Lab 3

Michael Eller

March 9, 2016

Contents

1 Introduction 2

2 Distances Between Tracks 2

3 Conclusion 10

A Figures 11
A.1 Distance Matrix . . . 11
A.1.1 Chroma . . . 11
A.1.2 MFCC . . . 13
A.2 Genre Distribution Matrix . . . 16
A.2.1 Chroma . . . 16
A.2.2 MFCC . . . 18

B Code 20

1 Introduction

This is the third and final lab on digital signal processing methods for content-based music information retrieval. In this lab, we will be comparing various *distances* between audio tracks and using those distances along with various classification methods to automatically detect the genre of a given track. The distances we are using are based on the MFCC coefficients, the rhythm, and the chroma.

The absolute end goal of this lab is to develop an algorithm that can upload a song and return a label with a certain probability. For example, if the song "Recitative" from the album "Dark Intervals" by Keith Jarrett is selected, one would expect that the algorithm classifies it as "classical" or "jazz" with a high probability.

2 Distances Between Tracks

The goal of the classification is to assign a genre to a song. In this lab, we are forcusing on simple techniques, such as the k-nearest neighbors.

After condensing our previous MFCC from 40 bins to 12 bins, we can more accurately compare the effectiveness of the MFCC features vs the Chroma features we developed in Lab 2.

Given either matrix (MFCC or Chroma), we are able to calculate the Kullback-Leibler divergence:

$$KL(G^s, G^{s'}) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_{s'}^{-1}\Sigma_s) + (\mu_{s'} - \mu_s)^T\Sigma_{s'}^{-1}(\mu_{s'} - \mu_s) - K + \log\left(\frac{\det\Sigma_{s'}}{\det\Sigma_s}\right)\right) \tag{1}$$

We may note that this distance is very similar to the Mahalanobis distance discussed in class.

Finally, the KL distance is rescaled using an exponential kernel, and we define the distance between the tracks s and s' as

$$d(s, s') = \exp\left(-\gamma KL(G^s, G^{s'})\right) \tag{2}$$

Assignment

1. Implement the computation of the the distance D given by Equation 2. Your function should be able to use the 12 merged MFCC coefficients or the Normalized Pitch Class Profile.
2. Compute the matrix of pairwise distance

   $$D(s, s'), s, s' = 1, \ldots, 150 \tag{3}$$

   Display the distance matrix as an image, and discuss its structure
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.
4. Compute the 6×6 average distance matrix between the genres, defined by

   $$\bar{D}(i, j) = \frac{1}{25^2}\sum_{s\in\text{genre } i,\ s'\in\text{genre } j} d(s, s'), \quad i, j = 1, \ldots, 6 \tag{4}$$

5. Experiment with different values of γ, and find a value of γ that maximizes the seperation between the different genres, as defined by the 6×6 average distance matrix $\bar{D}$.
6. Compare the MFCC and the Normalized Pitch Class Profile in terms of the 6×6 average distance matrix $\bar{D}$.

As seen below in Listing 2, the Kullback-Leibler distance is a fairly straightforward computation. At one point, before I fixed accompanying computations, there were somehow distances generated outside of the 0

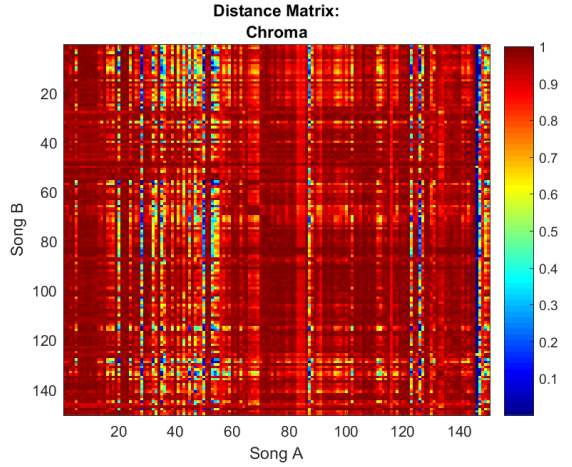
to 1 range that the function should generate. As a sanity check, I clamped the final values between 0 and 1, and the extra 3 computations have not seemed to affect the performance of my function.

Listing 1: kullbackDistance.m

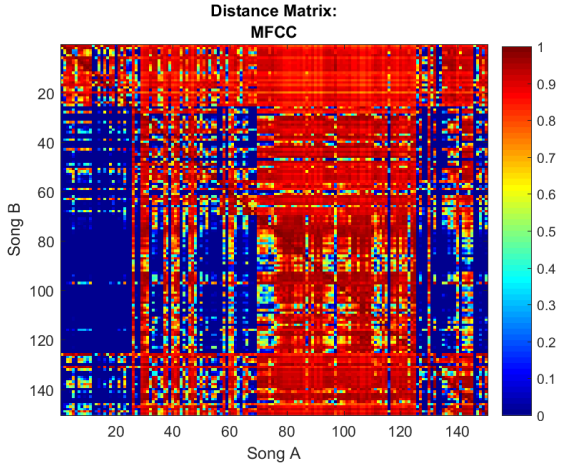
```
function [ d ] = kullbackDistance( P,Q)
%kullbackDistance Computes the distance between two tracks
%   First computes the Kullback-Leibler divergence
%   Then the KL distance is rescaled using an exponential kernel

mu1 = mean(P,2,'omitnan');
Cov1=cov(P','omitrows');
mu2 = mean(Q,2,'omitnan');
Cov2=cov(Q','omitrows');

K=12;

KL=(0.5*trace(pinv(Cov2)*Cov1+(pinv(Cov1)*Cov2)))-K+...
    (0.5*(mu1-mu2)')*(pinv(Cov2)+pinv(Cov1))*(mu1-mu2);

gamma=0.002; % gamma should be very very small
d=exp(-1*gamma*KL);% Apply weighting kernel
end
```

After the basic function was created, we would next compute the distance between all 150 songs with each other. To save time, we could have just computed the upper triangle of our matrix and then mirrored it over the diagonal (resulting in only 11,175 calculations), but our distance function is unfortunately not symmetrical. This forces us to compute all $150 \times 150 = 22,500$ combinations. We are, however, able to slightly reduce the number of calculations by noting that any distance between itself will always be 1. This saves us 150 calculations.

With the same γ in both distance matrix calculations, the MFCC and the Chroma give very different results. The MFCC matrix seems like a more viable candidate because it is computationally faster (to compute the 150 mfcc matrices) and it is less saturated.



(a) Distance Matrix: Chroma



(b) Distance Matrix: MFCC

Listing 2: distanceMatrix.m

```
function [ distMatrix ] = distanceMatrix( xList,whichPlot,timeLength)
%distanceMatrix Computes the distance between every song
%   Uses a precomputed list of the mfcc/normPCP matrices to compute the
%   distance matrix

```

```
numTracks=length(xList);
distMatrix=zeros(numTracks,numTracks);

h=waitbar(0,'Computing Distance Matrix...');
for i=1:numTracks
    for j=1:numTracks
        if(i==j)
            distMatrix(i,j)=1;
        else
            distMatrix(i,j)=kullbackDistance(xList{i},xList{j});
        end
    end
    waitbar(i/numTracks)
end

close(h); % close waitbar
if(nargin>=2) %plot graph and save away
    h=figure;
    switch whichPlot
        case 0
            name='Chroma';
        case 1
            name='MFCC';
        case 2
            name='Chroma-Test';
        case 3
            name='MFCC-Test';
        case 4
            name='Chroma-Train';
        case 5
            name='MFCC-Train';
        otherwise
            name='';
    end
    timeLength=num2str(timeLength);
    imagesc(distMatrix);
    colormap 'jet';
    colorbar;
    title({'Distance Matrix:';[name '(' timeLength 'seconds)']});
    xlabel('Song A');
    ylabel('Song B');
    saveName=['distanceMatrix' name timeLength '.png'];
    saveas(gca,saveName);

    close(h);
end
end
```

The results from the genre histograms are surprisingly helpful. Punk and Rock (which acoustically sound very similar to each other) have fairly similar histograms. World and Classical (which are known for both being very wide genres) have a large spike at zero. This implies that many songs within the same genre are very loosely related. Electronic and Jazz also have very similar looking histograms. As the two genres personally do not sound very similar, I do not know if this has any quantifiable meaning.

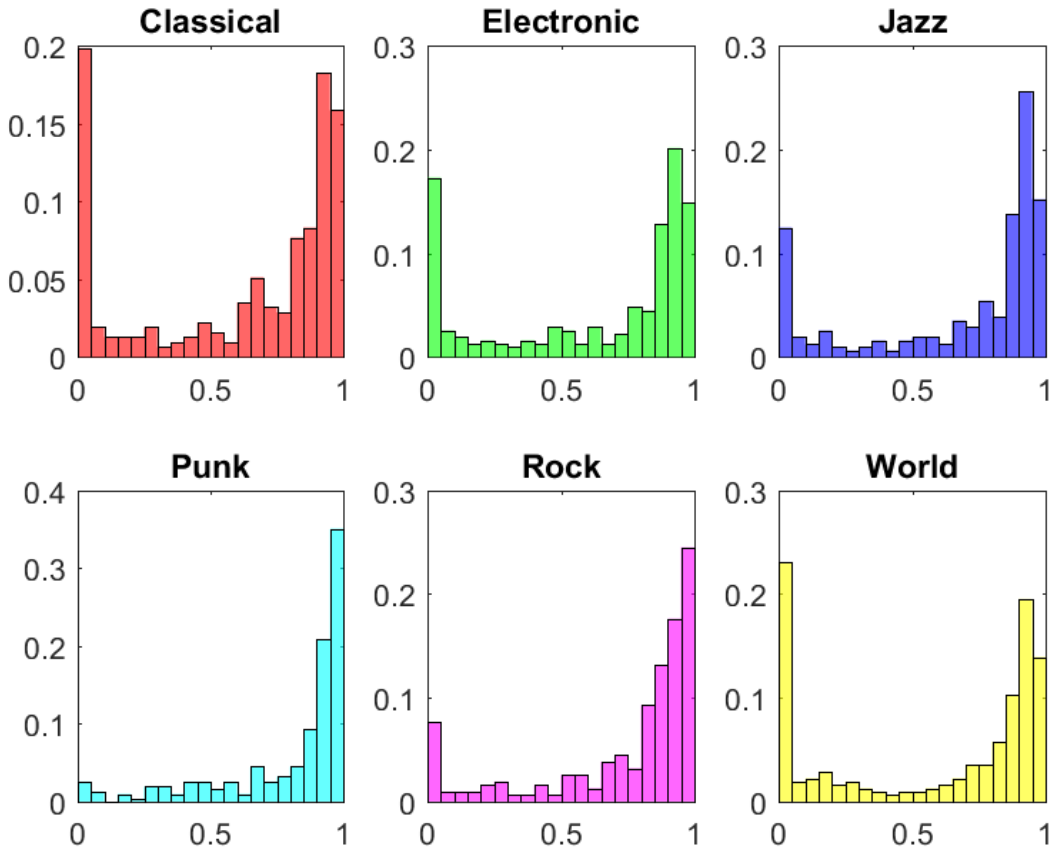


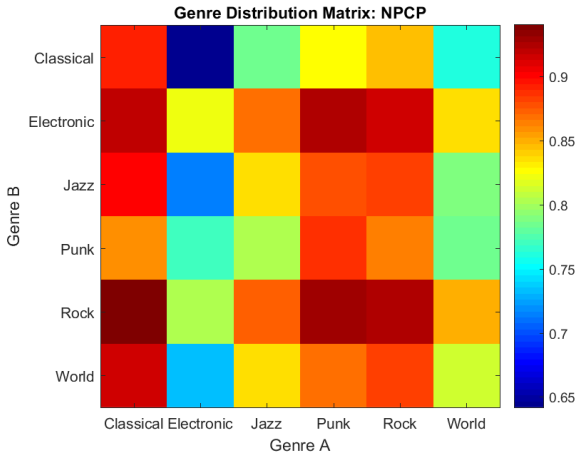
Figure 2: Genre Histograms

Listing 3: genreHistogram.m

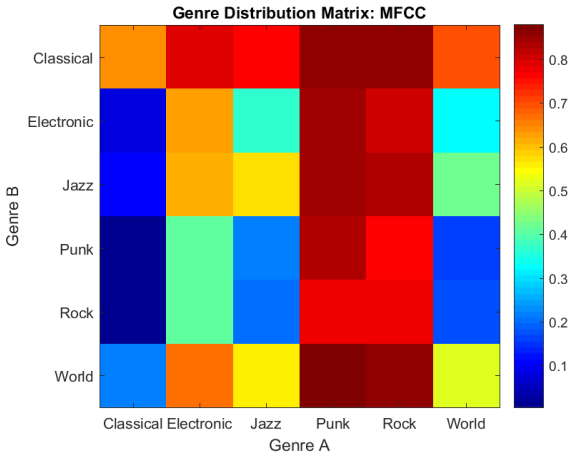
```
function [ genreHisto ] = genreHistogram( xList,~ )
%genreHistogram generates histograms for all 6 genres from distance matrix


a=combnk(1:25,2);
len=nchoosek(25,2);
genreHisto=zeros(25,25,6)-1; % Sanity check for debugging
for j=1:6
    for i=1:len
        ind=a(i,:);
        temp=kullbackDistance(...
            xList{ind(1)+(25*(j-1))},xList{ind(2)+(25*(j-1))});
        genreHisto(ind(1),ind(2),j)=temp;
        genreHisto(ind(2),ind(1),j)=temp; % mirror across diagonol
    end
    for i=1:25
        genreHisto(i,i,j)=1;
    end
end

if(nargin==2)
    h=figure;

    subplot(2,3,1);
```

```
25      h1=histogram(genreHisto(:,:,1));
26      h1.Normalization='probability';
27      h1.BinWidth=0.05;
28      h1.FaceColor='red';
29      title('Classical');
30
31      subplot(2,3,2);
32      h2=histogram(genreHisto(:,:,2));
33      h2.Normalization='probability';
34      h2.BinWidth=0.05;
35      h2.FaceColor='green';
36      title('Electronic');
37
38      subplot(2,3,3);
39      h3=histogram(genreHisto(:,:,3));
40      h3.Normalization='probability';
41      h3.BinWidth=0.05;
42      h3.FaceColor='blue';
43      title('Jazz');
44
45      subplot(2,3,4);
46      h4=histogram(genreHisto(:,:,4));
47      h4.Normalization='probability';
48      h4.BinWidth=0.05;
49      h4.FaceColor='cyan';
50      title('Punk');
51
52      subplot(2,3,5);
53      h5=histogram(genreHisto(:,:,5));
54      h5.Normalization='probability';
55      h5.BinWidth=0.05;
56      h5.FaceColor='magenta';
57      title('Rock');
58
59      subplot(2,3,6);
60      h6=histogram(genreHisto(:,:,6));
61      h6.Normalization='probability';
62      h6.BinWidth=0.05;
63      h6.FaceColor='yellow';
64      title('World');
65      saveas(gca,'genreHistogram.png');
66      close(h);
67  end
68  end
```

Despite taking a while to calculate, once calculated, the distance matrix is very easy to parse. The genreDist function sums $1/36$ of the matrix at a time. Each of these chunks are arranged so that it is entirely the intersection of two genres. And through experimentation, I found that a small value around 0.05 tended to give me the most distinction between genres. This is despite hearing from colleagues that an even smaller value would produce better results. When I tried a smaller γ, all the components on the distance matrix tended towards 1 and I just saw a high saturated graph.



(a) Genre Distribution Matrix: Chroma



(b) Genre Distribution Matrix: MFCC

Listing 4: genreDist.m

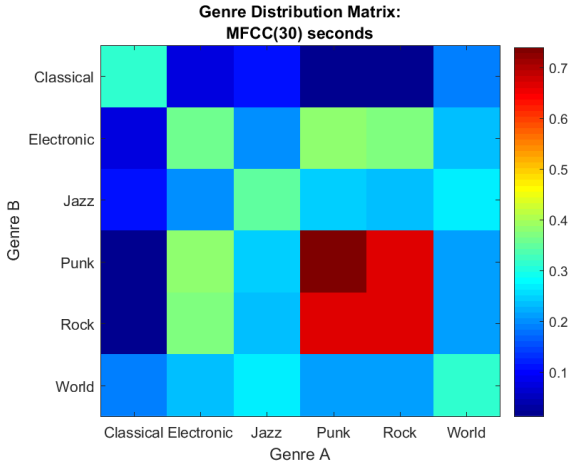
```
function [ genreDistMatrix ] = genreDist(distMatrix,whichPlot,songTime)
%genreDist Computes 6x6 average distance matrix between the genres

genreDistMatrix=zeros(6,6);
[len,width]=size(distMatrix);
if(len~=width)
    error('Matrix must be square');
end
len=len/6; % saves extra divisions later
for i=1:6
    for j=1:6
        genreDistMatrix(i,j)=1/(len*len)*...
            sum(sum(distMatrix((1+(len*(i-1))):len+(len*(i-1))),...
            (1+(len*(j-1))):len+(len*(j-1)))));
    end
end
if(nargin>=2)
    songTime=num2str(songTime);
    h=figure;
    switch whichPlot
        case 0
            name='NPCP';
        case 1
            name='MFCC';
        case 2
            name='NPCP-Test';
        case 3
            name='MFCC-Test';
        otherwise
            name='';
    end
    imagesc(genreDistMatrix);
    ax=gca;
    colorbar;
    colormap 'jet';
    bottomTitle=[name '(' songTime ') seconds'];
    title({'Genre Distribution Matrix:'; bottomTitle});
    xlabel('Genre A');
    ylabel('Genre B');
    ax.XTickLabels={'Classical','Electronic','Jazz','Punk','Rock','World'};
    ax.YTickLabels={'Classical','Electronic','Jazz','Punk','Rock','World'};
    saveas(gca,['genreDistributionMatrix' name songTime '.png']);
    close(h);
```

```
44  end
45  end
```

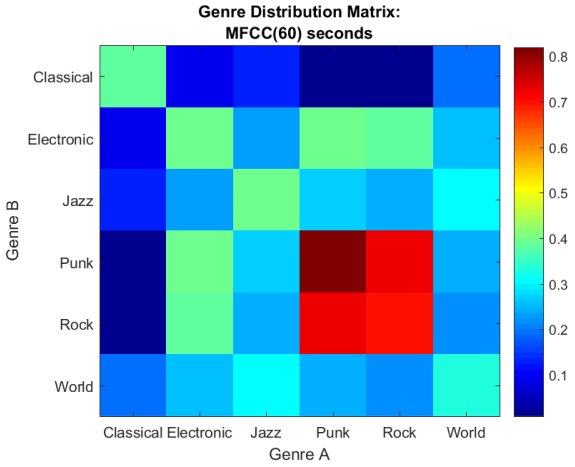
Assignment

7. Evaluate the effect of the length of the audio segment using the following lengths: 30, 60, 120, 240 seconds. For each length you will compare the seperation of the different genres using the 6×6 average distance matrix $\bar{D}$.

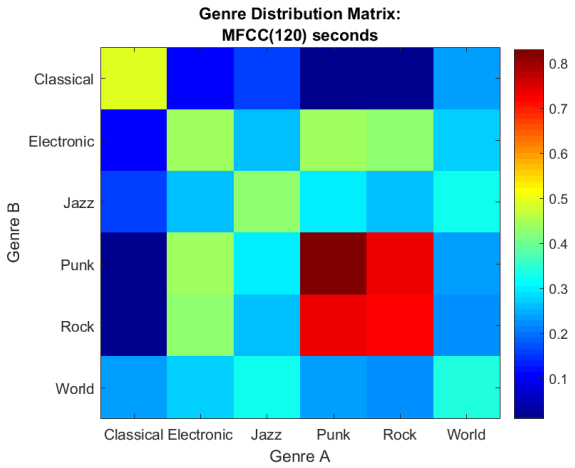
Despite using larger samples for the songs, the genre distribution matrices did not seem to change much. There may be a slight increase in accuracy from 30 seconds to 240 seconds, but the additional computation time required does not warrant the insubstantial improvement.



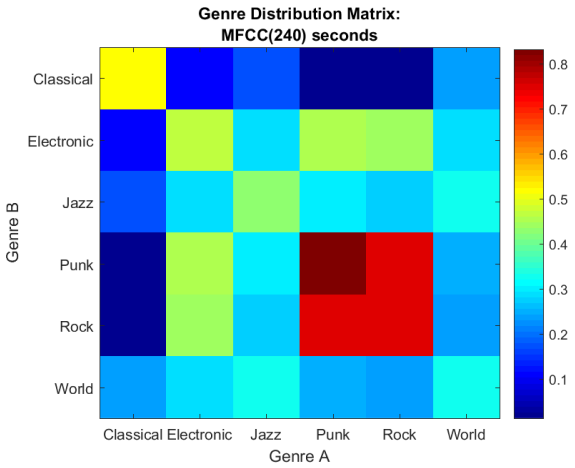
(a) Genre Distribution Matrix: MFCC 30 seconds



(b) Genre Distribution Matrix: MFCC 60 seconds



(c) Genre Distribution Matrix: MFCC 120 seconds



(d) Genre Distribution Matrix: MFCC 240 seconds

8. Implement a classifier based on the following ingredients:
   - computation of the 12 mfcc coefficients, or 12 Normalized Pitch Class Profile
   - modified Kullback-Leibler distance d defined by Equation 2

- genre = majority vote among the 5 nearest neighbors

9. Using cross validation evaluate your classification algorithm. You will compute the mean and standard deviation for all the entries in the confusion matrices.

10. Compare the performance of the classification using the 12 mfcc coefficients, or 12 Normalized Pitch Class Profile.

Upon implementing my confusion matrix, the results were quite surprising. While my previous results looked less than promising, my nearest songs genre picker (as seen in listing 5) seemed to pick the correct genres most of the time, except in the case of World, which should be expected since it has no true classification that puts a song into that genre.

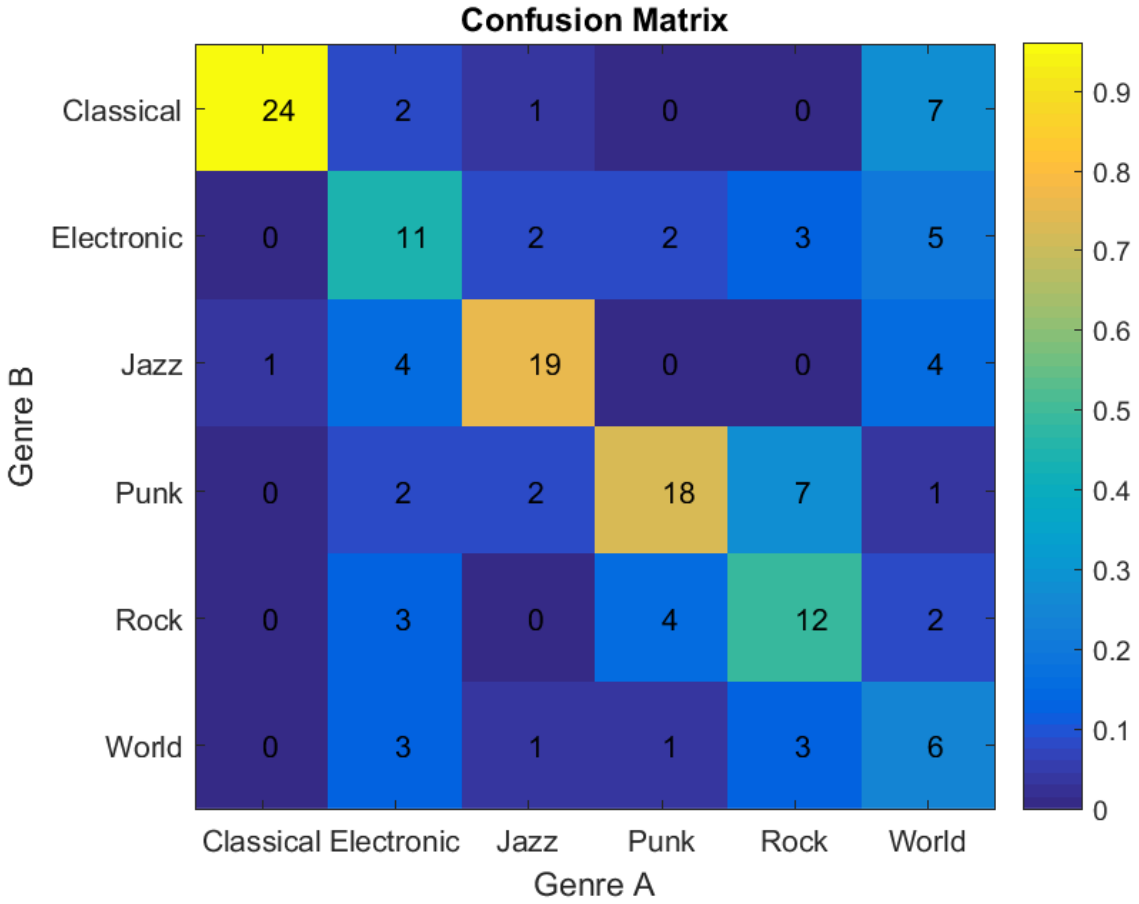


Figure 5: Confusion Matrix

Listing 5: nearestSongs.m

```
1  function [ genreGuess ] = nearestSongs( distMatrixTest,folderNames )
2  %nearestSongs Finds the genre from the 5 nearest songs
3
4  genreLen=length(folderNames); % How many genres
5  [a,b]=size(distMatrixTest);
```

```
distMatrixTest(logical(eye(a,b))) = 0; % set diag to 0
genreGuess=zeros(1,a);
for i=1:a
    [~,index]=sort(distMatrixTest(i,:),'descend');

    genrePicks=floor(index(1:5)/(a/genreLen))+1;
    if(length(unique(genrePicks))==5)
        genreGuess(i)=genrePicks(1);
    else
        genreGuess(i)=mode(genrePicks);
    end
end
end
```

As far as performance, there is no question as whether to use the MFCC coefficients or the Normalized Pitch Class Profile. The MFCC coefficients are computed orders of magnitude faster than the Normalized Pitch Class Profile. It was computationally unreasonable to compute all 150 Normalized Pitch Class Profiles with a time sample of 240 seconds. With an upper bound of 35 seconds on computing one Normalized Pitch Class Profile matrix, computing all 150 tracks would have taken around 1.5 hours. Instead, I opted to only compute the NPCP matrices for all 150 songs using a 30 second sample.

11. Improve the classifier using one of the suggestions above. Two different improvements will earn extra-credit.

A possible improvement could have been using Euclidian distance instead of our custom Kullback-Leibler distance with its weighting kernel. MATLAB has toolkits already to handle classifying data and it naturally uses Euclidian distance because it is easy and also effective.

3 Conclusion

There are many improvements that could be made with these classification methods. A large concern would be to better parallelize this so that it could more efficiently run on a large array of cloud servers by Pandora or Spotify or some other large music service.

All of my work can be found online at my `https://github.com/Krabby127/DSP_Lab`

A Figures

A.1 Distance Matrix

A.1.1 Chroma

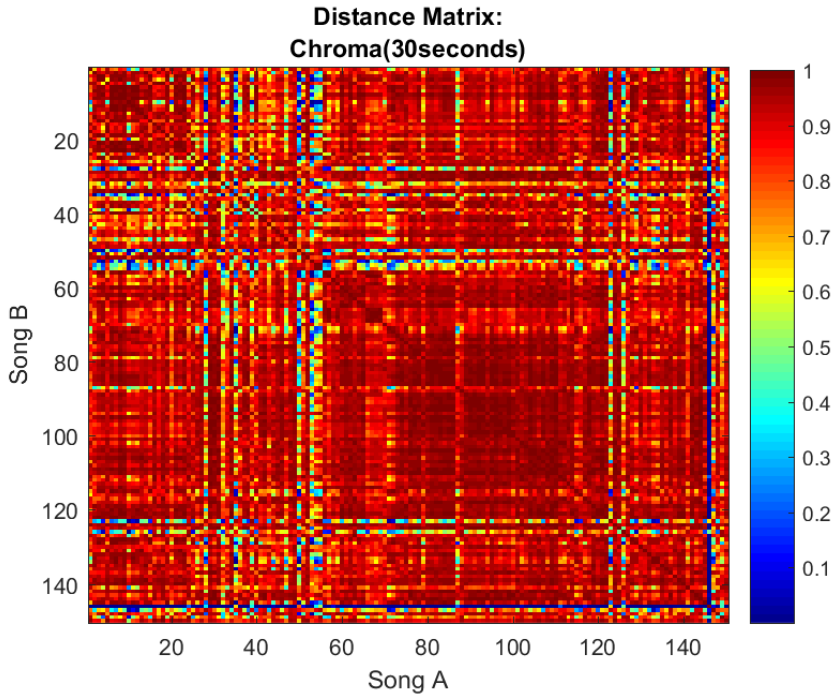


Figure 6: Distance Matrix Chroma: 30 seconds

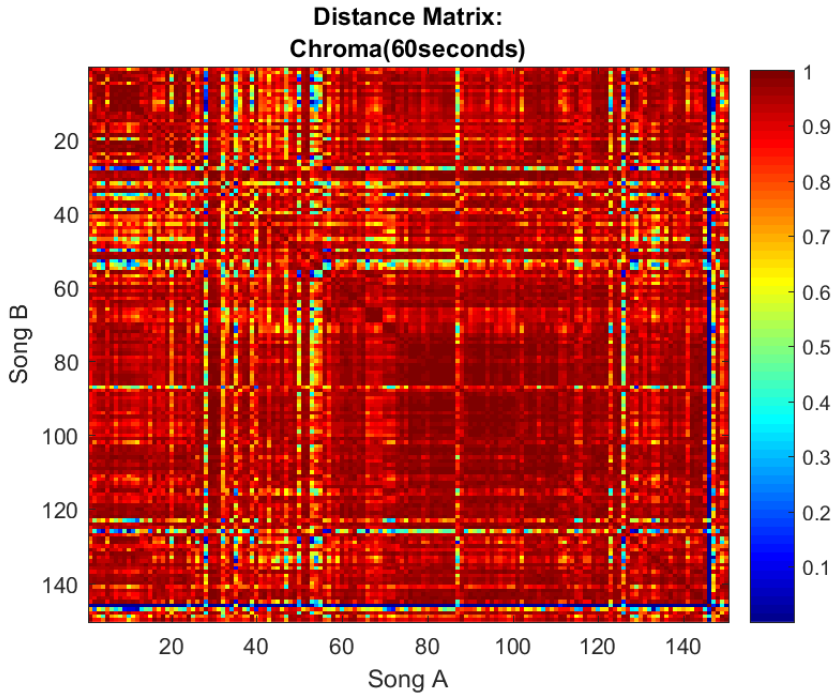


Figure 7: Distance Matrix Chroma: 60 seconds

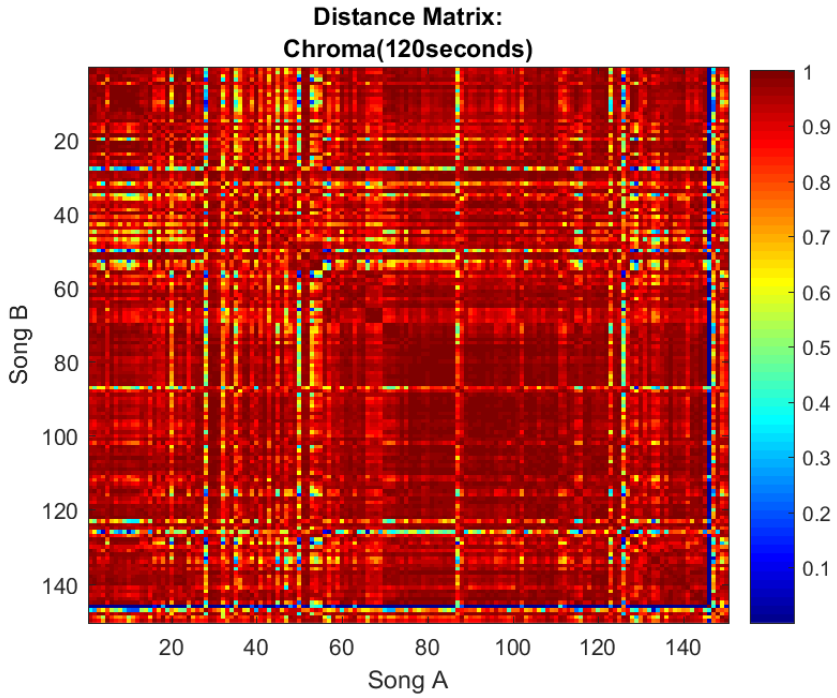


Figure 8: Distance Matrix Chroma: 120 seconds

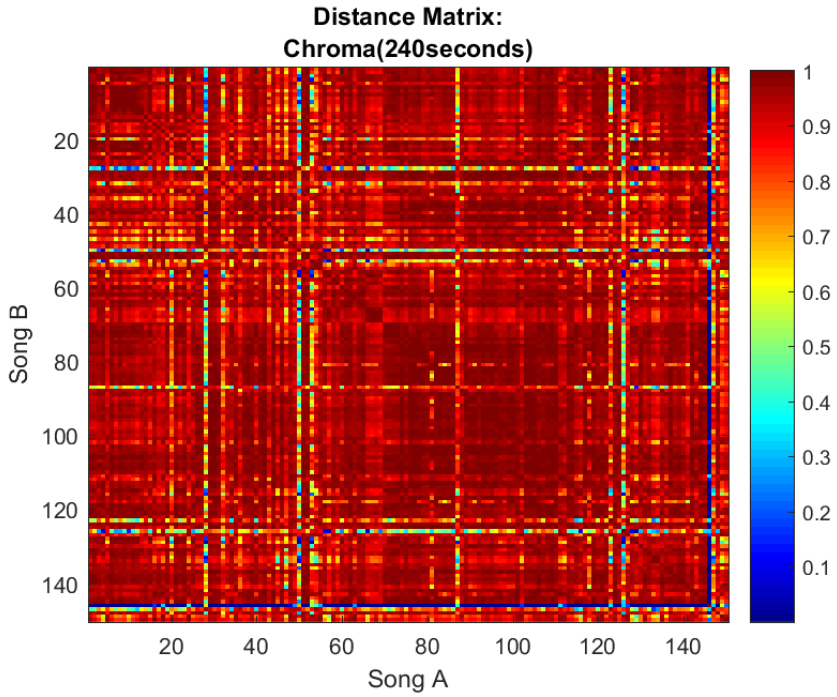


Figure 9: Distance Matrix Chroma: 240 seconds

A.1.2 MFCC

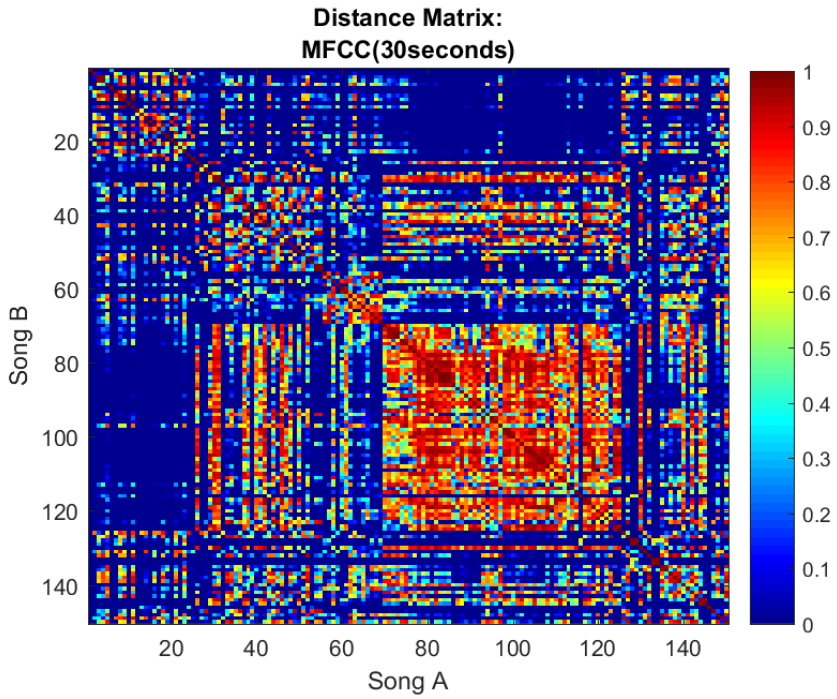


Figure 10: Distance Matrix MFCC: 30 seconds

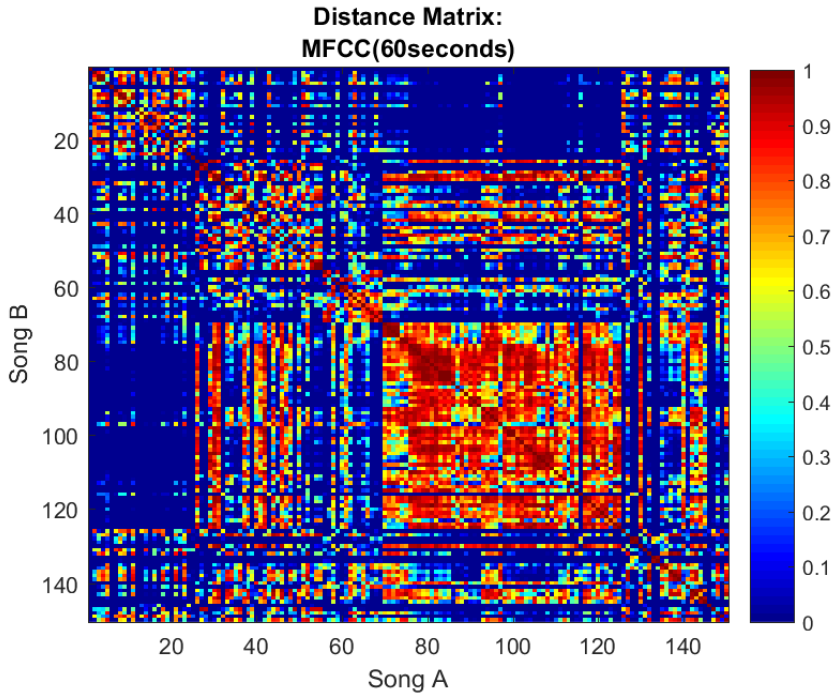


Figure 11: Distance Matrix MFCC: 60 seconds

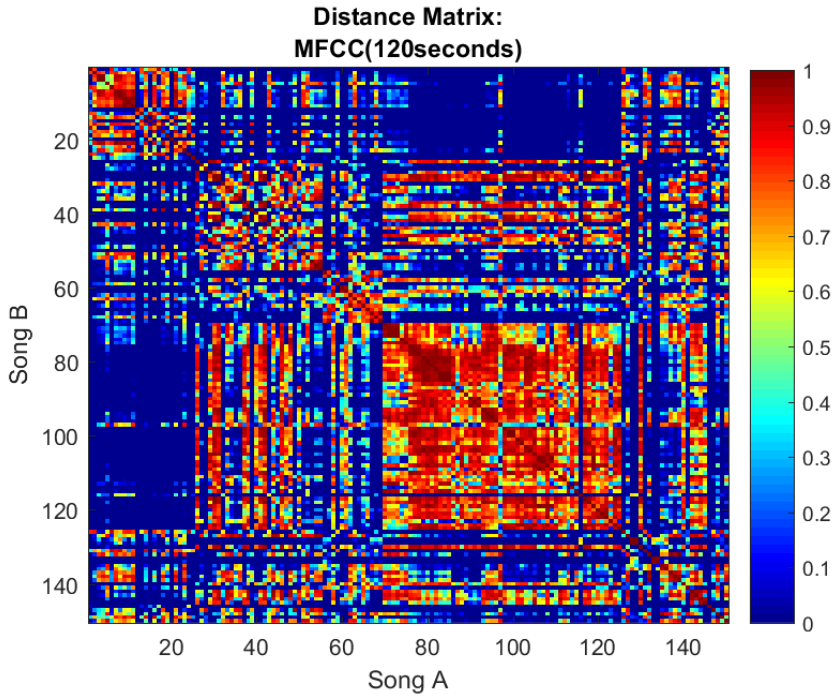


Figure 12: Distance Matrix MFCC: 120 seconds

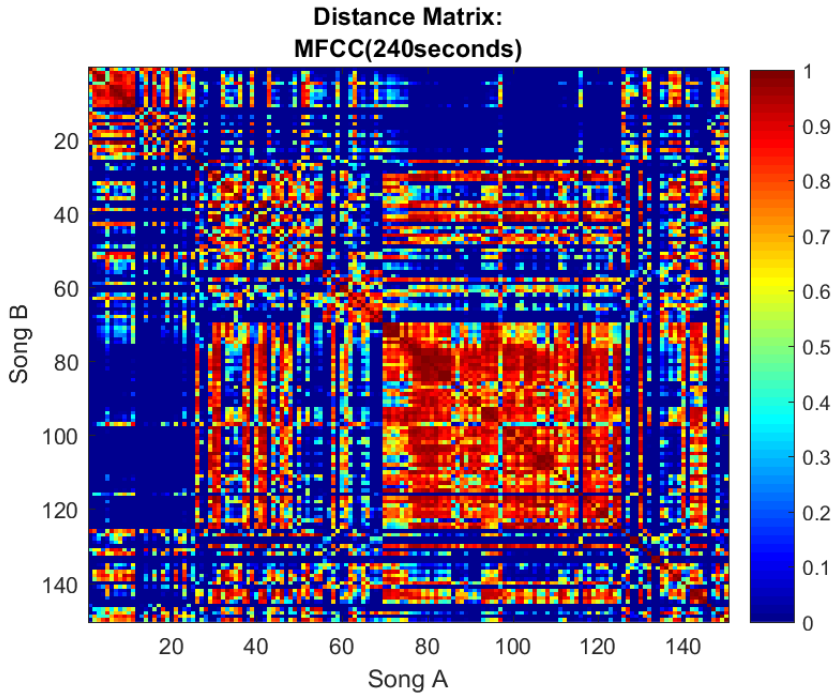


Figure 13: Distance Matrix MFCC: 240 seconds

A.2 Genre Distribution Matrix

A.2.1 Chroma

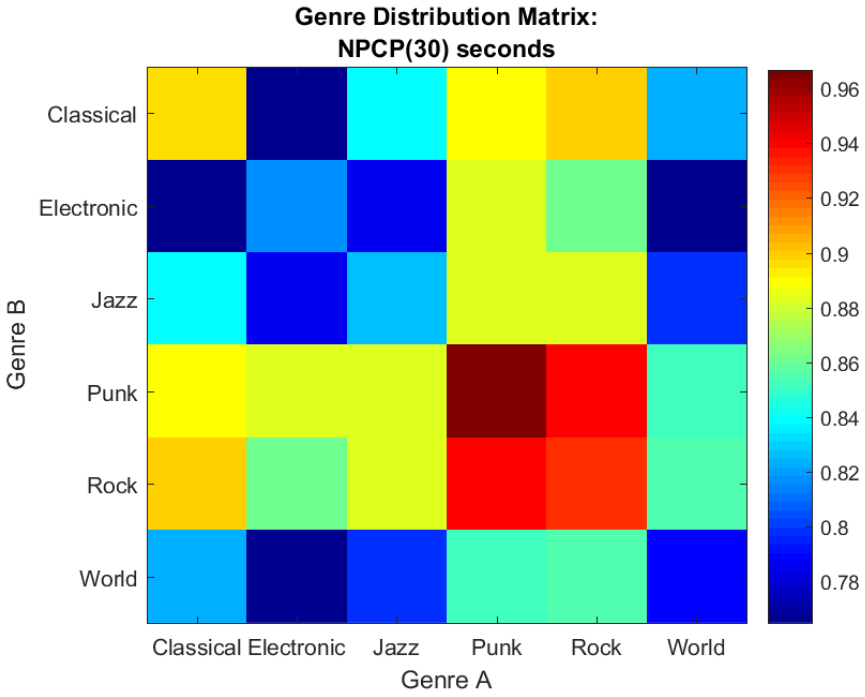


Figure 14: Genre Distribution Matrix Chroma: 30 seconds

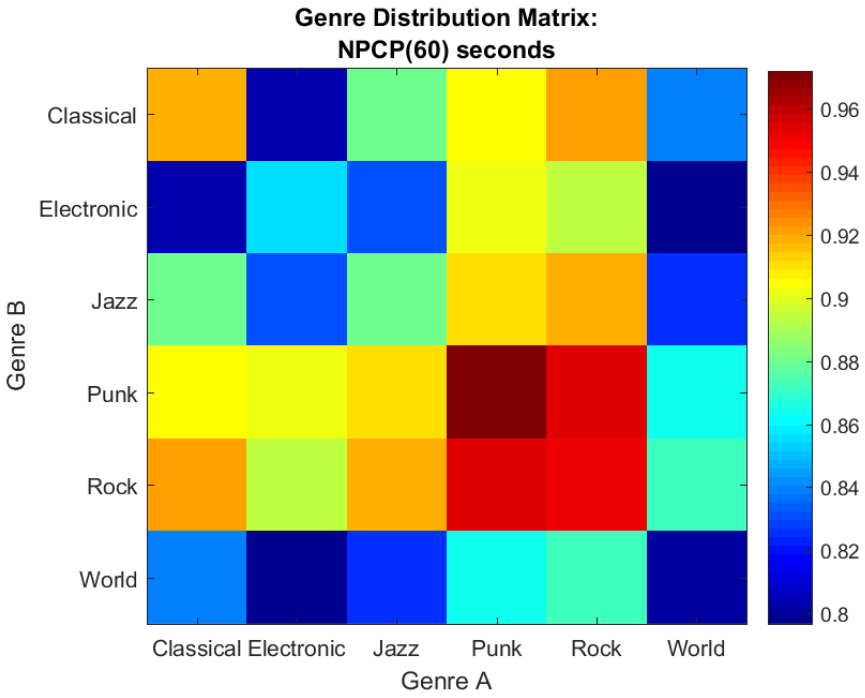


Figure 15: Genre Distribution Matrix Chroma: 60 seconds

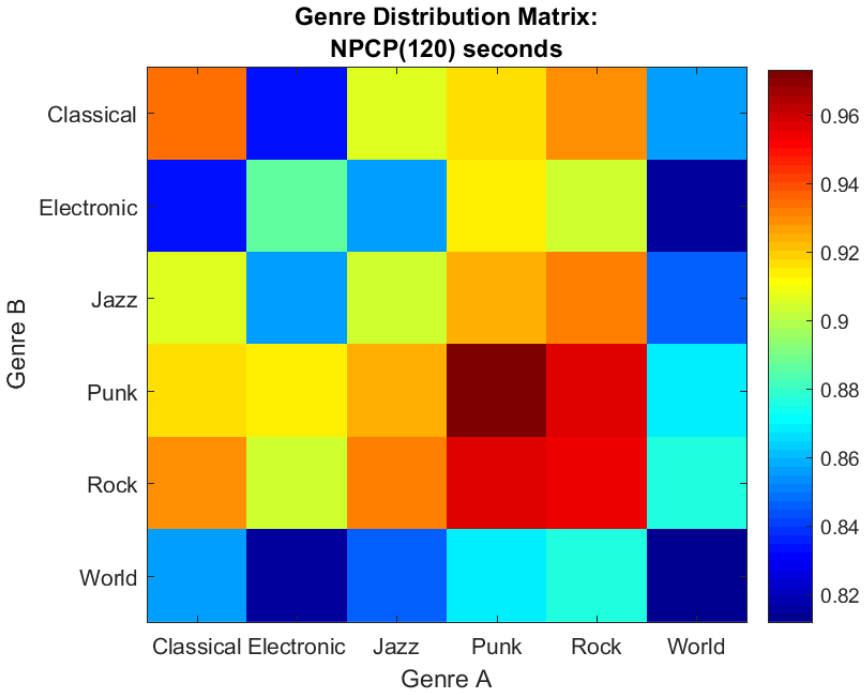


Figure 16: Genre Distribution Matrix Chroma: 120 seconds

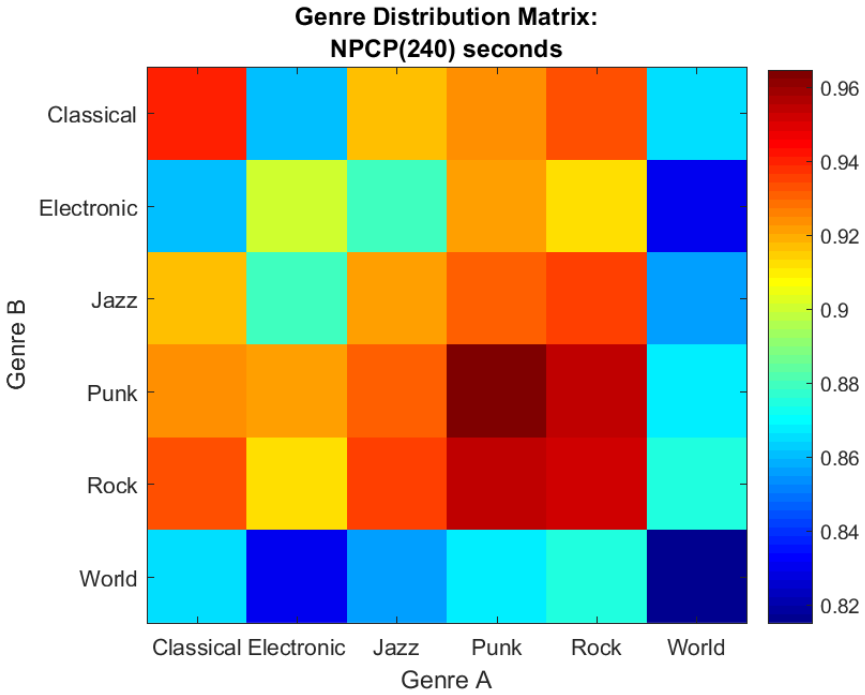


Figure 17: Genre Distribution Matrix Chroma: 240 seconds

A.2.2 MFCC

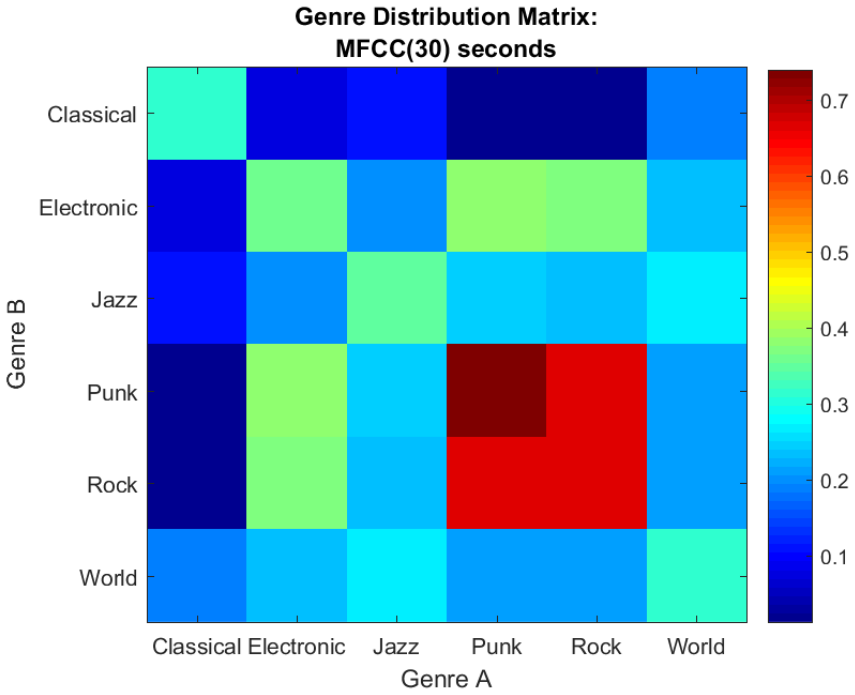


Figure 18: Genre Distribution Matrix Chroma: 30 seconds

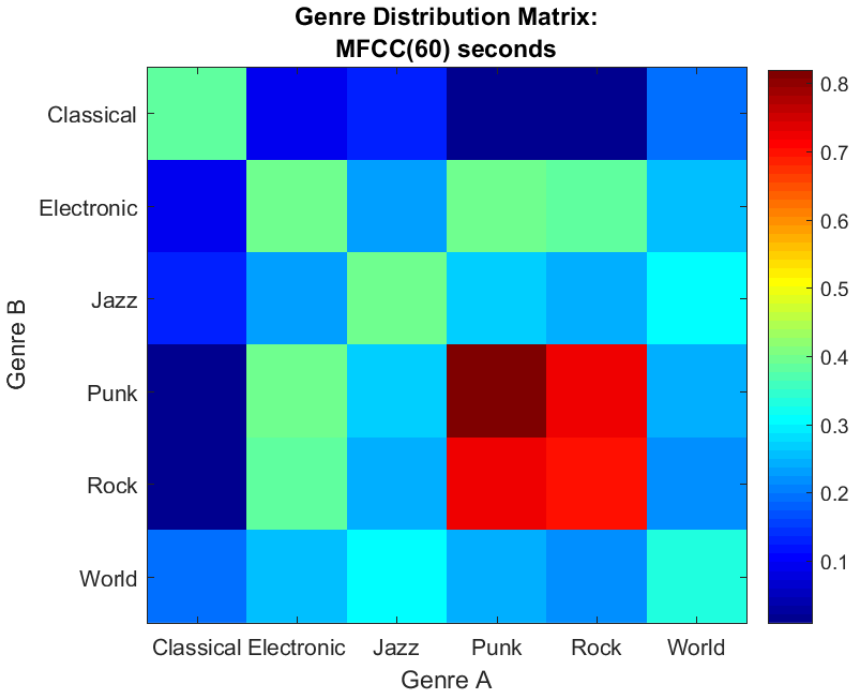


Figure 19: Genre Distribution Matrix Chroma: 60 seconds

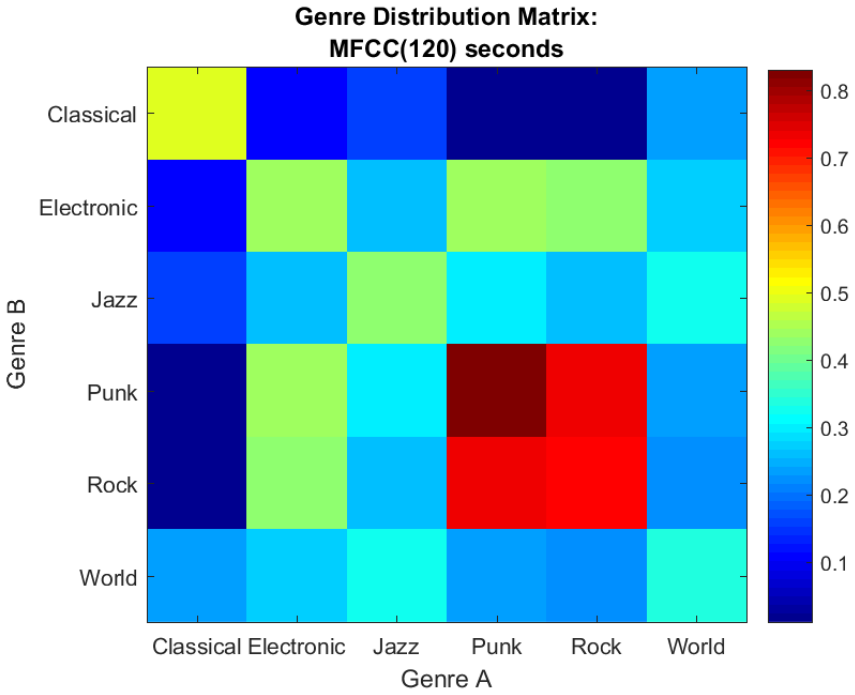


Figure 20: Genre Distribution Matrix Chroma: 120 seconds

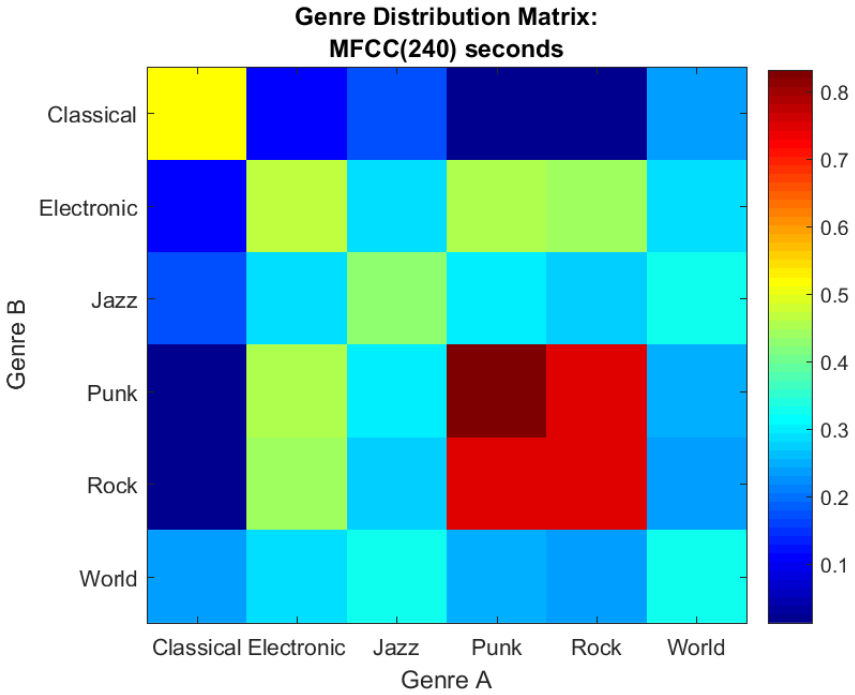


Figure 21: Genre Distribution Matrix Chroma: 240 seconds

B Code

Listing 6: buildReport.m

```
%% Build everything necessary for report
%% Prepare environment
clear d a b c  trackTest trackTrain filename mfccpLists npcpLists ...

clear tracks folderNames;
close all;

%% Generate random list from set of files
d=randTest(1); % generate the random numbers for testing and save to file
[a,b,c]=size(d);
[tracks,folderNames]=setupFiles();
[trackTest,trackTrain]=assignRandTracks(d,tracks);
npcpLists=cell(4,1);
mfccLists=cell(4,1);
%% Run functions on all tracks
timeLength=[30,60,120,240];
genreDistMatrix=zeros(6,6,4,2); % One for each time length and mfcc/chroma
h=waitbar(0,'This takes a while...');
for i=1:4
    mfccLists{i}=mfccpList(tracks,folderNames,timeLength(i));
    waitbar((4*i-1)/16);
    npcpLists{i}=chromaList( tracks,folderNames,timeLength(i));
    waitbar((4*i-2)/16);
    distMatrixM=distanceMatrix(mfccLists{i},1,timeLength(i));
    waitbar((4*i-3)/16);
    genreDistMatrix(:,:,i,1)=genreDist(distMatrixM,1,timeLength(i));
    distMatrixC=distanceMatrix(npcpLists{i},0,timeLength(i));
    genreDistMatrix(:,:,i,2)=genreDist(distMatrixC,0,timeLength(i));
    waitbar((4*i)/16);
end
close(h);
```

Listing 7: kullbackDistance.m

```
function [ d ] = kullbackDistance( P,Q)
%kullbackDistance Computes the distance between two tracks
%   First computes the Kullback-Leibler divergence
%   Then the KL distance is rescaled using an exponential kernel

mu1 = mean(P,2,'omitnan');
Cov1=cov(P','omitrows');
mu2 = mean(Q,2,'omitnan');
Cov2=cov(Q','omitrows');

K=12;

KL=(0.5*trace(pinv(Cov2)*Cov1+(pinv(Cov1)*Cov2)))-K+...
    (0.5*(mu1-mu2)')*(pinv(Cov2)+pinv(Cov1))*(mu1-mu2);

gamma=0.002; % gamma should be very very small
d=exp(-1*gamma*KL);% Apply weighting kernel
end
```

Listing 8: chromaList.m

```
function [ npcpList ] = chromaList( tracks,folderNames,timeLength)
%chromaList Precomputes the npcp for all 150 songs

% allows for easier indexing
```

```
[~,b]=size(tracks);
folderNames=repmat(folderNames,[1,b]);
folderNames=folderNames';

tracks=tracks';% allows us to keep songs within same genre close together
numTracks=numel(tracks);


npcpList=cell(numTracks,1);
parfor i=1:numTracks
    filename=fullfile('..','data',folderNames{i},tracks{i});
    npcpList{i}=normPCP(filename,extractSound(filename,timeLength));
end
end
```

Listing 9: mfccpList.m

```
function [ mfccList ] = mfccpList( tracks,folderNames,timeLength )
%mfccpList Precomputes the mfcc for all 150 songs

% allows for easier indexing
[~,b]=size(tracks);
folderNames=repmat(folderNames,[1,b]);
folderNames=folderNames';

fbank=melBank();
tracks=tracks';% allows us to keep songs within same genre close together
numTracks=numel(tracks);


mfccList=cell(numTracks,1);
parfor i=1:numTracks
    filename=fullfile('..','data',folderNames{i},tracks{i});
    mfccList{i}=mfcc(fbank,freqDist(extractSound(filename,timeLength)));
end
end
```

Listing 10: distanceMatrix.m

```
function [ distMatrix ] = distanceMatrix( xList,whichPlot,timeLength)
%distanceMatrix Computes the distance between every song
%   Uses a precomputed list of the mfcc/normPCP matrices to compute the
%   distance matrix

numTracks=length(xList);
distMatrix=zeros(numTracks,numTracks);

h=waitbar(0,'Computing Distance Matrix...');
for i=1:numTracks
    for j=1:numTracks
        if(i==j)
            distMatrix(i,j)=1;
        else
            distMatrix(i,j)=kullbackDistance(xList{i},xList{j});
        end
    end
    waitbar(i/numTracks)
end

close(h); % close waitbar
if(nargin>=2) %plot graph and save away
    h=figure;
    switch whichPlot
        case 0
            name='Chroma';
```

```
        case 1
            name='MFCC';
        case 2
            name='Chroma-Test';
        case 3
            name='MFCC-Test';
        case 4
            name='Chroma-Train';
        case 5
            name='MFCC-Train';
        otherwise
            name='';
    end
    timeLength=num2str(timeLength);
    imagesc(distMatrix);
    colormap 'jet';
    colorbar;
    title({'Distance Matrix:';[name '(' timeLength 'seconds)']});
    xlabel('Song A');
    ylabel('Song B');
    saveName=['distanceMatrix' name timeLength '.png'];
    saveas(gca,saveName);

    close(h);
end
end
```

Listing 11: genreHistogram.m

```
function [ genreHisto ] = genreHistogram( xList,~ )
%genreHistogram generates histograms for all 6 genres from distance matrix


a=combnk(1:25,2);
len=nchoosek(25,2);
genreHisto=zeros(25,25,6)-1; % Sanity check for debugging
for j=1:6
    for i=1:len
        ind=a(i,:);
        temp=kullbackDistance(...
            xList{ind(1)+(25*(j-1))},xList{ind(2)+(25*(j-1))});
        genreHisto(ind(1),ind(2),j)=temp;
        genreHisto(ind(2),ind(1),j)=temp; % mirror across diagonol
    end
    for i=1:25
        genreHisto(i,i,j)=1;
    end
end

if(nargin==2)
    h=figure;

    subplot(2,3,1);
    h1=histogram(genreHisto(:,:,1));
    h1.Normalization='probability';
    h1.BinWidth=0.05;
    h1.FaceColor='red';
    title('Classical');

    subplot(2,3,2);
    h2=histogram(genreHisto(:,:,2));
    h2.Normalization='probability';
    h2.BinWidth=0.05;
    h2.FaceColor='green';
    title('Electronic');

    subplot(2,3,3);
```

```
    h3=histogram(genreHisto(:,:,3));
    h3.Normalization='probability';
    h3.BinWidth=0.05;
    h3.FaceColor='blue';
    title('Jazz');

    subplot(2,3,4);
    h4=histogram(genreHisto(:,:,4));
    h4.Normalization='probability';
    h4.BinWidth=0.05;
    h4.FaceColor='cyan';
    title('Punk');

    subplot(2,3,5);
    h5=histogram(genreHisto(:,:,5));
    h5.Normalization='probability';
    h5.BinWidth=0.05;
    h5.FaceColor='magenta';
    title('Rock');

    subplot(2,3,6);
    h6=histogram(genreHisto(:,:,6));
    h6.Normalization='probability';
    h6.BinWidth=0.05;
    h6.FaceColor='yellow';
    title('World');
    saveas(gca,'genreHistogram.png');
    close(h);
end
end
```

Listing 12: assignRandTracks.m

```
function [trackTest,trackTrain] = assignRandTracks(d,tracks)
[a,b,c]=size(d); % 6,5,10
trackTest=cell(a,b,c); %6,5,10
[e,f]=size(tracks); %6,25
if(e ~=a)
    error('mismatching input sizes');
end
A=1:f;
A=repmat(A,[f,1,c]);
newA=zeros(a,f-b,c); %6, 20
for i=1:a
    for j=1:c
        newA(i,:,j)=setdiff(A(i,:),d(i,:,j));
    end
end

trackTrain=cell(size(newA));
% trackTrain=tracks;
% % trackTrain=repmat(tracks,[1,1,c]);
for i=1:a %6 genres
    for j=1:(f-b) % all 25 possible songs
        for k=1:c %10 itterations
            % MATLAB really doesn't play well with cells
            if(j<=b)
                trackTest{i,j,k}=tracks{i,d(i,j,k)};
            end
            trackTrain{i,j,k}=tracks{i,newA(i,j,k)};
        end
    end
end



end
```

Listing 13: freqDist.m

```
function [ Xk ] = freqDist(song)
%freqDist Computes the frequency distribution
frames_overlap = buffer(song,512,256);
w=kaiser(512);
N=512;
Y=fft(w.*frames_overlap(:,2));
Xk=Y(1:256);
K=N/2+1;

for i = 1:length(frames_overlap)
    Y=fft(w.*frames_overlap(:,i));
    Xk(1:K,i)=Y(1:K);
end
end
```

Listing 14: extractSound.m

```
function [ soundExtract,p ] = extractSound( filename, time)
%extractSound Extracts time (in seconds) from the middle of the song
%   Write a MATLAB function that extract T seconds of music from a
%   given track. You will use the MATLAB function audioread to
%   read a track and the function play to listen to the track.
narginchk(1, 2);
if(nargin == 1)
    time = 120; %default is to read 2 minutes
end
info = audioinfo(filename);
song=audioread(filename);
if time >= info.Duration
    soundExtract=song;
%     warning('Data may be less accurate with less than 2 minutes');
    if(nargout == 2)
        p=audioplayer(soundExtract,info.SampleRate);
    end
    return;
elseif time<= 1/info.SampleRate
    error('Too small of a time to sample');
end
samples=time*info.SampleRate;

soundExtract=song(floor(info.TotalSamples/2)-floor(samples/2):1: ...
    floor(info.TotalSamples/2)+floor(samples/2));
if(nargout == 2)
    p=audioplayer(soundExtract,info.SampleRate);
end
end
```

Listing 15: genreDist.m

```
function [ genreDistMatrix ] = genreDist(distMatrix,whichPlot,songTime)
%genreDist Computes 6x6 average distance matrix between the genres

genreDistMatrix=zeros(6,6);
[len,width]=size(distMatrix);
if(len~=width)
    error('Matrix must be square');
end
len=len/6; % saves extra divisions later
for i=1:6
    for j=1:6
        genreDistMatrix(i,j)=1/(len*len)*...
            sum(sum(distMatrix((1+(len*(i-1)):len+(len*(i-1))),...
            (1+(len*(j-1)):len+(len*(j-1))))));
    end
```

```
end
if(nargin>=2)
    songTime=num2str(songTime);
    h=figure;
    switch whichPlot
        case 0
            name='NPCP';
        case 1
            name='MFCC';
        case 2
            name='NPCP-Test';
        case 3
            name='MFCC-Test';
        otherwise
            name='';
    end
    imagesc(genreDistMatrix);
    ax=gca;
    colorbar;
    colormap 'jet';
    bottomTitle=[name '(' songTime ') seconds'];
    title({'Genre Distribution Matrix:'; bottomTitle});
    xlabel('Genre A');
    ylabel('Genre B');
    ax.XTickLabels={'Classical','Electronic','Jazz','Punk','Rock','World'};
    ax.YTickLabels={'Classical','Electronic','Jazz','Punk','Rock','World'};
    saveas(gca,['genreDistributionMatrix' name songTime '.png']);
    close(h);
end
end
```

Listing 16: genreHistogram.m

```
function [ genreHisto ] = genreHistogram( xList,~ )
%genreHistogram generates histograms for all 6 genres from distance matrix


a=combnk(1:25,2);
len=nchoosek(25,2);
genreHisto=zeros(25,25,6)-1; % Sanity check for debugging
for j=1:6
    for i=1:len
        ind=a(i,:);
        temp=kullbackDistance(...
            xList{ind(1)+(25*(j-1))},xList{ind(2)+(25*(j-1))});
        genreHisto(ind(1),ind(2),j)=temp;
        genreHisto(ind(2),ind(1),j)=temp; % mirror across diagonol
    end
    for i=1:25
        genreHisto(i,i,j)=1;
    end
end

if(nargin==2)
    h=figure;

    subplot(2,3,1);
    h1=histogram(genreHisto(:,:,1));
    h1.Normalization='probability';
    h1.BinWidth=0.05;
    h1.FaceColor='red';
    title('Classical');

    subplot(2,3,2);
    h2=histogram(genreHisto(:,:,2));
    h2.Normalization='probability';
    h2.BinWidth=0.05;
```

```
    h2.FaceColor='green';
    title('Electronic');

    subplot(2,3,3);
    h3=histogram(genreHisto(:,:,3));
    h3.Normalization='probability';
    h3.BinWidth=0.05;
    h3.FaceColor='blue';
    title('Jazz');

    subplot(2,3,4);
    h4=histogram(genreHisto(:,:,4));
    h4.Normalization='probability';
    h4.BinWidth=0.05;
    h4.FaceColor='cyan';
    title('Punk');

    subplot(2,3,5);
    h5=histogram(genreHisto(:,:,5));
    h5.Normalization='probability';
    h5.BinWidth=0.05;
    h5.FaceColor='magenta';
    title('Rock');

    subplot(2,3,6);
    h6=histogram(genreHisto(:,:,6));
    h6.Normalization='probability';
    h6.BinWidth=0.05;
    h6.FaceColor='yellow';
    title('World');
    saveas(gca,'genreHistogram.png');
    close(h);
end
end
```

Listing 17: mellBank.m

```
function [ fbank ] = melBank(~)
%melBank Creates a set of mel filter banks
%   Implement the computation of the triangular filterbanks
%   Hp, p = 1,...,NB. Your function will return an array fbank of size
%   NB x K such that fbank(p,:) contains the filter bank Hp.

[~,fs]=audioread(fullfile('..','data','classical',...
    'artist_1_album_2_track_2.wav')); % use this file as sample for fs
N=512;
K=N/2+1;

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
    [~,indx] = min(abs(melFrq - melIdx(i)));
    melIdx2Frq(i) = linFrq(indx);
end

fbank=zeros(nbanks,K);
%mapping frequencies banks
fbank_freq = linspace(0, 5512, 257);

```

```
for i = 2:nbanks+1
    range = 2/(melIdx2Frq(i+1) - melIdx2Frq(i-1));

    for j = 1:K
        filt_val = range;

        if (fbank_freq(j)<=melIdx2Frq(i))&&(fbank_freq(j)>melIdx2Frq(i-1))

            filt_val = filt_val*((fbank_freq(j)-melIdx2Frq(i-...
                1))/(melIdx2Frq(i) - melIdx2Frq(i-1)));
            fbank(i-1,j) = filt_val;

        elseif (fbank_freq(j) < melIdx2Frq(i+1)) && (fbank_freq(j) >=...
                melIdx2Frq(i))

            filt_val = filt_val*((melIdx2Frq(i+1) -...
                fbank_freq(j))/(melIdx2Frq(i+1) - melIdx2Frq(i)));
            fbank(i-1,j) = filt_val;

        else
            fbank(i-1,j) = 0;
        end
    end
end

if(nargin)
    h=figure;
    plot(fbank.');
    title('Mel Filter Bank');
    xlabel('Frequency (Hz)');
    ylabel('Filter Magnitude');
    xlim([0,length(fbank)]);
    saveas(gca,'melFilterBank.png');
    close(h);
end
end
```

Listing 18: mfcc.m

```
function [ mfccp ] = mfcc( fbank,Xn,~)
%mfcc Computes the Mel frequency coeffecients
%   The mel-spectrum (MFCC) coefficient of the n-th frame is defined for
%   p = 1,...,NB
narginchk(2, 3);
% Xn=freqDist(filename);
mfccp=(fbank*abs(Xn)).^2;

% Convert to only using 12 banks
t = zeros(1,36);
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;
mel2 = zeros(12,size(mfccp,2));
for i=1:12,
    mel2(i,:) = sum(mfccp(t==i,:),1);
end
mfccp=mel2;
% Lets us still use non-log mfcc (if needed)
if(nargin==3)
    mfccp=10*log10(mfccp);
    return;
end
end
```

Listing 19: normPCP.m

```
function [PCP] = normPCP( filename,x,~)
%rhythmIndex Identifies rhythmic periods
%   B(l) is the sum of all the entries on the lth upper diagonal.
%   The index l associated with the largest value of B(l) corresponds to
%   the presence of a rhythmic period of l*K/f_s seconds
info=audioinfo(filename);
% x=extractSound(filename);
[s,~,~]=spectrogram(x,kaiser(512),256,512,info.SampleRate,'yaxis');
s=abs(s); % Need to work with non-complex values
[a,b]=size(s);

% pks=zeros(a,b);
locs=zeros(a,b);
peaks=zeros(a,b);
for i=1:b
    len=length(findpeaks(s(:,i)));
    [peaks(1:len,i),locs(1:len,i)]=findpeaks(s(:,i));
end
freqVals=(info.SampleRate/2/a)*locs;
f0=27.5;
sm=round(12*log2(freqVals/f0));
r=12*log2(freqVals/f0)-sm;
c=mod(sm,12);
[~,B]=size(c); % 257,1032

w=(cos(pi*r/2)).^2;
w(isnan(w))=0;

% k is the number of peaks
% c is the chroma (A,A#,B,etc.)
% n is the frame number
% Weighting function, w, needs to be k*n*c large

PCP=zeros(12,B);
peaks=peaks.^2;

for i=1:B %nFrames
    for j=0:11
        a=find(c(:,i)==j);
        PCP(12-j,i)=sum(w(a,i).*peaks(a,i));
    end
end
PCP=bsxfun(@rdivide,PCP,sum(PCP));
PCP(isnan(PCP))=0;
if(nargin==3)
    h=figure;
    [pathstr,name,ext] = fileparts(filename) ;
    genre=pathstr(find(pathstr==filesep,1,'last')+1:end);
    bottomTitle=[genre ' ' name ext];
    imagesc(PCP);
    ax=gca;
    ax.YTickLabel=fliplr({'A ','A#','B ','C ','C#','D ','D#','E ',...
        'F ','F#','G ','G#'});
    ax.YTick=linspace(1,12,12);
    colormap 'jet';
    % make underscores appear fine
    title({'Chroma:';bottomTitle},'Interpreter','none');
    xlabel('frames');
    ylabel('Note');
    colorbar;
    saveas(gca,['NPCP_' upper(genre) name '.png']);
    close(h);
end
end
```

Listing 20: nearestSongs.m

```
function [ genreGuess ] = nearestSongs( distMatrixTest,folderNames )
%nearestSongs Finds the genre from the 5 nearest songs

genreLen=length(folderNames); % How many genres
[a,b]=size(distMatrixTest);
distMatrixTest(logical(eye(a,b))) = 0; % set diag to 0
genreGuess=zeros(1,a);
for i=1:a
    [~,index]=sort(distMatrixTest(i,:),'descend');

    genrePicks=floor(index(1:5)/(a/genreLen))+1;
    if(length(unique(genrePicks))==5)
        genreGuess(i)=genrePicks(1);
    else
        genreGuess(i)=mode(genrePicks);
    end
end
end
```

Listing 21: randTest.m

```
function [d] = randTest(~)
%randTest Setup random numbers to use for testing
% Assumes 6 genres, 5 songs each, and 10 passes
% need 6*5*10 random numbers
d=zeros(6,5,10); % hold the index for random songs
for n=1:10 % average over the randomization
    for i=1:6 % an array for each genre
        % generates 5 random numbers between 1 and 25
        d(i,:,n)=randperm(25,5);
    end
end
if(nargin)
    % We now have d, which is our song indices
    % Save away the list of indices to a file
    DateString=datestr(datetime('now'));
    DateString(DateString==' ')='_';
    DateString(DateString==':')='';
    name= ['randomNum_', DateString '.mat'];
    save(name,'d');
end
end
```

Listing 22: setupFiles.m

```
function [tracks,folderNames] = setupFiles()
folderNames=ls(fullfile('..','data')); % grab foldernames in "data" folder
if (isunix == 0)
    folderNames(1:2,:)=[]; % first two will always be . and .. (in Windows)
end
if isunix
    folderNames=strsplit(folderNames);
    folderNames(end)=[]; % Remove space
end
folderNames=cellstr(folderNames); % convert to cells
if isunix
    folderNames=folderNames';
end
NGenres=length(folderNames); % count the number of genre folders
NSongs = 25; % assuming 25 songs per genre
tracks=cell(NGenres,NSongs); % crete an empty cell array
for i=1:NGenres
    % feed into array
    trackTemp=ls(fullfile('..','data', folderNames{i},'*') );
```

```
20
21      if (isunix==0)
22          trackTemp(1:2,:)=[]; % first two will always be . and .. (in Windows)
23      end
24      if isunix
25          trackTemp=strsplit(trackTemp);
26          trackTemp(end)=[];
27      end
28
29      trackTemp=cellstr(trackTemp); % convert to cells
30      for j=1:NSongs
31          [~,name,ext]=fileparts(trackTemp{j});
32          tracks{i,j}=[name ext]; % MATLAB doesn't like cell math
33      end
34  end
35  end
```

Listing 23: confuseMatrix.m

```
1   function [ confusionMatrix] = confuseMatrix( genreGuess,~ )
2   %confuseMatrix Computes a confusion matrix from the genre guesses
3
4   confusionMatrix=zeros(6,6)-1;% helpful for debugging
5   A=reshape(genreGuess,[length(genreGuess)/6,6]);
6   for i=1:6
7       for j=1:6
8           confusionMatrix(j,i)=sum(numel(find(A(:,i)==j)));
9       end
10  end
11  if nargin==2
12      h=figure;
13      confuseString=num2str(confusionMatrix(:));
14      textStrings = strtrim(cellstr(confuseString));
15      imagesc(confusionMatrix./25);
16      ax=gca;
17      for i=1:6
18          for j=1:6
19              text(j,i,textStrings{sub2ind([6,6],i,j)});
20          end
21      end
22      colorbar;
23      % colormap jet; % can't see numbers with "jet"
24      title({'Confusion Matrix'});
25      xlabel('Genre A');
26      ylabel('Genre B');
27      ax.XTickLabels={'Classical','Electronic','Jazz','Punk','Rock','World'};
28      ax.YTickLabels={'Classical','Electronic','Jazz','Punk','Rock','World'};
29      saveas(ax,'confusionMatrix.png');
30      close(h);
31  end
32  end
```