size_temp, dummy]=size(song_temp);
        if size_temp < length*22050        %if the song is shorter than wanted length,
                                           %make the length consistent with
                                           %the wanted length
            wav(1:size_temp,i+(j-1)*25)=...%extract from the tracks
            song_temp(1:size_temp,1);
        else
            wav(:,i+(j-1)*25) = song_temp...
            (start*22050+1:(length+start)*22050,1);
        end
    end
end
%-----------------------------------------------------------------------%

%-------------II. Compute mfcc&merge into 12 channels---------------------%
window=hann(512);                                %hann window
fft_size=512;                                    %fft size is 512
t = zeros(1,36);                                 %merge 40 mfcc coefficients into 12 coefficients
t(1) =1;t(7:8)=5;t(15:18)= 9;
t(2) = 2; t( 9:10) = 6; t(19:23) = 10;
t(3:4) = 3; t(11:12) = 7; t(24:29) = 11;
t(5:6) = 4; t(13:14) = 8; t(30:36) = 12;
mfcc = mfcc1(wav(:,1),fs(1,1),fft_size,window); %determine size of the mel space
mel2 = zeros(12,size(mfcc,2)-1,size_files);

size_mfcc=0;
for i=1:size_files
     mfcc = mfcc1(wav(:,i),fs(1,i),fft_size,window);%call mfcc
     [~, size_mfcc] = size(mfcc(1,:));           %find the number of frames
     mfcc(:,size_mfcc)=[];                       %discard the last unused frame
     size_mfcc=size_mfcc-1;
     for j=1:12
        mel2(j,:,i) = sum(mfcc(t==j,:),1);       %merge mfcc coefficients
     end
end
for i=1:size_files
    for j =1:12
        for k = 1:size_mfcc
            if mel2(j,k,i)==-Inf
                mel2(j,k,i)=0;                   %discard the -Inf mel2 results
            end
        end
    end
end
```

```
mfcc = mel2;
%-------------------------------------------------------------------------%

%----------------------III.compute covariance-----------------------------%
mu = mean(mfcc,2);                              %compute the average
Cov = zeros(12,12,size_files);                  %compute the covariance
for i=1:size_files
    Cov(:,:,i) = cov(mfcc(:,:,i)');
end

iCov = zeros(12,12,size_files);
for i=1:size_files
    iCov(:,:,i) = pinv(Cov(:,:,i));             %compute inverse coveriance
end
%-------------------------------------------------------------------------%

%---------------------IV. compute distance matrix-------------------------%
gam=1;                                          %set scalling parameter gamma
KL = zeros(size_files,size_files);              %create space for KL
d = zeros(size_files,size_files);               %create space for d
for i=1:size_files
    for j=i:size_files                          %compute KL divergence
        KL(i,j) = 0.5*(trace(Cov(:,:,i)*iCov(:,:,j)) + ...
        trace(Cov(:,:,j)*iCov(:,:,i)) + ...
        trace((iCov(:,:,i)+iCov(:,:,j))*(mu(:,:,i)-mu(:,:,j))*...
        (mu(:,:,i)-mu(:,:,j))'));
        d(i,j) = 1 - exp(-gam/(KL(i,j)+eps));   %compute distance
    end
end

%KL = KL + KL';
d = d + d'-diag(diag(d));
%-------------------------------------------------------------------------%

%-------------------V. Pairwise distance histrogram-----------------------%
genre_histrogram=zeros(300,6);                  %create space for histogram
for i=1:6
    genre_histrogram(1:24,i)=d(1,2:25);         %put the distance result computed
    for j=2:24                                  %in the histogram space
        genre_histrogram(sum((25-j+1):24)+1:sum((25-j):24),i)=...
        d(j+25*(i-1),j+1+25*(i-1):25+25*(i-1));
    end
end

figure
subplot(3,2,1)
plot(genre_histrogram(:,1))
title('Classical distance histrogram')

subplot(3,2,2)
plot(genre_histrogram(:,2))
title('Electronic distance histrogram')

subplot(3,2,3)
plot(genre_histrogram(:,3))
title('Jazz distance histrogram')

subplot(3,2,4)
plot(genre_histrogram(:,4))
title('Punk distance histrogram')

subplot(3,2,5)
plot(genre_histrogram(:,5))
title('Rock distance histrogram')

subplot(3,2,6)
plot(genre_histrogram(:,6))
title('World distance histrogram')
%-------------------------------------------------------------------------%
```

```
%--------------VI. compute average distance matrix------------------------%
D = zeros(6,6);
for n=1:6
    for m=n:6
        for i = 1:25                        %compute average distance matrix
            D(n,m) = D(n,m) + sum(d(i+(n-1)*25,25*(m-1)+1:25*(m-1)+25));
        end
    end
end
D= D/(25^2);
D = D+D'--diag(diag(D));

%compute average difference of average distance matrix(D_mean)
D_sum=zeros(1,6);
for i=1:6
    D_sum(1,i)=sum(D(i,i)-D(1,:));
end

D_mean = mean(D_sum)/(6*min(min(D_sum)));
%-----------------------------------------------------------------------%
```

## II. DSP_lab3_classifier.m (cross-validation of the classifier)

```
%------------I.Import 150 songs as training samples-----------------------%
size_files=150;%amount of audio files
fs = zeros(1,size_files);%music track sampling frequency
length = 30;%determine the length uint=seconds
wav = zeros(length*22050,size_files);  %extracted in each song
start = 1;                          %start point (uint = seconds)

for j=1:6                           %keep 6 genres in order
    if j == 1
        file_info=dir('contest/data/classical');
    elseif j == 2
        file_info=dir('contest/data/electronic');
    elseif j == 3
        file_info=dir('contest/data/jazz');
    elseif j == 4
        file_info=dir('contest/data/punk');
    elseif j == 5
        file_info=dir('contest/data/rock');
    elseif j == 6
        file_info=dir('contest/data/world');
    end
    for i=1:25
        [song_temp,fs(1,i+(j-1)*25)]=audioread(file_info(i+2).name);
        [size_temp, dummy]=size(song_temp);
        if size_temp < length*22050
            wav(1:size_temp,i+(j-1)*25) = song_temp(1:size_temp,1);
        else
            wav(:,i+(j-1)*25) = song_temp(start*22050+1:(length+start)*22050,1);
        end
    end
end
%-----------------------------------------------------------------------%

confusion = zeros(6,6,10);
for iter=1:10
%-----------------II.Define test set and training set---------------------%
[size_wav,~] = size(wav);
num_test = 5;
test_set=zeros(size_wav,num_test*6);
train_set=zeros(size_wav,150-num_test*6);
set=zeros(25,1);
for i=1:6
    set=randperm(25);               %genrate random order of 25 numbers
    for j=1:5
        test_set(:,j+(i-1)*5)=...
        wav(:,set(j)+(i-1)*25);% take the first 5 numbers as the test set index
    end
    for j=6:25
```

```matlab
        train_set(:,j-5+(i-1)*20)=...
        wav(:,set(j)+(i-1)*25);% take the rest 20 numbers as the test set index
    end
end
[~,size_train] = size(train_set);
[~,size_test] = size(test_set);

%-------------III. Compute mfcc&merge into 12 channels--------------------%
window=hann(512);%hann window
fft_size=512;    %fftsize
t = zeros(1,36); %merge 40 mfcc into 12
t(1) =1;t(7:8)=5;t(15:18)= 9;
t(2) = 2; t( 9:10) = 6; t(19:23) = 10;
t(3:4) = 3; t(11:12) = 7; t(24:29) = 11;
t(5:6) = 4; t(13:14) = 8; t(30:36) = 12;
train_mfcc = mfcc1(train_set(:,1),fs(1,1),fft_size,window);
train_mel2 = zeros(12,size(train_mfcc,2)-1,size_train);
test_mfcc = mfcc1(test_set(:,1),fs(1,1),fft_size,window);
test_mel2 = zeros(12,size(test_mfcc,2)-1,size_test);

size_mfcc=0;
for i=1:size_train
     train_mfcc =...                              %call mfcc
     mfcc1(train_set(:,i),fs(1,i),fft_size,window);
     [~, size_mfcc] = size(train_mfcc(1,:));     %find the number of frames
     train_mfcc(:,size_mfcc)=[];                 %discard the last unused frame
     size_mfcc=size_mfcc-1;
     for j=1:12
        train_mel2(j,:,i) =...                   %merge mfcc coefficients
        sum(train_mfcc(t==j,:),1);
     end
end
for i=1:size_train
    for j =1:12
        for k = 1:size_mfcc
            if train_mel2(j,k,i)==-Inf      %discard -Inf data
                train_mel2(j,k,i)=0;
            end
        end
    end
end
train_mfcc = train_mel2;

size_mfcc=0;
for i=1:size_test
     test_mfcc = mfcc1(test_set(:,i),fs(1,i),fft_size,window);
     [~, size_mfcc] = size(test_mfcc(1,:));     %find the number of frames
     test_mfcc(:,size_mfcc)=[];                     %discard the last unused frame
     size_mfcc=size_mfcc-1;
     for j=1:12
        test_mel2(j,:,i) = sum(test_mfcc(t==j,:),1);
     end
end
for i=1:size_test
    for j =1:12
        for k = 1:size_mfcc
            if test_mel2(j,k,i)==-Inf
                test_mel2(j,k,i)=0;
            end
        end
    end
end
test_mfcc = test_mel2;
%-------------------------------------------------------------------------%

%--------------------------III.2.Compute chroma---------------------------%
fftsize_chroma = 1024;
window_chroma = kaiser(1024);
train_chm = chroma(wav(:,1),fs(1,1),fftsize_chroma,window_chroma);
train_chm = zeros(12,size(train_chm,2),size_train);
```

```matlab
for i=1:size_train
     train_chm(:,:,i) = chroma(train_set(:,i),fs(1,i),fftsize_chroma,window_chroma);
end

test_chm = chroma(wav(:,1),fs(1,1),fftsize_chroma,window_chroma);
test_chm = zeros(12,size(test_chm,2),size_test);

for i=1:size_test
     test_chm(:,:,i) = chroma(test_set(:,i),fs(1,i),fftsize_chroma,window_chroma);
end
%--------------------------------------------------------------------------%

%------------------------IV.compute mu and covariance----------------------%
%for mfcc
train_mu = mean(train_mfcc,2);
train_Cov = zeros(12,12,size_train);
for i=1:size_train
    train_Cov(:,:,i) = cov(train_mfcc(:,:,i)');
end

test_mu = mean(test_mfcc,2);
test_Cov = zeros(12,12,size_test);
for i=1:size_test
    test_Cov(:,:,i) = cov(test_mfcc(:,:,i)');
end
%for mfcc end

% %%for chroma
% train_mu = mean(train_chm,2);
% train_Cov = zeros(12,12,size_train);
% for i=1:size_train
%     train_Cov(:,:,i) = cov(train_chm(:,:,i)');
% end
%
% test_mu = mean(test_chm,2);
% test_Cov = zeros(12,12,size_test);
% for i=1:size_test
%     test_Cov(:,:,i) = cov(test_chm(:,:,i)');
% end
% %%for chroma end

train_iCov = zeros(12,12,size_train);
for i=1:size_train
    train_iCov(:,:,i) = pinv(train_Cov(:,:,i));
end

test_iCov = zeros(12,12,size_test);
for i=1:size_test
    test_iCov(:,:,i) = pinv(test_Cov(:,:,i));
end
%--------------------------------------------------------------------------%

%------------------------IV. compute distance------------------------------%
gam=0.9;
KL = zeros(size_test,size_train);
d = zeros(size_test,size_train);

for i=1:size_test
    for j=1:size_train
        KL(i,j) = 0.5*(trace(test_Cov(:,:,i)*train_iCov(:,:,j)) + ...
        trace(train_Cov(:,:,j)*test_iCov(:,:,i)) + ...
        trace((test_iCov(:,:,i)+train_iCov(:,:,j))*(test_mu(:,:,i)-train_mu(:,:,j))*...
        (test_mu(:,:,i)-train_mu(:,:,j))'));

%          KL(i,j) = 0.5*(trace(test_Cov(:,:,i)*train_iCov(:,:,j))+...
%          (train_mu(:,:,j)-test_mu(:,:,i))'*train_iCov(:,:,j)*...
%          (train_mu(:,:,j)-test_mu(:,:,i))+log(det(train_Cov(:,:,j)/...
%          det(test_Cov(:,:,i)))));
```

```
        %d(i,j) = 1 - exp(-gam/(KL(i,j)+eps));
        d(i,j) = exp(-gam*(KL(i,j)+eps));
    end
end

%---------------------------V. compute KNN------------------------------%
vote = zeros(30,5);
for j=1:30
[vote_val,vote_idx]=sort(d(j,:),'descend');
    for i=1:5                          %assign genre to vote indexes
        if vote_idx(i)>=1&&vote_idx(i)<=20
            vote(j,i) = 1;
        elseif vote_idx(i)>=21&&vote_idx(i)<=40
            vote(j,i) = 2;
        elseif vote_idx(i)>=41&&vote_idx(i)<=60
            vote(j,i) = 3;
        elseif vote_idx(i)>=61&&vote_idx(i)<=80
            vote(j,i) = 4;
        elseif vote_idx(i)>=81&&vote_idx(i)<=100
            vote(j,i) = 5;
        elseif vote_idx(i)>=101&&vote_idx(i)<=120
            vote(j,i) = 6;
        end
    end
end

vote_result=zeros(30,1);
for i=1:30                             %find the max 5 vote result
    [~,vote_result_idx]=max(histc(vote(i,:),[1:6]));
    vote_result(i)=vote_result_idx;
end


for i=1:6                              % find the amount of major vote and
                                       %put the result in confusion matrix
    confusion(i,:,iter)=histc(vote_result(1+(i-1)*5:5+(i-1)*5),[1:6])';
end
end

confusion_avg=zeros(6,6);
confusion_std=zeros(6,6);
for i=1:6
    for j=1:6                          %compute average and std of confusion matrix
        confusion_avg(i,j)=mean(confusion(i,j,:));
        confusion_std(i,j)=std(confusion(i,j,:));
    end
end


Classical=confusion_avg(:,1);         %put the result in the table
Electronic=confusion_avg(:,2);
Jazz=confusion_avg(:,3);
Punk=confusion_avg(:,4);
Rock=confusion_avg(:,5);
World=confusion_avg(:,6);
True_genre = {'Classical','Electronic','Jazz','Punk','Rock','World'};
T_avg = table(Classical,Electronic,Jazz,Punk,Rock,World,'RowNames',True_genre)

Classical=confusion_std(:,1);
Electronic=confusion_std(:,2);
Jazz=confusion_std(:,3);
Punk=confusion_std(:,4);
Rock=confusion_std(:,5);
World=confusion_std(:,6);
True_genre = {'Classical','Electronic','Jazz','Punk','Rock','World'};
T_std = table(Classical,Electronic,Jazz,Punk,Rock,World,'RowNames',True_genre)
```

## III. mfcc.m

```matlab
%8. --------------transfer the frames into frequency domain-----------------%
    Y=zeros(N,num_frame);                    %creaate space for fft result
    for n=1:num_frame
        Y(:,n)=fft(xn(:,n).*window);         %compute fft
    end

    %8. -------------------discard the negative frequencies then square it--------------------%
    K = N/2+1;                               %size we need for positive frequency
    Xn=zeros(K,num_frame);                   %creaate space for fft true result
    for n=1:num_frame
        Xn(:,n)=abs(Y(1:K,n));               %get the positive frequency part
    end

    Xn2=(Xn).^2;                             %calculate |Xn|^2

    %9.a.Spectral centroid and spread
    frame_sum=sum(Xn);                       %sum each frame
    Xn_hat=Xn./frame_sum;                    %calculate Xn hat. Xn hat = |Xn|/|frame sum|
    spread = std(Xn_hat);                    %compute spread
    spectral_centroid=zeros(1,num_frame);    %creaate space for spectral centroid
    for n=1:num_frame                        %loop across every data point in fft
        for k=1:K
            spectral_centroid(1,n)=spectral_centroid(1,n)+k.*Xn_hat(k,n);%sum up Hn_hat*k
        end
    end
    spectral_centroid = spectral_centroid/K;%normalize the result

    %9.b.Spectral flatness
    spectral_flatness=zeros(1,num_frame);    %creaate space for spectral flatness
    for n=1:num_frame
        spectral_flatness(1,n)=geomean(Xn(:,n))/mean(Xn(:,n));%flatness=geometric mean/mean
    end

    %9.c.Spectral flux
    spectral_flux=zeros(1,num_frame);        %creaate space for spectral flatness
    for n = 2:num_frame
        spectral_flux(1,n) = sum((Xn_hat(:,n) - Xn_hat(:,n-1)).^2);%compute spectral flux
    end

    %11. Mel filter banks generation
    nbanks = 40; %% Number of Mel frequency bands
    % linear frequencies
    linFrq = 20:fs/2;
    % mel frequencies
    melFrq = log ( 1 + linFrq/700) *1127.01048;
    % equispaced mel indices
    melIdx = linspace(1,max(melFrq),nbanks+2);
    % From mel index to linear frequency
    melIdx2Frq = zeros (1,nbanks+2);
    % melIdx2Frq (p) = \Omega_p
    for i=1:nbanks+2
    [val melIdx2Frq(i)] = min(abs(melFrq - melIdx(i)));%compute mel index equivalent frequency
    end
    melEq=linFrq(melIdx2Frq);                           %compute corresponding mel frequency
    melL = melEq(1:nbanks);                             %the left point frequency
    melR = melEq(3:nbanks+2);                           %the right point frequency
    melC = melEq(2:nbanks+1);                           %the center point frequency
    fbank = zeros(nbanks,fftsize/2+1);                  %space for fbank
    Hp = 2./(melR-melL);                                %filter hat
    linFrq = linspace(0,fs/2,fftsize/2+1);              %linear frequency space
    for i=1:nbanks
        fbank(i,:) = ...                                %compute fbank
        (linFrq > melL(i) & linFrq <= melC(i)).* ...
        Hp(i).*(linFrq-melL(i))/(melC(i)-melL(i)) + ...
        (linFrq > melC(i) & linFrq < melR(i)).* ...
        Hp(i).*(melR(i)-linFrq)/(melR(i)-melC(i));
    end

    %12. mfcc coefficients
    mfcc=zeros(40,num_frame);                            %space for mfcc coefficients
```

```
    for n=1:num_frame                                         %loop frames
        for p=1:40                                            %loop filter banks
            for k=1:K                                         %loop in one frame
                mfcc(p,n)=mfcc(p,n)+abs(fbank(p,k)*Xn(k,n)).^2;%compute mfcc coefficients
            end
        end
    end
```

## IV. chroma.m

```
function [ NPCP ] = chroma( wav, Fs, fftSize,window )
%This function compute the NPCP(normalized pitch class profile)of the music
%waveform. It also generates a chromagram of it.

%-------------------cut wav sequence into frames--------------------------%
N = fftSize;                              %fftsize=frame size
[length_wav dummy]=size(wav);             %get the track length
num_frame = ceil(length_wav/(N/2));       %find the number of frames
xn=zeros(N,num_frame);                    %create space for the frame structure
wav_cat=vertcat(wav,zeros(N*...           %add zeros to the tail of the frame
(num_frame/2)-length_wav,1));
for n=1:num_frame-1                       %loop number of frames
    for i=1:N                             %loop in one frame
        xn(i,n) = wav_cat(i+(n-1)*(N)); %put the track into the frame structure
    end
end
xn(:,num_frame)=[];                       %discard the last unused frame
num_frame=num_frame-1;
%---------------transfer the frames into frequency domain------------------%
Y=zeros(N,num_frame);                     %creaate space for fft result
for n=1:num_frame
    Y(:,n)=fft(xn(:,n).*window);          %compute fft
end

%--------------------discard the negative frequencies---------------------%
K = N/2+1;                                %size we need for positive frequency
Xn=abs(Y(1:K,:));                         %get the positive frequency part

%----------------------------I. peak detection---------------------------%

%1. find the maxium number of peaks to decide
%   the size of row of fk(peak frequency) matrix and sm(semitone) matrix
npeaks=zeros(1,num_frame);                %create space for number of peaks
for n=1:num_frame                         %loop frames
    [peak]= findpeaks(Xn(:,n));           %find peaks
    [npeaks(1,n) dummy]=size(peak);       %set the size of the matrix by max npeaks
end

%---------II.Assignment of the peak frequencies to semitones--------------%

%initialization
fk=zeros(max(npeaks),num_frame);     %local maxima frequencies
fk_mag=zeros(max(npeaks),num_frame);%peak frequency magnitude
%sm=zeros(max(npeaks),num_frame);     %semi-tones corresponding to fk
%c=zeros(max(npeaks),num_frame);     %map semitones to notes
nyquist_freq=Fs/2;                   %nyquist frequency
f0=27.5;                             %lowest frequency
hopSize = fftSize/2;                 %frame size

%a.find peak frequency, compute semitone and notes
for n=1:num_frame                    %loop frames
    [fk_mag(1:npeaks(n),n) fk(...    %find peak frequency
    1:npeaks(n),n)]=findpeaks(Xn(:,n));
end

%-------------------------noise canceling---------------------------------%
%In each frame, all the peaks whose magnitude is smaller than-60dB of the
%maximum peak magnitude are considered as noise. They will be wiped out.
max_fk_mag = max(fk_mag);            %find max peak in each frame
```

```
for n=1:num_frame                    %loop frames
    if fk_mag(1:npeaks(n),n) <= max_fk_mag(n)/1000
        fk_mag(1:npeaks(n),n) = 0;  %delete the small ripples
    end
end
%--------------------------------------------------------------------------%

sm = round(12*log2((fk./hopSize.*nyquist_freq)./f0));%compute semitone
c = mod(sm,12);                      %compute notes

%------------------------III.Pitch Class Profile---------------------------%

%a. raised cosine weighting function
r=12*log2((fk-1)/hopSize*nyquist_freq/f0)-sm;
w=cos(pi*abs(r)/2).^2;
if (r<=-1 | r>=1),
    w = 0;
end

%b. compute PCP(Pitch Class Profile)
% c matrix contains the information of notes.If certain frequency
% corresponds to a note i, it means the power of that frequency should sum
% into row i of PCP matrix.
PCP=zeros(12,num_frame);             %creeate space for PCP
for i=1:num_frame
    for j=1:npeaks(i)
        for k=0:11
            if c(j,i)==k             %compute PCP
                PCP(k+1,i)=PCP(k+1,i)+w(j,i)*(fk_mag(j,i)^2);
            end
        end
    end
end

%------------------------IV.Normalize PCP ---------------------------------%
%divide all PCPs by the maximum PCP in the note space
NPCP = PCP./max(max(PCP));           %compute NPCP
%--------------------------------------------------------------------------%
```