Digital Signal Processing Lab 3

By Zhi Jie Huang

1. Introduction:

This is the third and final lab of the content based music information retrieval. In this lab, we will utilize the MFCC coefficient, rhythm, low level temporal, spectral features, and Chroma that we calculated from the first two lab to automatically detect the genre of a track.

2. Implementation of the distance matrix

There are 6 genres in this project, classical, electronics, jazz/blues, metal/punk, pop/rock and world. Each genre has 25 songs, it makes the total of 150 songs. The following 5 picture vary gamma from 10 to 150 of the MFCC to maximizes the separation between the different genres. Since my distance matrix is a little different from the lab write up, therefore, my gamma value varies from 10 to 150 instead of 0.1 to1. I think gamma equal to 100 looks the best for the distance matrix. The idea of the distance matrix is to compute the distance between every two track of the 150 songs. As you can see below in the MFCC distance 100 graph, the first 25 songs are classical, as you can see it has a small distance within the first 25 songs, that's why you can see a lot of red in the top left area. And it's really blue to another other songs, therefore, you can clearly see the first region is classical. The second genres is electronics, the index 26~50 are electronics, as you can see in the MFCC distance100 graph, it does have a small distance between electronics; however, it also have a very small distance between metal/punk, pop/rock and electronics. Due to the similar instruments played in that music. But you can still see electronics has a relatively large between world, jazz/blues and classical. Metal/punk and pop/rock music are similar type, therefore I have a hard time distinguish between the two. World doesn't really have a small distance to anything, which makes sense, because world music is very rich, it included various instrument.

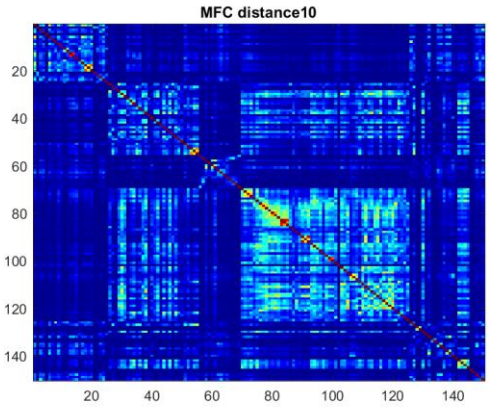

Fig 1: MFCC distance matrix

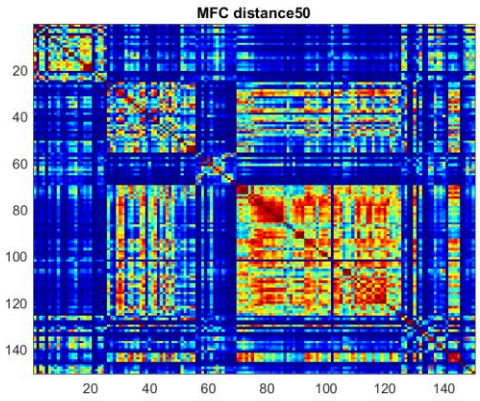

Fig 2: MFCC distance matrix

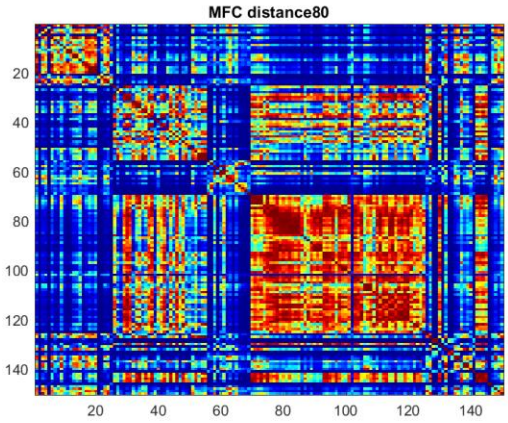

Fig 3: MFCC distance matrix with gamma 80

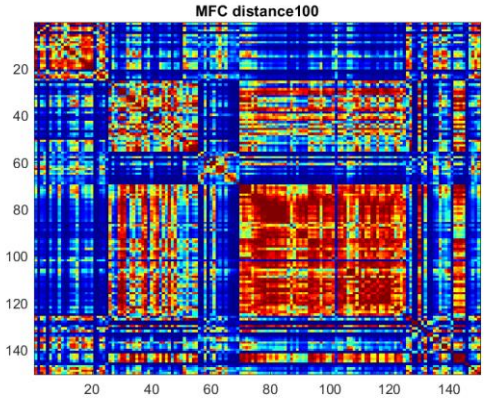

Fig 4: MFCC distance matrix with gamma 100

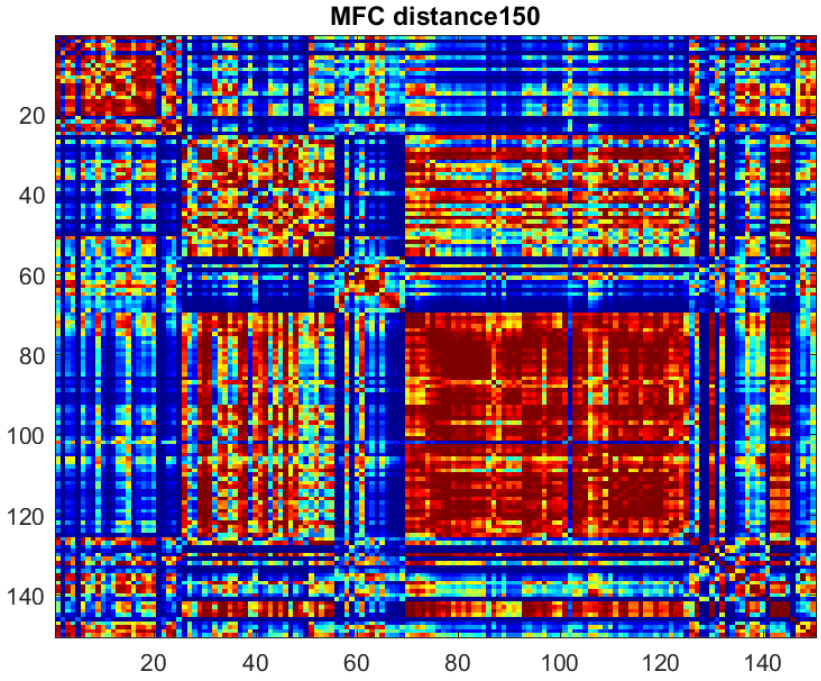

Fig 5: MFCC distance matrix with gamma 150

For the Chroma coefficient, I must do the same procedure for the distance matrix. The result are shown below. The distance matrix for Chroma clearly have more noise than the MFCC distance matrix, it is due to we destroy all the melody and bring everything into the first octave. As you can see in the result, it is very hard to use Chroma to distinguish one genres from another.

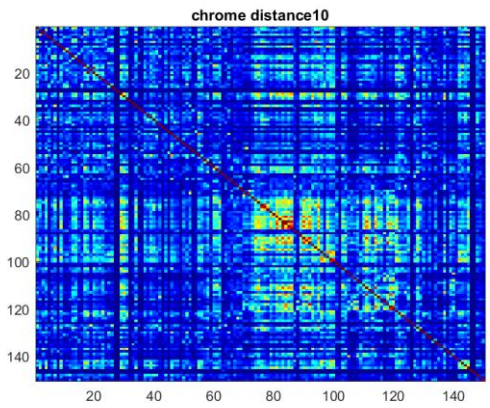

Fig 6: Chrome distance gamma 10

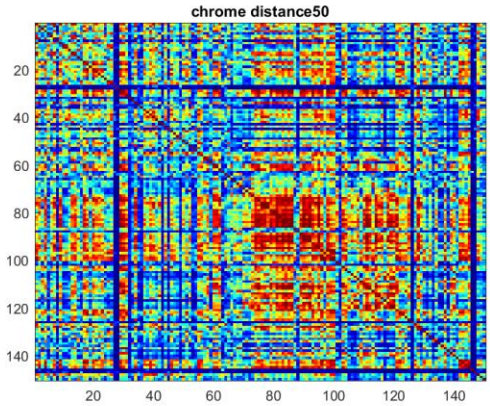

Fig 7: chrome distance gamma 50

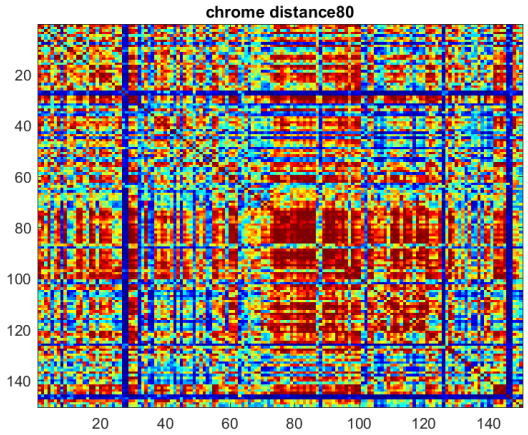

Fig 8: Chrome distance gamma 80

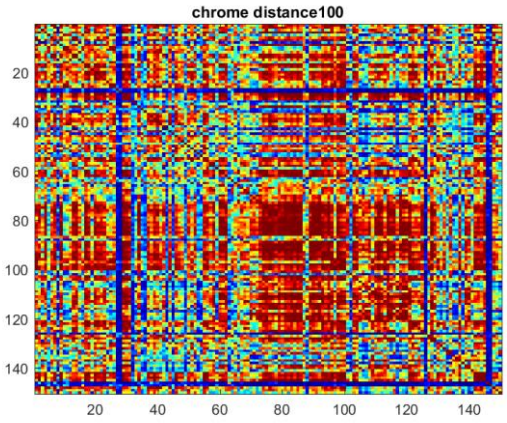

Fig 9: Chrome distance gamma 100

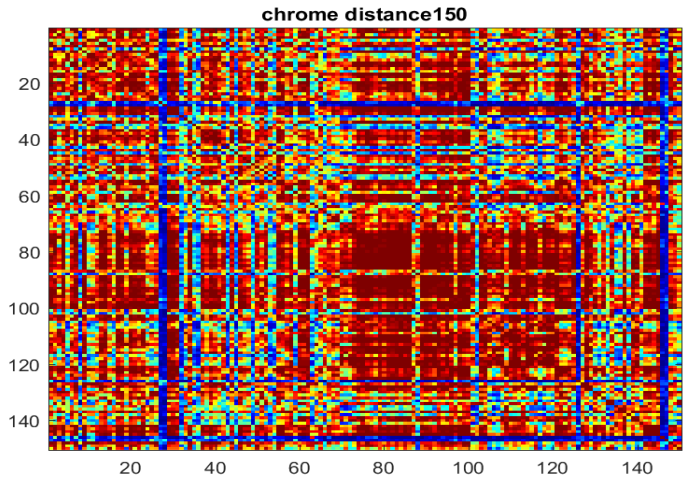

Fig 10:chrome distance gamma 150

3. Histograms:

Histogram is a graphical representation of the distribution of numerical data, it counts how many values fall into each interval. In this part of the lab, we are trying to analyze the distance value we have from one genres versus other genres. As you can see from figure 11 shown below, Classical, electronic, jazz, metal, and rock all have a lot of songs that are relatively close to each other, therefore using the K-nearest neighbor will really help identify which genres is which genres. However, Since world has such a wide spread of distance, and mostly not even close to each other, it makes it extremely hard to identify world music. Figure 11 and figure 12 are almost identical.

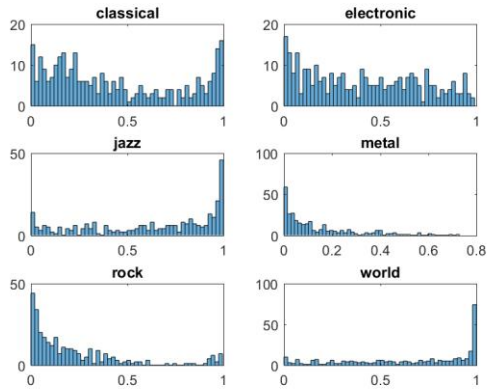

Fig 11:MFCC histogram

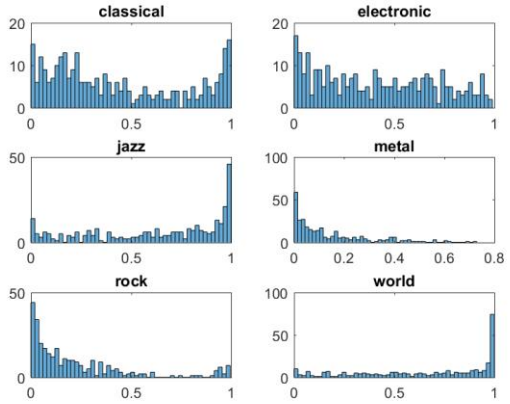

Fig 12: Chroma histogram

4. Average Matrix:

Since my distance function is a little different from the write up, therefore, my gamma value is in a different range. After I calculated the distance matrix between each 2 tracks of the 150 songs, I am going to average the value of each 25x25 matrix to get an average between genres. The below figures illustrate the separation between genres, as you can see at gamma equal to 100, it has the best separation between genres. When gamma equal to 150, there is barely any differences between gamma equal to 100 and gamma equal to 150.

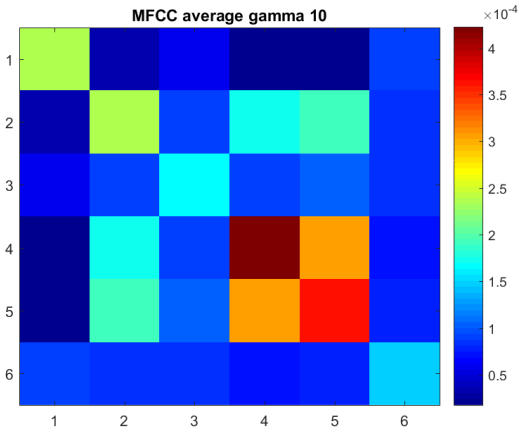

Fig 13: MFCC average with gamma 10

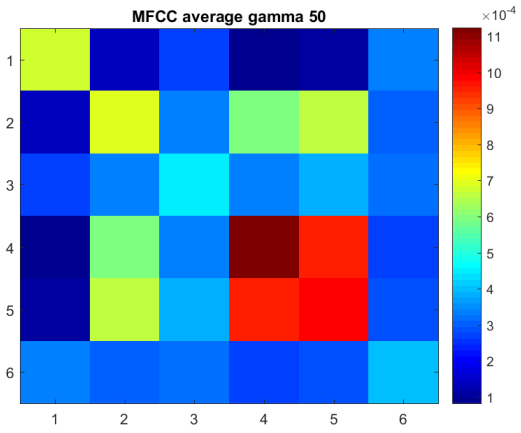

Fig 14:MFCC average with gamma 50

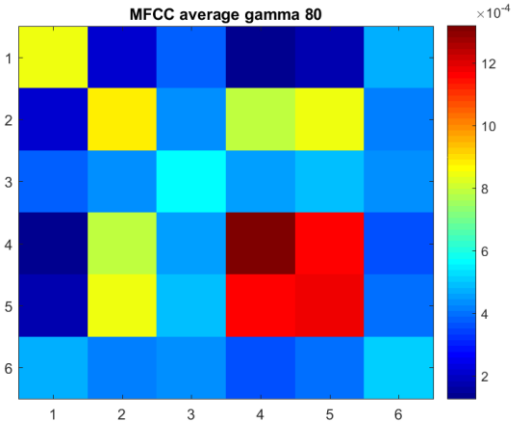

Fig 15:MFCC with gamma 80

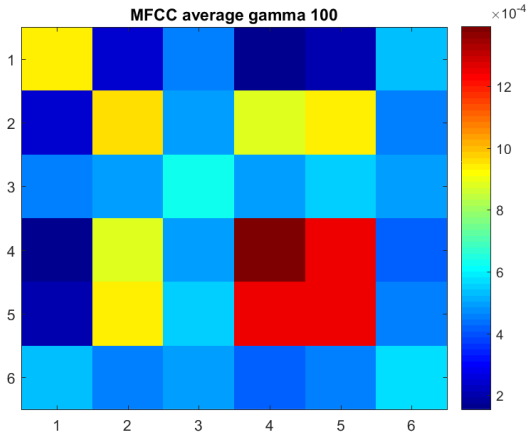

Fig 16:MFCC with gamma 100

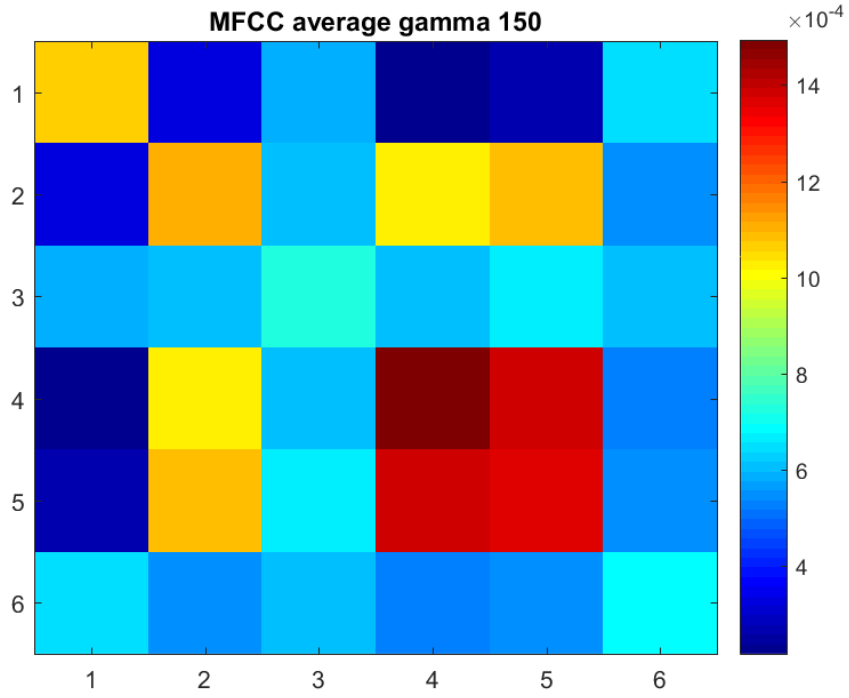

Fig 17: MFCC with gamma 150

The chroma average distance matrix doesn't have a really good distinction between genres.

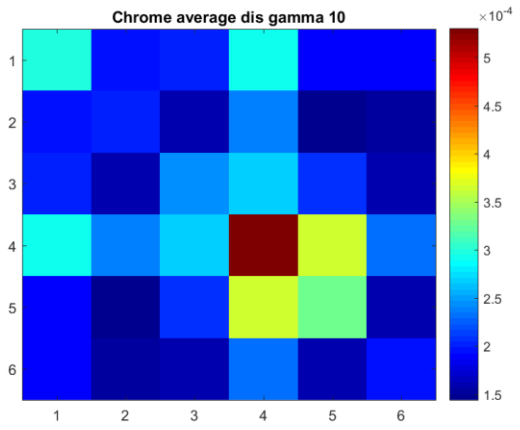

Fig 18: Chroma average matrix with gamma 10

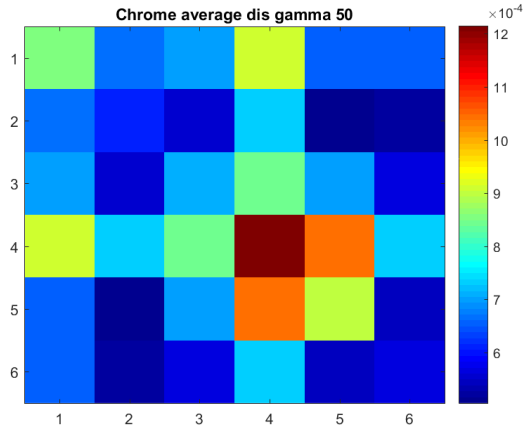

Fig 19: Chroma average matrix with gamma 50

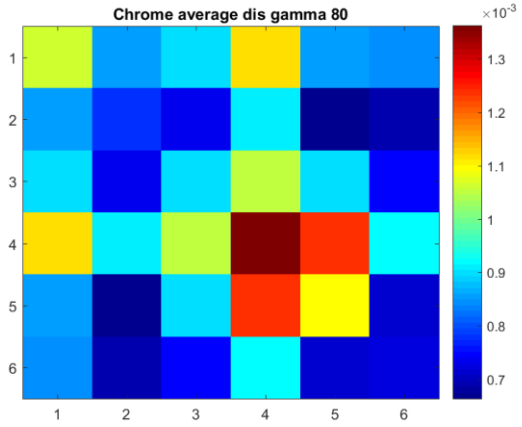

Fig 20: Chroma average matrix with gamma 80

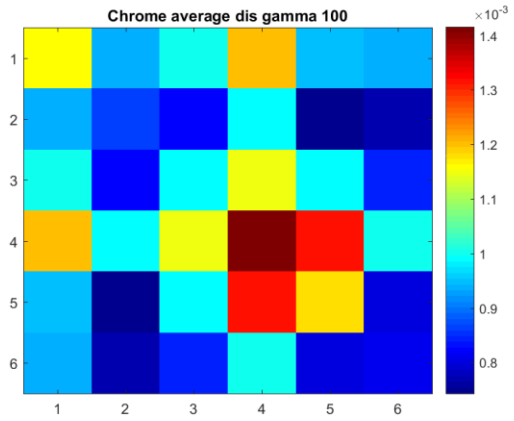

Fig 21: Chroma average matrix with gamma 100

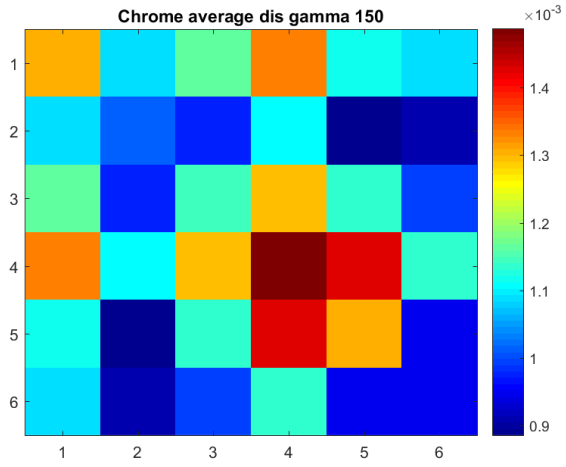

Fig 22: Chroma average matrix with gamma 150

5. Impact of the length of the excerpt on the classification:

By just looking at the average distance matrix, you can tell grabbing the longer help improve the separation of the distance between each genre. Let's focus first block for classical, you can see in the first two figure 23 and 24, block (1,1) are green. In those two cases, we are grabbing 30 and 60 seconds. Just increase the number of seconds to 120, it improves our average distance matrix significantly. Since my distance matrix is 1 subtract the exponential, my distance matrix will be trying to maximize the distance instead of minimizing the distance. You can see, increases the number of seconds does help improve the average distance matrix. However, there must be a balance between the accuracy of the classification and time. By grabbing more seconds from the song, you are increasing your computation time. And it looks like 120 seconds is the threshold that increasing time after that would not significantly improve your performance.

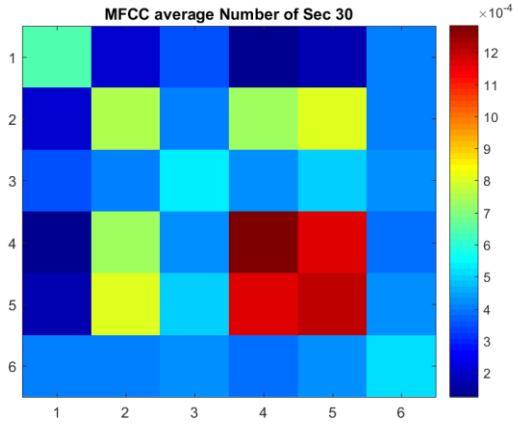

Fig 23: MFCC average matrix 30 sec

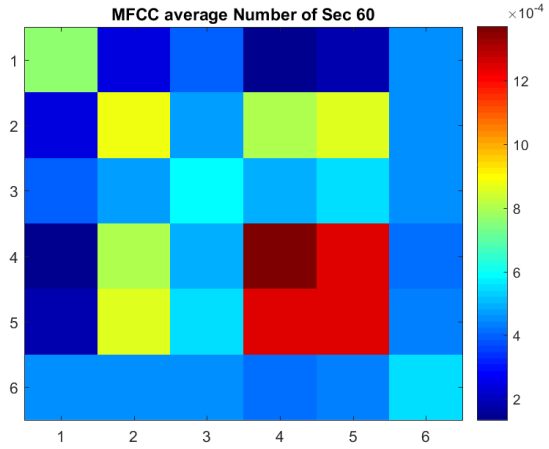

Fig 24: MFCC average matrix 60 sec

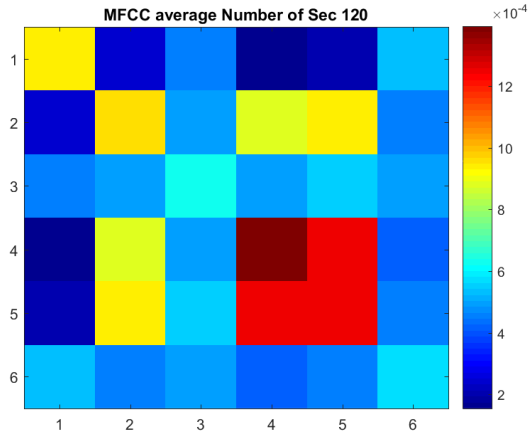

Fig 25: MFCC average matrix 120 sec

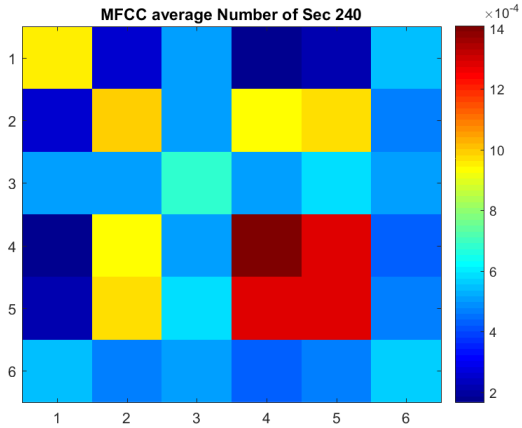

Fig 26: MFCC average matrix 240 sec

From looking at the distance matrix through different length of the song, you can see that the 120 seconds of MFCC has more red than MFCC 30 seconds.

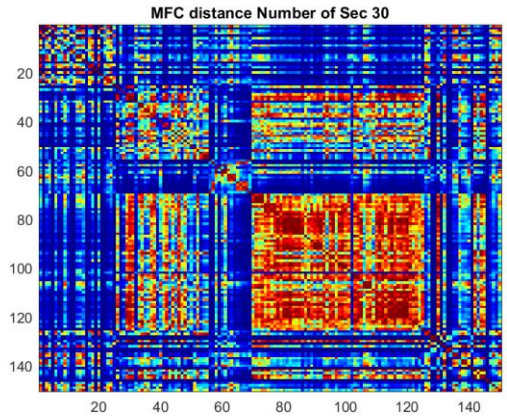

Fig 27: MFCC distance matrix with 30 seconds

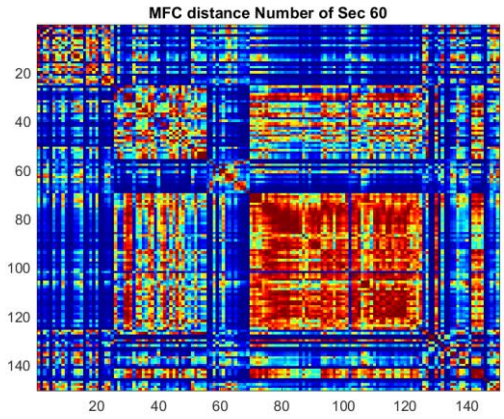

Fig 28: MFCC distance matrix with 60 seconds

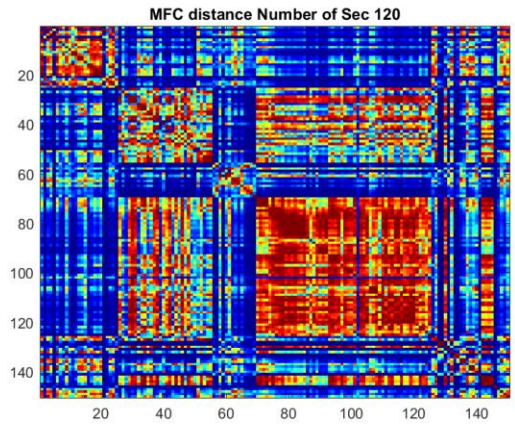

Fig 29: MFCC distance matrix with 120 seconds

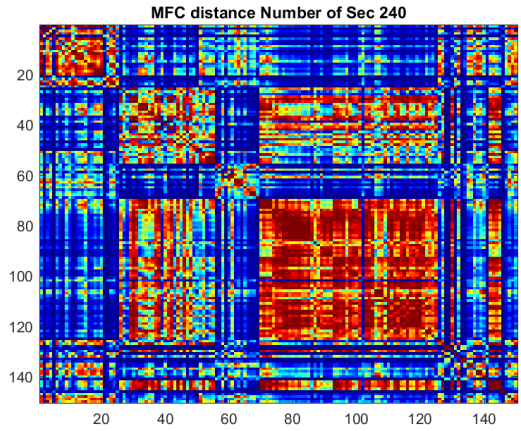

Fig 30: MFCC distance matrix with 240 seconds

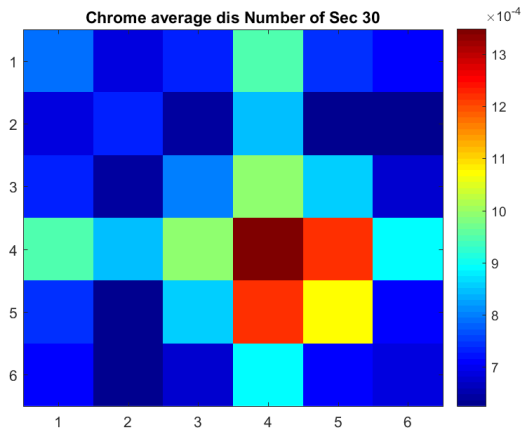

Fig 31: Chroma average distance 30 seconds

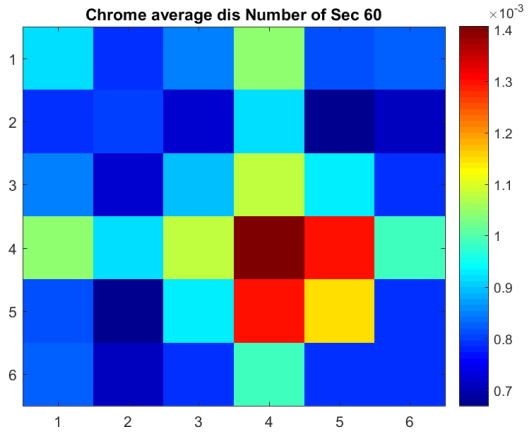

Fig 32: Chroma average distance 60 seconds

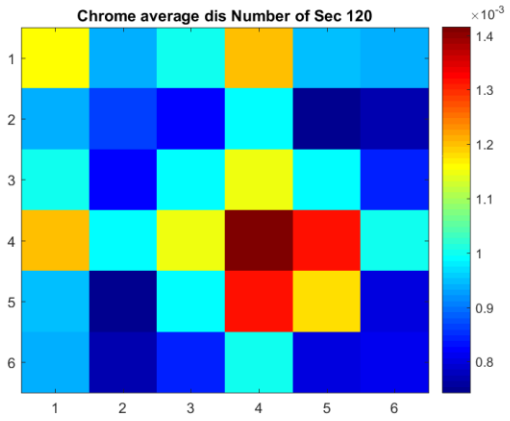

Fig 33: Chroma average distance 120 seconds

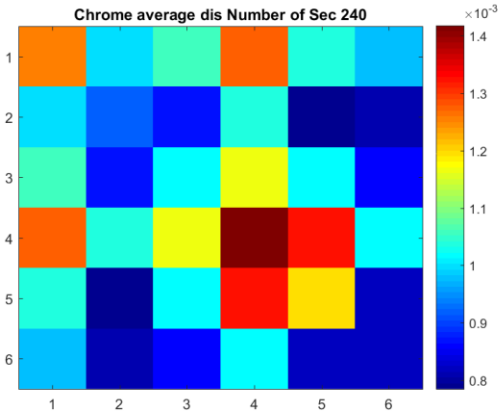

Fig 34: Chroma average distance 240 seconds

6. K – nearest neighbor confusion matrix

MFCC with K nearest neighbor:

This is the accuracy without running cross validation, from the distance matrix, with the 5 nearest neighbor algorithm, my distribution of songs are shown below.

	Classical	Electronic	Jazz	Metal/punk	Pop/rock	world
Classical	24	0	0	0	0	1
Electronic	0	20	0	0	3	2
Jazz	0	0	23	2	0	0
Metal/punk	0	1	0	20	3	1
Pop/rock	1	1	0	5	18	0
world	6	1	1	1	1	15

Accuracy:

The accuracy is [0.96,0.80,0.92,0.80,0.72,0.60].

This is pretty high accuracy, however, this is without any partition of dataset, this is not a true way to test how well does your algorithm works.

	Classical	Electronic	Jazz	Metal/punk	Pop/rock	world
Classical	0.96	0	0	0	0	0.04
Electronic	0	0.80	0	0	0.12	0.08
Jazz	0	0	0.92	0.08	0	0
Metal/punk	0	0.04	0	0.80	0.12	0.04
Pop/rock	0.04	0.04	0	0.20	0.72	0
world	0.24	0.04	0.04	0.04	0.04	0.60

Chroma with K nearest neighbor:

From the histogram, it is difficult to tell which algorithm is better, however, from the result of the K nearest neighbor, we can absolutely tell that even without any testing set, the MFCC is much better than the Chroma.

	Classical	Electronic	Jazz	Metal/punk	Pop/rock	world
Classical	22	2	1	0	0	0
Electronic	7	13	2	2	1	0
Jazz	1	4	14	5	1	0
Metal/punk	0	1	0	23	1	0
Pop/rock	2	0	1	10	11	1
world	7	2	0	5	2	9

Accuracy:

The accuracy is [0.88,0.52,0.56,0.92,0.44,0.36]

	Classical	Electronic	Jazz	Metal/punk	Pop/rock	world
Classical	0.88	0.08	0.04	0	0	0
Electronic	0.28	0.52	0.08	0.08	0.04	0
Jazz	0.04	0.16	0.56	0.20	0.04	0
Metal/punk	0	0.04	0	0.92	0.04	0
Pop/rock	0.08	0	0.04	0.40	0.44	0.04
world	0.28	0.08	0	0.20	0.08	0.36

7. Cross-Validation

MFCC Average matrix:

	Classical	Electronic	Jazz	Metal/punk	Pop/rock	world
Classical	4.80	0	0	0	0	0.20
Electronic	0.20	3	0	0.10	1	0.70
Jazz	0.60	0.40	3.30	0.40	0.10	0.20
Metal/punk	0	0.40	0	4.30	0.30	0
Pop/rock	0.10	0.70	0.20	1.10	2.90	0
world	2.50	0.40	0.40	0.40	0	1.30

Average hit rate is:

The hit rate is about what I expected, because from the distance matrix and average distance matrix, you can see that World is the hardest to classify, because it has some much variation, it is very hard to classify world music. Classical is one of the easiest to classify, because it's rhythm, tone, MFCC and NPCP coefficient is distinct.

Classical	Electronic	Jazz	Metal/punk	Pop/rock	world
0.96	0.60	0.66	0.86	0.58	0.26

Standard deviation:

The standard deviation tells us the how much variation in the distance matrix between each songs in each genres. The smaller the standard deviation is, the easier to classify the song.

Classical	Electronic	Jazz	Metal/punk	Pop/rock	world
0.4216	0.9428	0.8233	0.483	0.9944	0.6749

8. Performance with 12 MFCC vs 12 NPCP

The performance of MFCC versus NPCP is shown below, MFCC is better at almost all categories but Metal/punk. MFCC has a 86% accuracy Metal/punk, whereas NPCP has 96% in the Metal/punk. Since we are randomizing our test set, this make senses, it could be just a bad test set for the MFCC. But in general, MFCC has a better performance than NPCP.

MFCC performance:

Average hit rate is:

Classical	Electronic	Jazz	Metal/punk	Pop/rock	world
0.96	0.60	0.66	0.86	0.58	0.26

NPCP performance:

Average hit rate is:

Classical	Electronic	Jazz	Metal/punk	Pop/rock	world
0.94	0.46	0.60	0.96	0.48	0.22

9. Improve the classifier:

Using support vector machine to maximize the distance between each genres in order to give a better separation of each genres. The classification error is really small, from the two times that I ran it, it is about 0.03 – 0.04.

Appendix:

Main.m

```
clear all;close all;clc;
tic
pathName = 'C:\Users\JayHuang\OneDrive\Documents\CU Boulder\2017 Spring\ECEN
4532\lab 3';
fileList = dir(fullfile(pathName,'*.wav'));
fileName = dir('*.wav');
NumSong = size(fileName,1);
mfcc1 = cell(150,1);
Chrome1 = cell(150,1);
fftSize = 512;
w = hann(fftSize);
fftSizeC = 2048;
gamma = 100;
NumSec = 120;

%---------------For the effect of the different length of the song
% using gamma as 100
parfor i = 1:NumSong
    %--------Extract the length---------%
    [song,fs] = audioread(fileName(i).name);
    ExtractedSong = SongExtract(song,NumSec,fs);
     mfccCoe = mfcc(ExtractedSong,fs,fftSize,w);
    % demochrom('track201-classical');
    mfcc1{i} = Cov_Mu(mfccCoe);
    Chrome1{i} = mychroma(ExtractedSong',fs,fftSizeC);
end

MfccD = zeros(150,150);
chromeD = zeros(150,150);

for i = 1:NumSong
    for j = 1:i
        %---assignment 1: distance matrix--------------
        MfccD(i,j) = distance(mfcc1{i},mfcc1{j},gamma);
        MfccD(j,i) = MfccD(i,j);
        chromeD(i,j) = distance(Chrome1{i},Chrome1{j},gamma);
        chromeD(j,i) = chromeD(i,j);
    end
end
genreList = {'classical','electronic','jazz','metal','rock','world'};
%
% 3. For each genre, compute the histogram composed of the 300 pairwise
distances within that
% genre. Plot the six histograms on the same figure. Comment on the figure.
% index = 1;
% figure
% for i = 1:25:150
%     subplot(3,2,index)
%     his1 = triu(MfccD(i:i+24,i:i+24),1);
%     his2 = nonzeros(his1);
%     histogram(his2,50);
%     title(genreList(index));
```

```
%     index = index + 1;
% end
% saveas(gcf,'Historgram_for_Mfcc_distance.png');
%
% figure
% index2 = 1;
% for j = 1:25:150
%     subplot(3,2,index2);
%     his1 = triu(chromeD(j:j+24,j:j+24),1);
%     his2 = nonzeros(his1);
%     histogram(his2,50);
%     title(genreList(index2));
%     index2 = index2 + 1;
% end
% saveas(gcf,'Histogram_for_Chroma_distance.png');
% %----------------------end of problem 3----------------

DMFCCavg = zeros(6,6);
DChromeavg = zeros(6,6);
% ---------Calculate the average matrix---------

% for i = 1:25:150
%     for j = 1:25:150
%         i_ = fix(i/25);
%         j_ = fix(j/25);
%         DMFCCavg(i_+1,j_+1) = mean(mean(MfccD(i:i+24,j:j+24)))/625;
%         DChromeavg(i_+1,j_+1) = mean(mean(chromeD(i:i+24,j:j+24)))/625;
%     end
% end

%gcf get current figure
% figure
% colormap jet
% imagesc(DChromeavg);
% title(strcat('Chrome average dis Number of Sec',{' '},int2str(NumSec)));
% colorbar
% saveas(gcf,strcat('Chrome_average_dis NumSec',int2str(NumSec),'.png'));
%
% figure
% colormap jet
% imagesc(DMFCCavg);
% title(strcat('MFCC average Number of Sec',{' '},int2str(NumSec)));
% colorbar
% saveas(gcf,strcat('MFCC_average NumSec',int2str(NumSec),'.png'));
%
% figure
% colormap jet
% colorbar
% imagesc(MfccD)
% title(strcat('MFC distance Number of Sec',{' '},int2str(NumSec)));
% saveas(gcf,strcat('MFC_distance NumSec',int2str(NumSec),'.png'));
%
%
%
% figure
% colormap jet
```

```
% colorbar
% imagesc(chromeD);
% title(strcat('chrome distance Number of Sec',{' '},int2str(NumSec)));
% saveas(gcf,strcat('chromedistance',int2str(NumSec),'.png'));

% Assignment
% 8. Implement a classifier based on the following ingredients, as explained
above,
% • computation of the 12 mfcc coefficients, or 12 Normalized Pitch Class
Profile
% • modified Kullback-Leibler distance d defined by (6)
% • genre = majority vote among the 5 nearest neighbors
ConfusionMatrix = zeros(6,6);
    for i = 1:6 %rows are the true genres
   % columns are the genres classified by my algorithm
     ConfusionMatrix(i,:) = Classifier(chromeD,i);
    end
    Accuracy = ConfusionMatrix/25;
%------------End of problem 8-----------------

% 9. Using cross validation, as explained in section 5.2, evaluate your
classification algorithm. You
% will compute the mean and standard deviation for all the entries in the
confusion matrices.
%Cross validation of the algorithm:


p = zeros(6,25);
total = zeros(6,6);
% create cell array for average and standard deviation
RecordMatrix = zeros(10,6);
for n = 1:10 % start of repeat
%     -----------------Version 1--------------------%
    % pass in the 5-fold cross validation
    TestSong = zeros(6,5);
    TrainSong = zeros(6,20);
%     for k = 1:5
        for g = 1:6 %form a set of test songs and training songs
            p(g,:) = randperm(25)+25*(g-1);
            TestSong(g,:) = p(g,1:5);
%             TrainSong(g,:) = p(g,6:25);
        end
        %Since we have the distance song already,set all the
        %test song value to zeros
        i = 1;
        indices = zeros(1,30);
        %put all indices into a single row of matrix
        TestingSet = sort(reshape(TestSong,[1,30]));
        MfccD(TestingSet(1,1:30),TestingSet(1,1:30)) = 0;
        ConfusionCrossMatrix = zeros(6,6);
          for gen = 1:6
              ConfusionCrossMatrix(gen,:) =
ClassifierCross(chromeD,p(gen,1:5));
          end
          total = ConfusionCrossMatrix + total;
          RecordMatrix(n,:) = reshape(diag(ConfusionCrossMatrix),1,6);
```

```
%           RecordMatrix{n} = ConfusionCrossMatrix;
end

% average = total/10;

%---------------------End of version 1---------------------------

averageMatirx = total/10;

average = zeros(1,6);
StandMatrix = zeros(1,6);
for a = 1:6
    average = mean(RecordMatrix)/5;
    StandMatrix = std(RecordMatrix);
end

%11. Improve the classifier using support vector machines (SVM).
%create a empty cell array
for i = 1:25
    classname{i} = 'Classical';
    classname{i+25} = 'Electronic';
    classname{i+50} = 'Jazz';
    classname{i+75} = 'metal';
    classname{i+100} = 'rock';
    classname{i+125} = 'World';
end

Mdl = fitcecoc(MfccD,classname);
%%
% Display the coding design matrix.
Mdl.ClassNames
CodingMat = Mdl.CodingMatrix
%%
% A one-versus-one coding design on three classes yields three binary
% learners.  Columns of |CodingMat| correspond to learners and rows
% correspond to classes.  The class order corresponds to the order
% in |Mdl.ClassNames|. For example, |CodingMat(:,1)| is |[1; -1; 0]|, and
% indicates that the software trains the first SVM binary learner using
% all observations classified as |'setosa'| and |'versicolor'|.  Since
% |'setosa'| corresponds to |1|, it is the positive class, and since
% |'versicolor'| corresponds to |-1|, it is the negative class.
%%
% You can access each binary learner using cell indexing and dot notation.
for i = 1:6
    Mdl.BinaryLearners{i};                % The first binary learner
    Mdl.BinaryLearners{i}.SupportVectors % Support vector indices
end
%%
% Compute the in-sample classification error.
isLoss = resubLoss(Mdl)
%%
% The classification error is small, but the classifier might have been
% overfit.  You can cross validate the classifier using |crossval|.

toc
```

distance.m

```
function [d] = distance(mfcc1,mfcc2,gam)
    mu1 = mean(mfcc1,2);
    co1 = cov(mfcc1');
    mu2 = mean(mfcc2,2);
    co2 = cov(mfcc2');
    K = 12;
    iCo1 = pinv(co1);
    iCo2 = pinv(co2);
   %Fransico's formula:
   KL = 0.5*trace(co1*iCo2) + trace(co2*iCo1) + trace(((mu1-
mu2)'*(iCo1+iCo2)*(mu1-mu2)))-K;
   %----end of fransico's formula-----------%
   % this is a symmetric distance matrix
   %trace function add up the sum of the digonal matrix
    d = 1 - exp(-gam/(KL + eps));
%     d = exp(-gam*(KL + eps));
end
```

songExtract.m

```
function [output] = SongExtract(song,NumSec,fs)
SongLength = length(song);
GrabLength = NumSec*fs;
Mid = floor(SongLength/2);
    if SongLength <= GrabLength
        output = song;
    else
        output = song(Mid-GrabLength/2:Mid+GrabLength/2-1);
    end
end
```

oct2hz.m

```
function [hz] = oct2hz(octs,A440)

if nargin < 2;
    A440 = 440;
end

hz = (A440/16).*(2.^octs);

return;
```

mychroma.m

```
function [C] = mychroma (signal,samplingRate,lengthFFT)
%
%
%  Computes the 12 x nframes chroma representation
%
%  The computation procedes in two steps:
%    1) an estimate of the instantaneous frequency is computed using the
derivative of the
%    phase. This idea relies on the concept of the analytical signal.
%    2) the energy associated with the instantaneous frequency is folded back
into the octave
%    defined by A0.


A0    = 27.5;

frameSize   = lengthFFT/2;
frameSize2 = lengthFFT/4;

nchr = 12;

s = length(signal);
nFrames = 1 + floor((s - (frameSize))/(frameSize2));

T = (frameSize)/samplingRate;

win = 0.5*(1-cos([0:((frameSize)-1)]/(frameSize)*2*pi));
dwin = -pi / T * sin([0:((frameSize)-1)]/(frameSize)*2*pi);

norm = 2/sum(win);

iF = zeros(1 + frameSize, nFrames);
S = zeros(1 + frameSize, nFrames);

nmw1 = floor(frameSize2);
nmw2 = frameSize - nmw1;

ww = 2*pi*[0:(lengthFFT-1)]*samplingRate/lengthFFT;

%
% computation of the instantaneous frequency, using the analytical expression
of the derivative of the phase

for ifrm = 1:nFrames

    u = signal((ifrm-1)*(frameSize2) + [1:(frameSize)]);

    wu = win.*u;
    du = dwin.*u;

    wu = [zeros(1,nmw1),wu,zeros(1,nmw2)];
    du = [zeros(1,nmw1),du,zeros(1,nmw2)];
```

```
    t1 = fft(fftshift(du));
    t2 = fft(fftshift(wu));

    S(:,ifrm) = t2(1:(1 + frameSize))'*norm;

    t = t1 + j*(ww.*t2);
    a = real(t2);
    b = imag(t2);
    da = real(t);
    db = imag(t);

    instf = (1/(2*pi))*(a.*db - b.*da)./((a.*a + b.*b)+(abs(t2)==0));

    iF(:,ifrm) = instf(1:(1 + frameSize))';
end;


%
% now we find the local maxima of the instantaneous frequency

f_ctr = 1000;
f_sd = 1;

f_ctr_log = log2(f_ctr/A0);
fminl   = oct2hz(hz2oct(f_ctr)-2*f_sd);
fminu  = oct2hz(hz2oct(f_ctr)-f_sd);
fmaxl  = oct2hz(hz2oct(f_ctr)+f_sd);
fmaxu = oct2hz(hz2oct(f_ctr)+2*f_sd);

minbin  = round(fminl * (lengthFFT/samplingRate) );
maxbin = round(fmaxu * (lengthFFT/samplingRate) );

%
% compute the second order derivative of the instantaneous frequency to
detect the peaks

ddif    = [iF(2:maxbin, :);iF(maxbin,:)] - [iF(1,:);iF(1:(maxbin-1),:)];

% clean a bit

dgood = abs(ddif) < .75*samplingRate/lengthFFT;

dgood = dgood .* ([dgood(2:maxbin,:);dgood(maxbin,:)] >  0 |
[dgood(1,:);dgood(1:(maxbin-1),:)] > 0);

p  = zeros(size(dgood));
m = zeros(size(dgood));

%
%  try to fit a second order polynomial locally

for t = 1:size(iF,2)
```

```
    ds = dgood(:,t)';
    lds = length(ds);

    st = find(([0,ds(1:(lds-1))]==0) & (ds > 0));
    en = find((ds > 0) & ([ds(2:lds),0] == 0));
    npks = length(st);
    frqs = zeros(1,npks);
    mags = zeros(1,npks);

    for i = 1:length(st)
        bump = abs(S(st(i):en(i),t));
        frqs(i) = (bump'*iF(st(i):en(i),t))/(sum(bump)+(sum(bump)==0));
        mags(i) = sum(bump);

        if frqs(i) > fmaxu
            mags(i) = 0;
            frqs(i) = 0;
        elseif frqs(i) > fmaxl
            mags(i) = mags(i) * max(0, (fmaxu - frqs(i))/(fmaxu-fmaxl));
        end

        if frqs(i) < fminl
            mags(i) = 0;
            frqs(i) = 0;
        elseif frqs(i) < fminu
            mags(i) = mags(i) * (frqs(i) - fminl)/(fminu-fminl);
        end
        if frqs(i) < 0
            mags(i) = 0;
            frqs(i) = 0;
        end
    end

    bin = round((st+en)/2);
    p(bin,t) = frqs;
    m(bin,t) = mags;
end

ncols = size (p,2);

%
% clean up

Pocts = hz2oct(p+(p==0));
Pocts(p(:)==0) = 0;

nzp = find(p(:)>0);

[hn,hx]  = hist(nchr*Pocts(nzp)-round(nchr*Pocts(nzp)),100);
centsoff = hx(find(hn == max(hn)));

Pocts(nzp) = Pocts(nzp) - centsoff(1)/nchr;

PoctsQ = Pocts;
```

```
PoctsQ(nzp) = round(nchr*Pocts(nzp))/nchr;

Pmapc = round(nchr*(PoctsQ - floor(PoctsQ)));
Pmapc(p(:) == 0) = -1;
Pmapc(Pmapc(:) == nchr) = 0;

C = zeros(nchr,ncols);
for t = 1:ncols;
    C(:,t)=(repmat([0:(nchr-
1)]',1,size(Pmapc,1))==repmat(Pmapc(:,t)',nchr,1))*m(:,t);
end

return;
```

mfcc.m

```
function [mfcc] = mfcc(wav, fs, fftSize, window)
%
%
%
% USAGE
%
%   [mfcc] = mfcc(wav, fs, fftSize,window)
%
% INPUT
%      wav: vector of wav samples
%      fs : sampling frequency
%      fftSize: size of fft
%      window: a window of size fftSize
%
% OUTPUT
%   mfcc : matrix of 40 mel coefficients x nFrames



%
%
%   hard wired parameters

hopSize = fftSize/2;
nBanks = 40;

%
%
%    minimum and maximum frequencies for the analysis
fMin = 20;
fMax = fs/2;

%_____________________________________________________________
%
%     PART 1 : construction of the filters in the frequency domain
%_____________________________________________________________

% generate the linear frequency scale of equally spaced frequencies from 0 to
fs/2.

linearFreq = linspace(0,fs/2,hopSize+1);

fRange = fMin:fMax;

% map the linear frequency scale of equally spaced frequencies from 0 to fs/2
to an unequally spaced
%  mel scale.

melRange = log(1+fRange/700)*1127.01048;

% The goal of the next coming lines is to resample the mel scale to create
uniformly spaced mel
% frequency bins, and then map this equally spaced mel scale to the linear
scale.
```

```
%
% divide the mel frequency range in equal bins

melEqui = linspace (1,max(melRange),nBanks+2);

fIndex = zeros(nBanks+2,1);

% for each mel frequency on the equally spaces grid, find the closest
frequency on the unequally
% spaced mel scale

for i=1:nBanks+2,
    [dummy fIndex(i)] = min(abs(melRange - melEqui(i)));
end

%  now, we have the indices of the unequally mel scale that match the equally
mel grid. These
%  indices match the linear frequency, so we can assign a linear frequency
for each equally spaced
%  mel frequency

fEquiMel = fRange(fIndex);


%  design of the hat filters: we build two arrays that correspond to the
center, left and right ends
%  of each triangle.

fLeft   = fEquiMel(1:nBanks);
fCentre = fEquiMel(2:nBanks+1);
fRight  = fEquiMel(3:nBanks+2);

% clip filters that leak beyond the Nyquist frequency

[dummy, tmp.idx] = max(find(fCentre <= fs/2));
nBanks = min(tmp.idx,nBanks);

% this array contains the frequency response of the nBanks hat filters.

freqResponse = zeros(nBanks,fftSize/2+1);

hatHeight = 2./(fRight-fLeft);

% for each filter, we build the left and right edge of the hat.

for i=1:nBanks,
    freqResponse(i,:) = ...
        (linearFreq > fLeft(i) & linearFreq <= fCentre(i)).* ...
        hatHeight(i).*(linearFreq-fLeft(i))/(fCentre(i)-fLeft(i)) + ...
        (linearFreq > fCentre(i) & linearFreq < fRight(i)).* ...
        hatHeight(i).*(fRight(i)-linearFreq)/(fRight(i)-fCentre(i));
end
```

```
%
% plot a pretty figure of the frequency response of the filters.

% figure; set(gca,'fontsize',14)
% semilogx(linearFreq,freqResponse');
% axis([0 fRight(nBanks) 0 max(freqResponse(:))])
% title('Filterbank');

%_____________________________________________________________
%
%     PART 2 : processing of the audio vector. The processing is performed in
the Fourier domain.
%
%_____________________________________________________________

%
% estimate the number of frames that we will process

nFrames = floor (length(wav)/hopSize);
if (mod(nFrames,2) == 0)
    nFrames = nFrames -1;
end

%
% allocate the mel coefficients

mel = zeros(nBanks, nFrames);

%
% and the power spectrum

poweSpect = zeros(fftSize/2+1, nFrames);

%
% normalize the window

window = 2*window./sum(window);

chunkIndx = 1:fftSize;

for i=1:nFrames-1,
    X = abs(fft(wav(chunkIndx).*window,fftSize));

    poweSpect(:,i) = X(1:end/2+1);
    mel(:,i) = freqResponse * poweSpect(:,i);

    chunkIndx = chunkIndx + hopSize;
end

mfcc = mel;

% mfcc = 10*log10(mel);
```

```
%
% % plot the power spectrum
%
% figure; subplot(2,1,1); set(gca,'fontsize',14)
%
% poweSpect = 10*log10(poweSpect);
% imagesc(poweSpect);
%
% % plot the mel coefficients
% set(gca,'ydir','normal','xtick',[]); title('Spectrum')
% colormap('jet'); hold on; colorbar
%
% subplot(2,1,2); set(gca,'fontsize',14)
%
% tt = [1:size(mfcc,2)]*fftSize/(2*fs);
% imagesc(tt,[1:40],mfcc);
%
% set(gca,'ydir','normal'); caxis(max(caxis)+[-60 0])
% colormap('jet');hold on;colorbar
%
% title('MFCC Representation')

return;




end
```

FindNMinimum.m

```
function [ index] = FindNMinimum( input,n )
    SortMatrix = sort(input);
    len = size(input,2);
%     MinValue = SortMatrix(len-4:len);
    MinValue = SortMatrix(len-4:len);
    index = zeros(1,n);
    for i = 1:n
        index(1,i) = find(input == MinValue(1,i));
    end
end
```

demochrom.m

```
function demochrom (filename)

BasePath = ['Z:/ECEN4632/FinalProject/'];

% BasePath = ['/Users/francois/class/4532/tracks/'];

[x,fs] = audioread([BasePath '/' filename '.wav']);


x = x';

fftlen= 2048;

[C] = mychroma (x, fs, fftlen);

figure;

tt = [1:size(C,2)]*fftlen/4/fs;

imagesc(tt,[1:12],20*log10(C+eps));

axis xy
set (gca, 'YTick', [1:12]);
set (gca, 'YTickLabel', {'A',
'A#','B','C','C#','D','D#','E','F','F#','G','G#'});
set (gca, 'XTick', [0:60:ceil(tt(end))]);
caxis(max(caxis)+[-60 0])

title(['Chroma of ' filename]);
colormap('jet');hold on;colorbar;

return;
```

Con_Mu.m

```
function demochrom (filename)

BasePath = ['Z:/ECEN4632/FinalProject/'];

% BasePath = ['/Users/francois/class/4532/tracks/'];

[x,fs] = audioread([BasePath '/' filename '.wav']);


x = x';

fftlen= 2048;

[C] = mychroma (x, fs, fftlen);

figure;

tt = [1:size(C,2)]*fftlen/4/fs;

imagesc(tt,[1:12],20*log10(C+eps));

axis xy
set (gca, 'YTick', [1:12]);
set (gca, 'YTickLabel', {'A',
'A#','B','C','C#','D','D#','E','F','F#','G','G#'});
set (gca, 'XTick', [0:60:ceil(tt(end))]);
caxis(max(caxis)+[-60 0])

title(['Chroma of ' filename]);
colormap('jet');hold on;colorbar;

return;
```

ClassifierCross.m

```
function [ output ] = ClassifierCross(distance,distIndex)
%Find the 5 nearest neighbor
n = 5;
index = zeros(25,n);
temp = zeros(1,25);
% row = rows*25-24;
output = zeros(1,6);
% col = columns*25-24;
loopi = 1;
% for i = row:row+24
%     %looping through the 25 songs
%     index(loopi,:) = FindNMinimum(distance(i,:),5);
%     loopi = loopi + 1;
% end

for i = 1:5
    %loop through the 5 different songs
    index(loopi,:) = FindNMinimum(distance(distIndex(1,i),:),5);
     loopi = loopi + 1;
end

    index = floor(index/25)+1;

    for j = 1:5
        temp(1,j) = mode(index(j,:));
    end

    for k = 1:5
        switch(temp(1,k))
            case 1
                output(1,1) = output(1,1) + 1;
            case 2
                output(1,2) = output(1,2) + 1;
            case 3
                output(1,3) = output(1,3) + 1;
            case 4
                output(1,4) = output(1,4) + 1;
            case 5
                output(1,5) = output(1,5) + 1;
            case 6
                output(1,6) = output(1,6) + 1;
        end
    end
end
```

Classifer.m

```
function [ output ] = Classifier(distance,rows)
%Find the 5 nearest neighbor
n = 5;
index = zeros(25,n);
temp = zeros(1,25);
row = rows*25-24;
output = zeros(1,6);
% col = columns*25-24;
loopi = 1;
for i = row:row+24
    %looping through the 25 songs
    index(loopi,:) = FindNMinimum(distance(i,:),5);
    loopi = loopi + 1;
end
    index = floor(index/25)+1;

    for j = 1:25
        temp(1,j) = mode(index(j,:));
    end

    for k = 1:25
        switch(temp(1,k))
            case 1
                output(1,1) = output(1,1) + 1;
            case 2
                output(1,2) = output(1,2) + 1;
            case 3
                output(1,3) = output(1,3) + 1;
            case 4
                output(1,4) = output(1,4) + 1;
            case 5
                output(1,5) = output(1,5) + 1;
            case 6
                output(1,6) = output(1,6) + 1;
        end
    end
end
```