ECEN4532, DSP Laboratory

Music Classification

Lab 3

Jeffery Lim
Jeffery.Lim@colorado.edu

Contents

1 Introduction

For the last lab concerning music, songs will finally be classified and a prediction method will be developed in order to properly determine the genres of each song. The first half of the lab is to determine a good way of calculating distance, and this will be utilized by finding KLDistances. This distance matrix will then be used in order to do classifying of genres.

2 KLDistance

The KL distance function takes in a two coefficient matrices and a gamma value. The one big factor is the introduction of $-\infty$. This was discovered during debugging and identical songs would cause the exponential term to go to infinity, thus making d = $-\infty$. This was fixed where $-\infty$ was replaced with 1. To calculate d, a different method was used from guidance from Professor Francois Meyer. This means the gamma value actually ranges from 10 to 100. Using the original distance calculation would only cause the distance matrix to place heavy emphasis on the diagonals, and not the actual collection of songs.

```
1  function [ d ] = KLDistance(coefficient1, coefficient2, gam)
2  % KLDistance calculates the KL Distance between two coefficient matrix
3
4  %calculate mean and covariance of both songs
5  mu0 = mean(coefficient1,2);
6  cov0 = cov(coefficient1');
7
8  mu1 = mean(coefficient2,2);
9  cov1 = cov(coefficient2');
10
11 icov0 = inv(cov0);
12 icov1 = inv(cov1);
13
14 %KL   =  1/2*(trace(icov1*cov0) + (mu1 - mu0)'*(icov1)*(mu1-mu0) - 12 + ...
       log(det(cov1/cov0)));
15 KL = 1/2*(trace(icov1*cov0 + icov0*cov1)) - 12 + 1/2*((mu1 - ...
       mu0)'*(icov1+icov0)*(mu1-mu0));
16 %KL = 0.5*(trace(cov0*icov1) + trace(cov1*icov0)) + ...
       0.5*trace((icov0+icov1)*(mu0-mu1)*(mu0-mu1)') - 12;
17 %d    = exp(-gam*(KL));
18 d    = (1-exp(-gam/(KL + eps)));
19
20 %If the songs are exactly the same, d will go to -inf
21 %Remapping it to 1
22 if d == -inf
23     d = 1;
24 end
25
26 end
```

2.1 Distance Calculation

2.2 Preparation

The preparation of the mfcc and npcp distance matrix involved organizing the data in a fast and efficient method. The first part of the code calculates the mfcc and npcp matrix values for each song and saves them into a results directory.

The steps is as follows:

- Load all songs
- mfcc and npcp matrix generation
- save mfcc and npcp to a result directory

A lot more work could be done to minimize the amount of work the code takes, such as going further and calculating μ or Cov as opposed to the entire mfcc and npcp matrix. For the testing of different gamma values, 120 seconds were taken from each song.

```
%lab 3
%This workspace is necessary for loading alll the data for easier analysis
%and reuse. This needs to be run before workspace_b

close all;
clear;

%Adding the path of data into the current directory
addpath(genpath('audio/'));

%Load all tracks
classical = dir('audio/classical/*.wav');
electronic = dir('audio/electronic/*.wav');
jazz = dir('audio/jazz/*.wav');
punk = dir('audio/punk/*.wav');
rock = dir('audio/rock/*.wav');
world = dir('audio/world/*.wav');

%Put each music genre in its own row
music = [classical, electronic, jazz, punk, rock, world];
music_labels = {'classical';'electronic'; 'jazz'; 'punk'; 'rock'; 'world'};

%Define frame size
frameL = 512;
N = 512;

%Song Length
songL = 240;

%Status report
statusbar = 1;
totalbar = size(music,1) * size(music,2);

for i=1:size(music,2)
    for j = 1:size(music,1)

    fprintf('Percent done: %3.2f%%\n', statusbar/totalbar * 100); %Status Bar
    statusbar = statusbar + 1;

    %obtain the song and clip the song
    [wav, fs] = audioread(music(j, i).name);
    if length(wav) < songL
            clipLength = floor(length(wav)/fs);
        else
            clipLength = songL;
    end;

    clippedWav = clipAudio(wav,fs, clipLength);

    w = kaiser(N);
    %Calculating the fbanks and the mfcc

```

```
53      [Hp] = fbanks(fs, N);
54      [MFCC] = mfcc(clippedWav, N, w, Hp);
55      [NPCP] = npcp(clippedWav, fs, frameL, w);
56
57      %Gets rid of extra frames that are NANs
58      MFCC = MFCC(:, 1:end-1);
59
60       %Remapping mfcc to different notes
61      t = zeros(1,36);
62      t(1)    = 1; t(7:8)    = 5; t(15:18)= 9;
63      t(2)    = 2; t(9:10) = 6; t(19:23) = 10;
64      t(3:4) = 3; t(11:12) = 7; t(24:29) = 11;
65      t(5:6) = 4; t(13:14) = 8; t(30:36) = 12;
66
67      mel = zeros(12, length(MFCC));
68      for k=1:12,
69          mel(k,:) = sum(MFCC(t==k,:),1);
70      end
71      MFCC = mel;
72
73      %Write mfcc data
74      fid = fopen(char(strcat('results/mfcc/', strcat(strcat(music_labels(i), ...
            num2str(j)), '.dat'))), 'wt');
75      for k = 1:size(MFCC,1)
76          fprintf(fid, '%d ', MFCC(k,:));
77          fprintf(fid, '\n');
78      end
79      fclose(fid);
80
81       %Write npcp data
82      fid = fopen(char(strcat('results/npcp/', strcat(strcat(music_labels(i), ...
            num2str(j)), '.dat'))), 'wt');
83      for k = 1:size(NPCP,1)
84          fprintf(fid, '%d ', NPCP(k, :));
85          fprintf(fid, '\n');
86      end
87      fclose(fid);
88
89      end
90  end
```

workspace_a.m is the first step, which loads the songs and saves the mfcc and npcp matrix into a directory, which would be later reloaded. This saves a lot of time on the calculation and optimization of gamma.

```
1   function [mfcc, npcp] = loadCoefficients
2   %This function loads the mfcc and npcp dir and stores the data from the
3   %previous function.
4   %Returns cell arrays of the imported data
5
6   mfcc_dir = dir('results/mfcc/*.dat');
7   npcp_dir = dir('results/npcp/*.dat');
8
9   %Import mfcc and npcp data
10  for i = 1:size(mfcc_dir)
11      fprintf('Loading data: %3.2f%%\n', i/length(mfcc_dir)*100); %Status Bar
12      mfcc{i} = importdata(strcat('results/mfcc/', mfcc_dir(i).name));
13      npcp{i} = importdata(strcat('results/npcp/', npcp_dir(i).name));
14  end;
15
16  return;
```

loadCoefficients is the code to run when importing the data from the result of workspace_a.m. Each mfcc and npcp matrix is stored into a cell array, which gets returned.

2.3 Computing Distance

```
function [D_mfcc, D_npcp] = computeDistance(gam, mfcc, npcp)
addpath(genpath('results/'));

%Preallocate the matrices
D_mfcc = zeros(length(mfcc), length(mfcc));
D_npcp = D_mfcc;

%Analyze the KLDistances
for i = 1:length(mfcc)
    fprintf('Analyzing Data at gam: %d : %3.2f%%\n', gam, ...
        i/length(mfcc)*100); %Status Bar
    for j = 1:i
        %Retrieve the two sets of data
        mfcc_i = mfcc{i};
        npcp_i = npcp{i};

        mfcc_j = mfcc{j};
        npcp_j = npcp{j};

        %Calculate the KLDistances
        D_mfcc(i,j) = KLDistance(mfcc_i, mfcc_j, gam);
        D_npcp(i,j) = KLDistance(npcp_i, npcp_j, gam);

        %Since i -> j is the same as j -> i
        D_mfcc(j,i) =  D_mfcc(i,j);
        D_npcp(j,i) = D_npcp(i,j);

    end
end

end
```

The computeDistance function takes the γ, and both mfcc and npcp matrix, and analyzes the KL distance between all songs.

2.4 Gamma Optimization

```
%Chunksize and hopsize for 25 songs per category
chunkSize_i = 1:25;
chunkSize_j = 1:25;
hopSize = 25;

%Load coefficients of mfcc and npcp if it is not in the workspace already
if ¬exist('mfcc', 'var')
    [mfcc, npcp] = loadCoefficients;
end

for gam = 10:10:100
%Compute the distances
[D_mfcc, D_npcp] = computeDistance(gam, mfcc, npcp);

%plots
figure(1);
```

```
subplot(1,2,1);
imagesc((D_mfcc));

title('MFCC Distance Matrix');xlabel('Songs');ylabel('Songs');
colorbar;

subplot(1,2,2);
imagesc((D_npcp));

title('NPCP Distance Matrix');xlabel('Songs');ylabel('Songs');
colorbar;

saveImage('gamma.png', num2str(gam));

averageD_mfcc = zeros(6,6);
averageD_npcp = zeros(6,6);

%Calculating the averages of D_mfcc and D_npcp
for i = 1:6
    for j = 1:6
        averageD_mfcc(i,j) = mean(mean(D_mfcc(chunkSize_i, chunkSize_j)));
        averageD_npcp(i,j) = mean(mean(D_npcp(chunkSize_i, chunkSize_j)));

        chunkSize_j = chunkSize_j + hopSize;
    end;
    chunkSize_i = chunkSize_i + hopSize;
    chunkSize_j = 1:25;
end;

chunkSize_i = 1:25;
chunkSize_j = 1:25;

%Normalizing the average
averageD_mfcc = averageD_mfcc/max(max(averageD_mfcc));
averageD_npcp = averageD_npcp/max(max(averageD_npcp));

%Math in order to determine the max separation values between the
%distance
for i = 1:6
    maxSep_mfcc(i, gam/10) =  max(averageD_mfcc(i,:)) - max(averageD_mfcc(i, ...
        averageD_mfcc(i,:)<max(averageD_mfcc(i,:))));
    maxSep_npcp(i, gam/10) =  max(averageD_npcp(i,:)) - max(averageD_npcp(i, ...
        averageD_npcp(i,:)<max(averageD_npcp(i,:))));
end

%Graphing the average distance matrix
genre = {'classical', 'electronic' , 'jazz', 'punk', 'rock', 'world' };
figure(2);
subplot(1,2,1);
imagesc(averageD_mfcc);
title('Average MFCC Distances');
colorbar; colormap jet;set(gca, 'XtickLabel', genre(:), 'YtickLabel', genre(:));

subplot(1,2,2);
imagesc(averageD_npcp);
title('Average NPCP Distances');
colorbar; colormap jet;set(gca, 'XtickLabel', genre(:), 'YtickLabel', genre(:));

saveImage('average.png', num2str(gam));

end;
```

Each γ value was evaluated from 10 to 100 in increments of 10, as seen in workspace_b.m.

Initially, the graphs were compared visually, but more analysis is needed in order to determine the best γ value. Instead of analyzing each gamma value, the analysis will be done once the best gamma value is chosen.

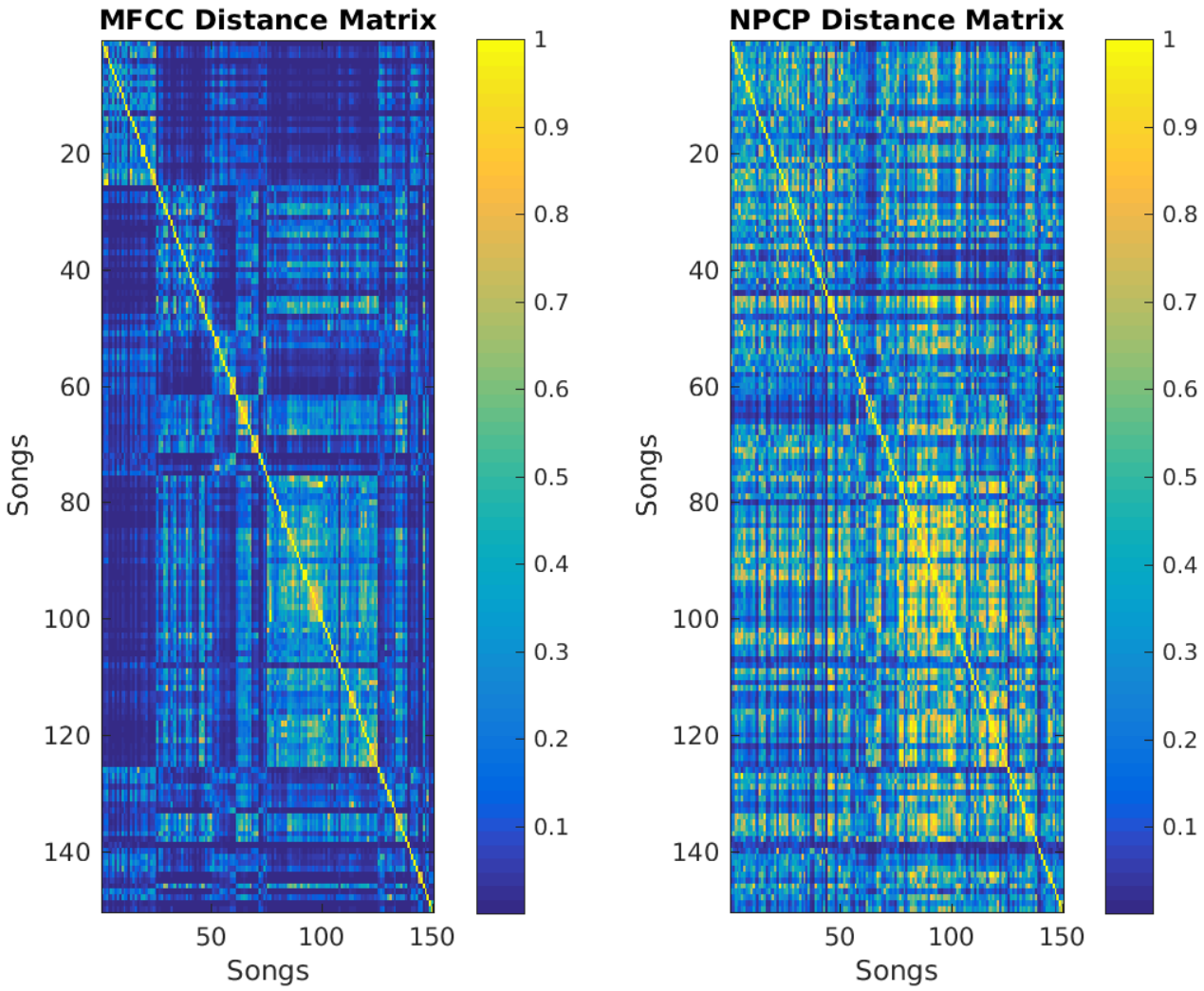

Figure 1: $\gamma = 10$

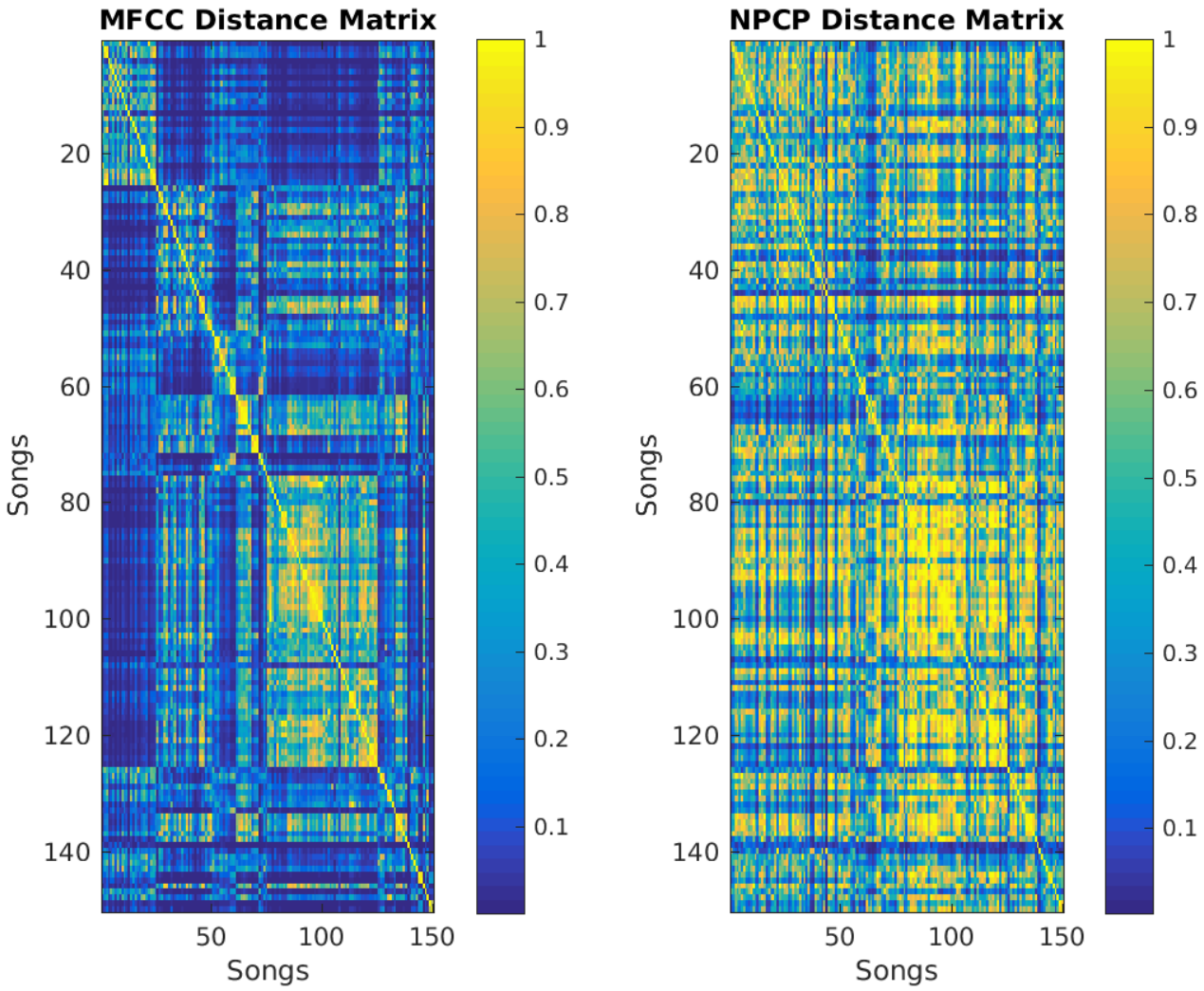

Figure 2: $\gamma = 20$

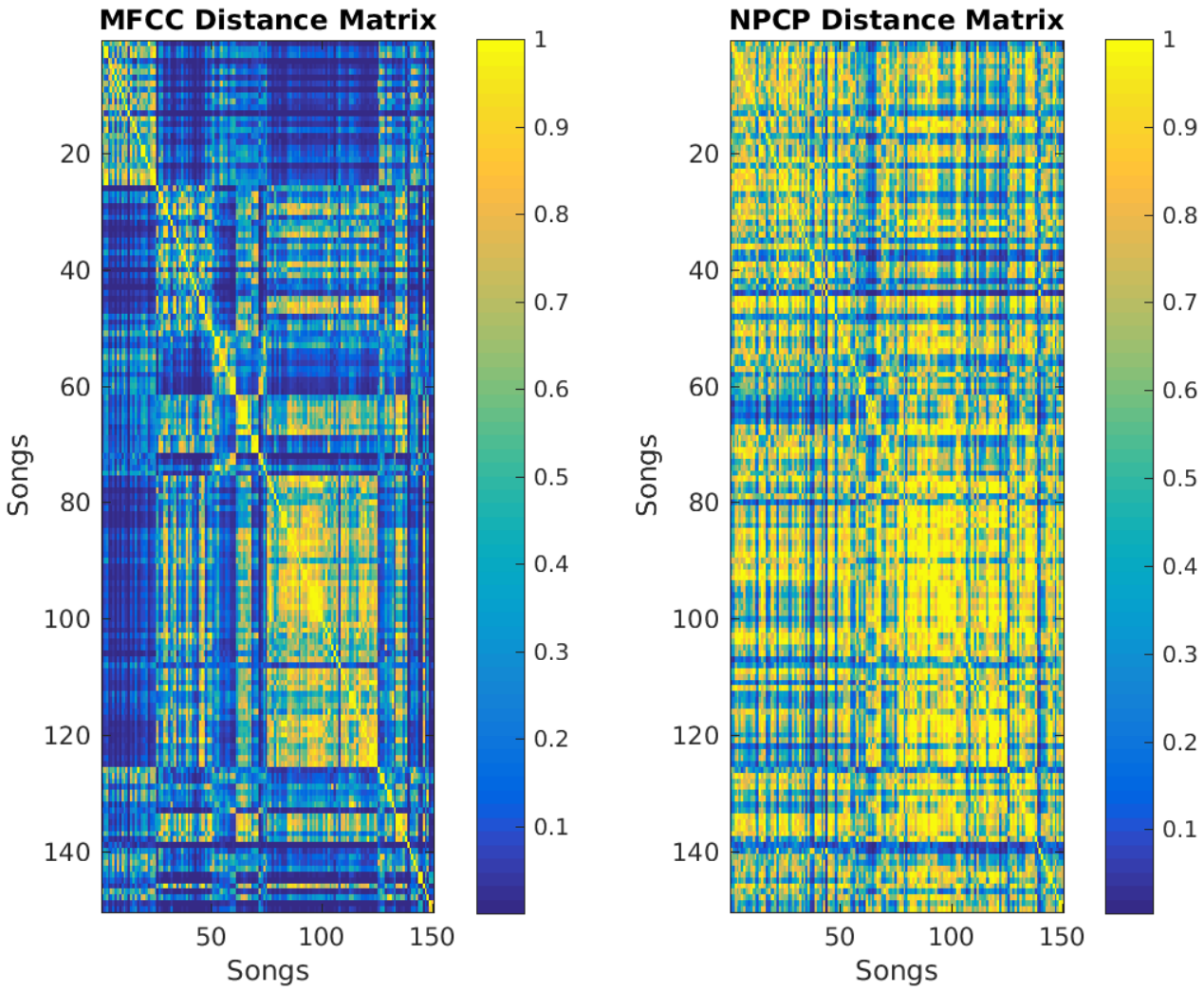

Figure 3: $\gamma = 30$

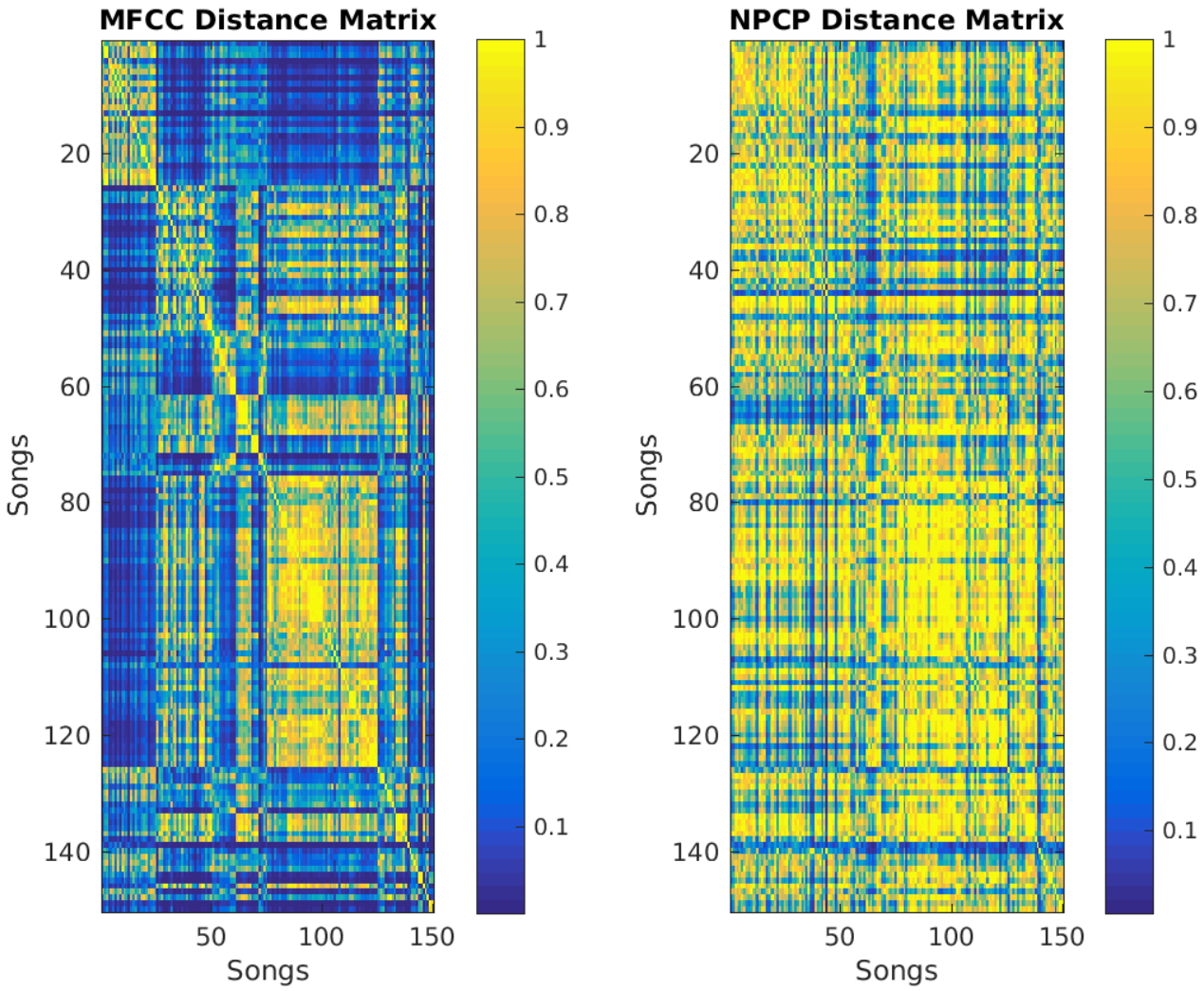

Figure 4: $\gamma = 40$

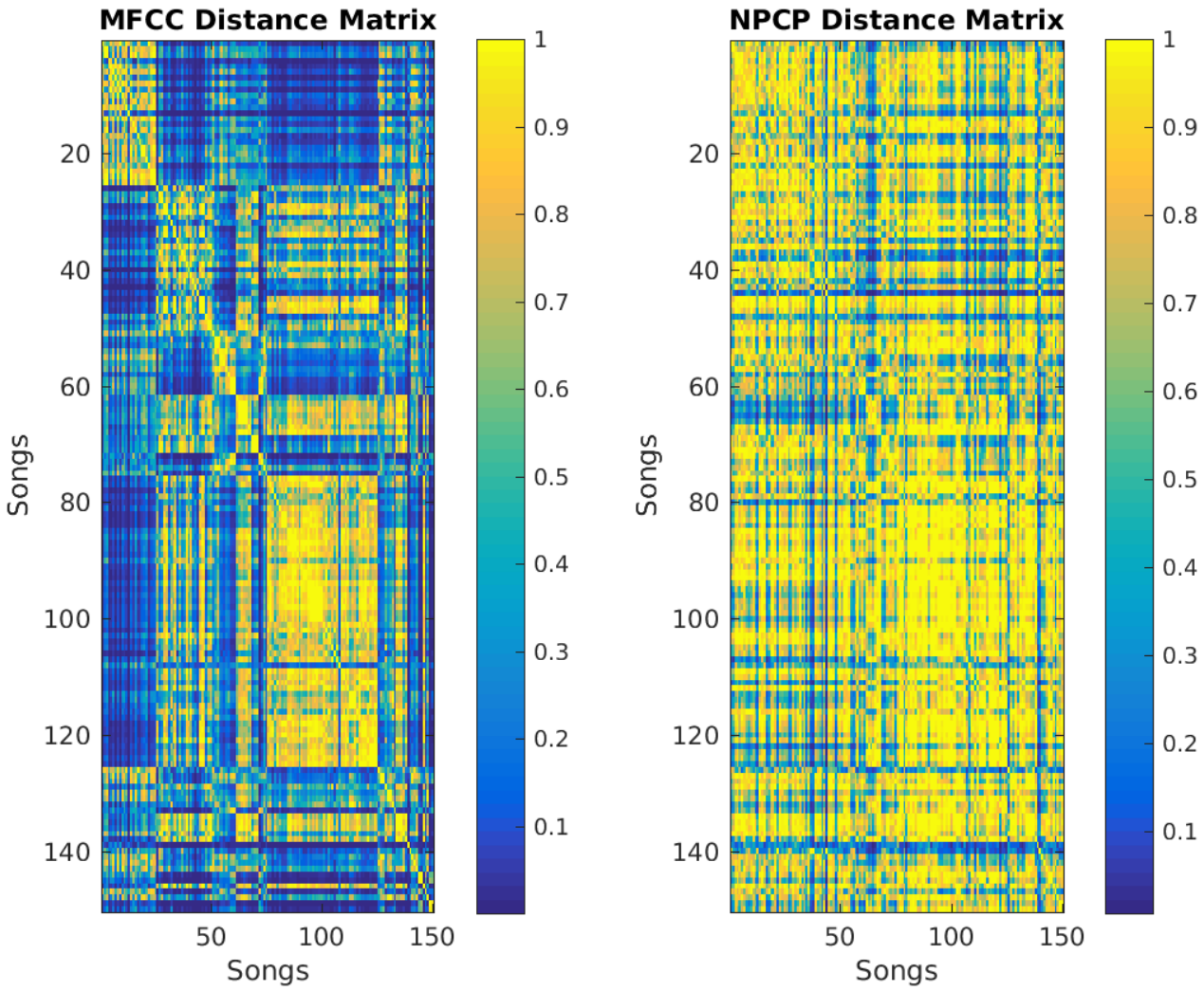

Figure 5: $\gamma = 50$

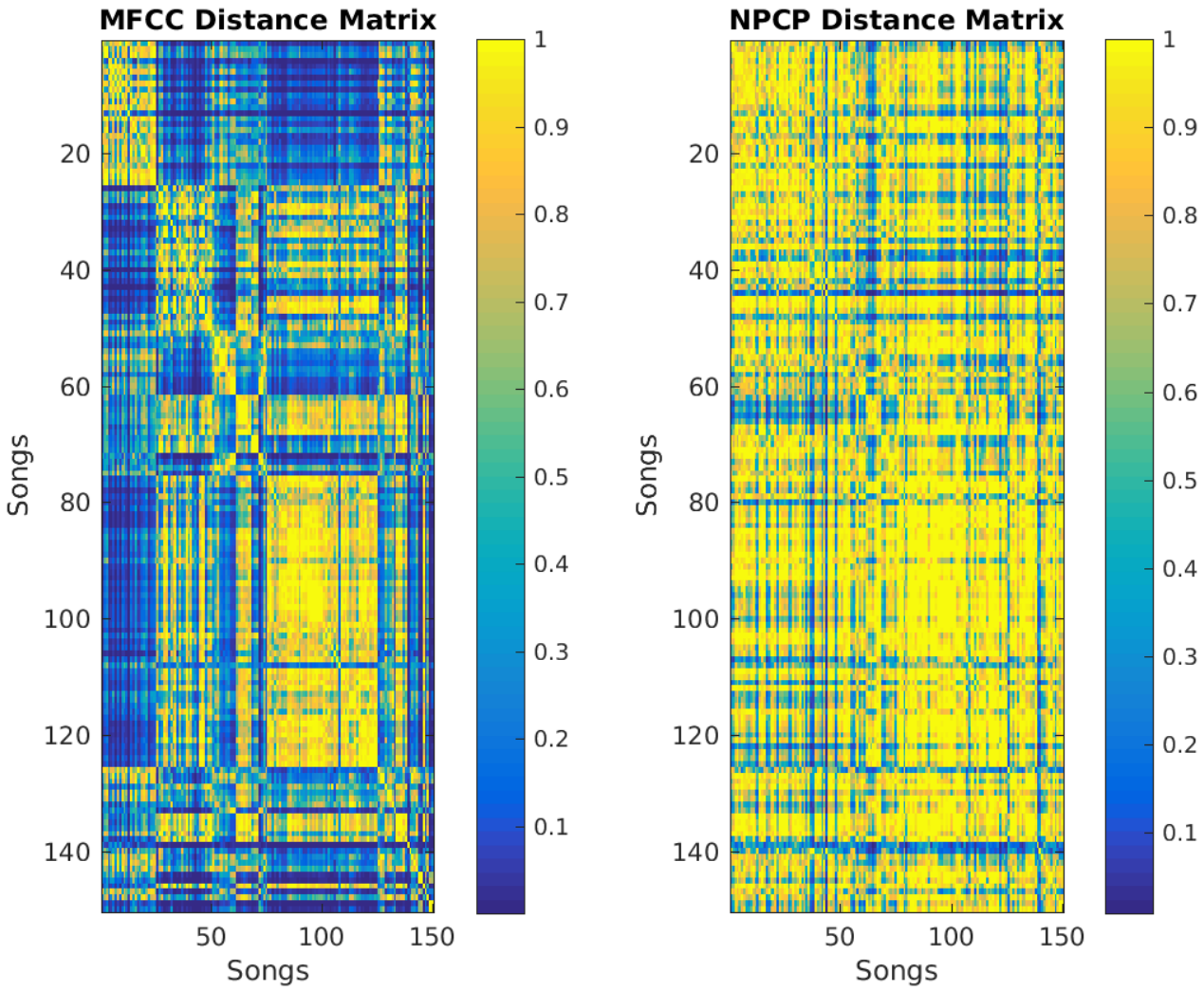

Figure 6: $\gamma = 60$

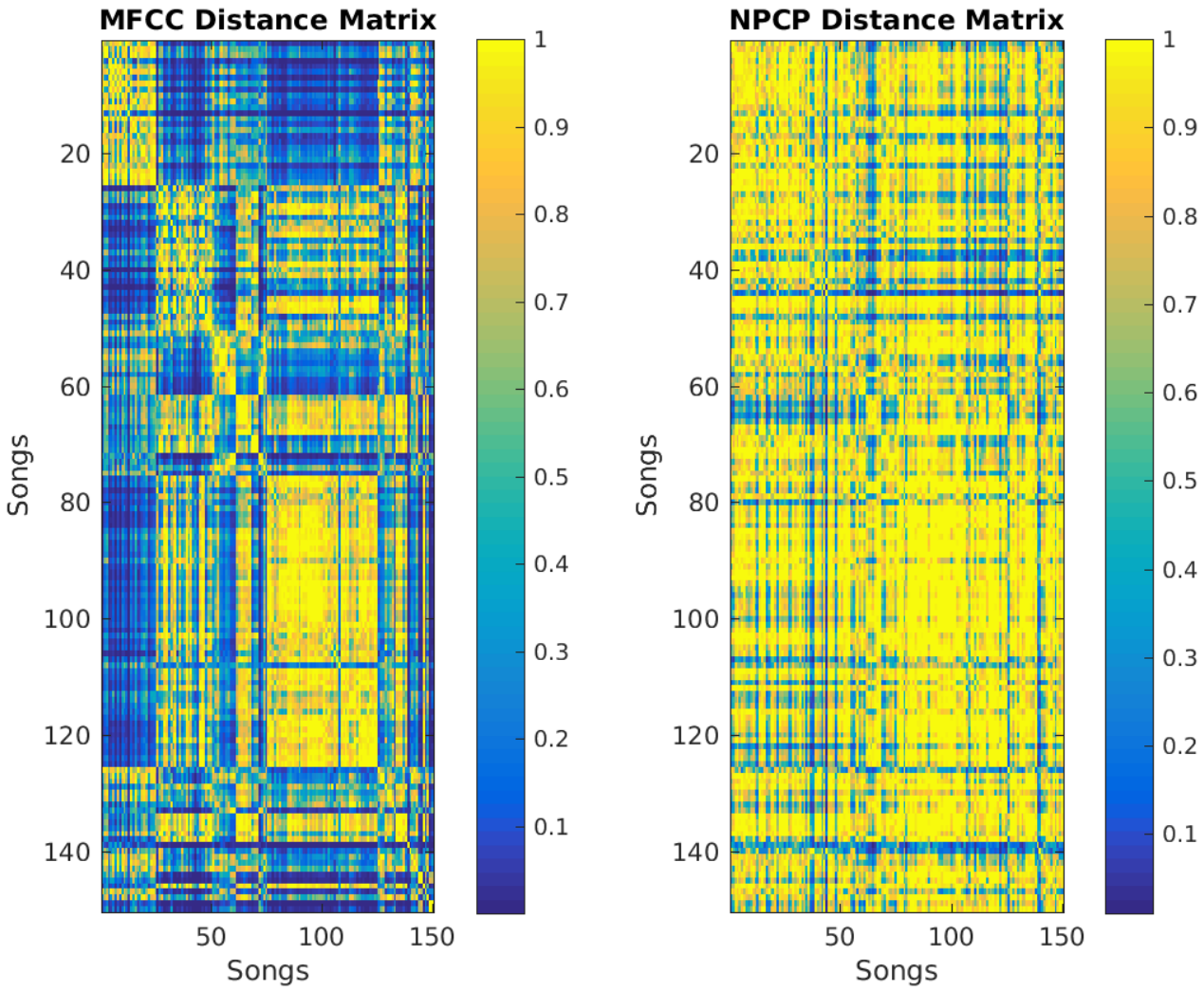

Figure 7: $\gamma = 70$

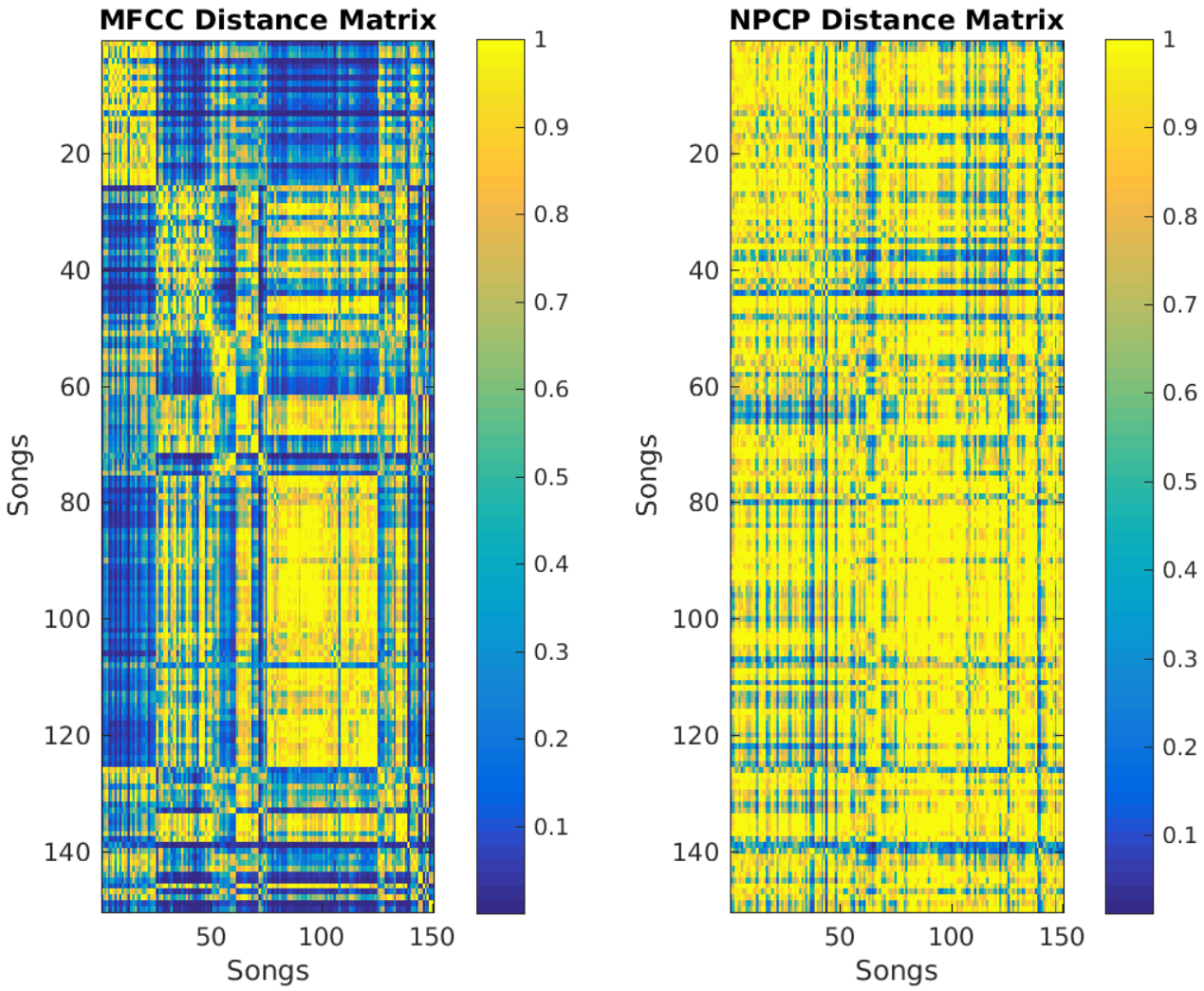

Figure 8: $\gamma = 80$

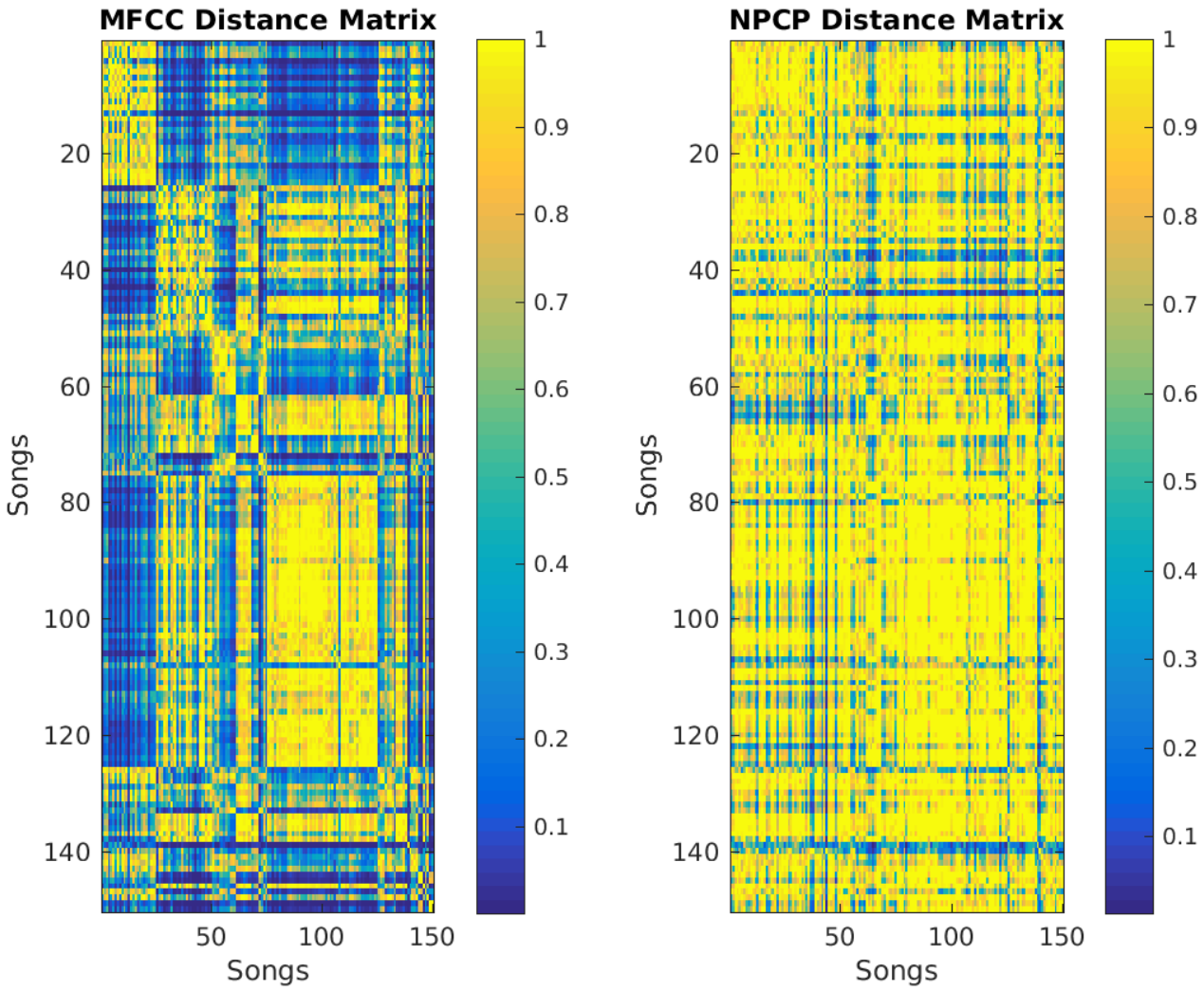

Figure 9: $\gamma = 90$

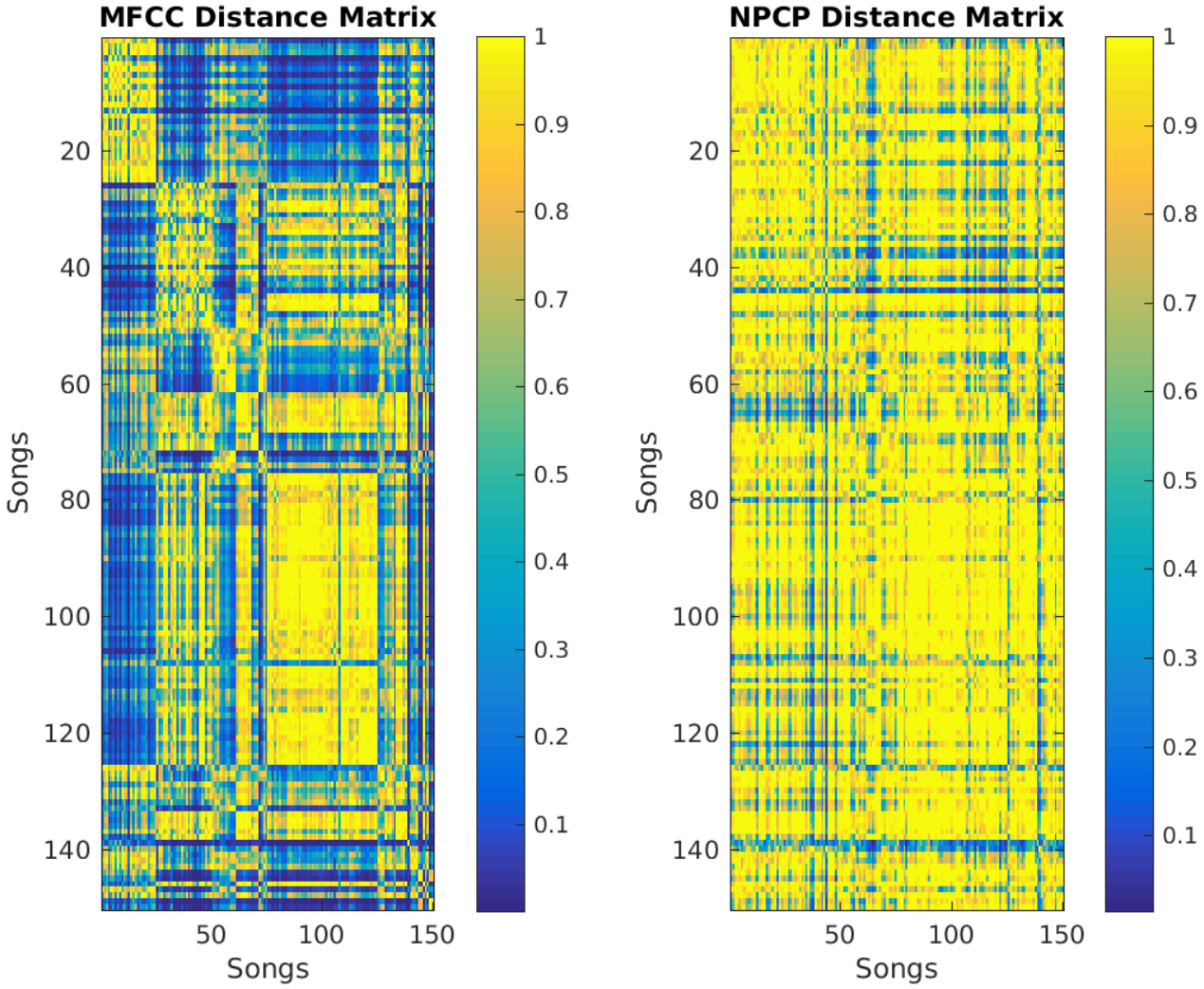

Figure 10: $\gamma = 100$

Overall, increasing the gamma values causes the magnitude of the overall distance matrix to increase. It helps differentiate the same genre from the others. Another thing to note is that the NPCP distance matrix is not as clear cut as the mfcc matrix is. As a result, it is expected to see that the average values will be just as obvious.

More specifically, throughout all the different values of γ, the classical music is very well defined within its region. There are some large values when comparing world, but considering world is a mixture of all genres, it is expected to see world have a lot of influence.

Punk and Rock, around songs 80 to 120, share a lot of similarities, as expected, considering the two genres are usually composed of the same instruments and somewhat similar styles. Its expected that these two genres may be difficult to differentiate from each other.

3 Averages

workspace_b.m covers the averaging of each genre. It simply goes through each genre and averages the genre compared to others. Overall, when comparing the mfcc and the npcp distances, it is clear that the mfcc has a more distinct differentiation between the songs. It is important to note that the average of same genres is no longer 1.

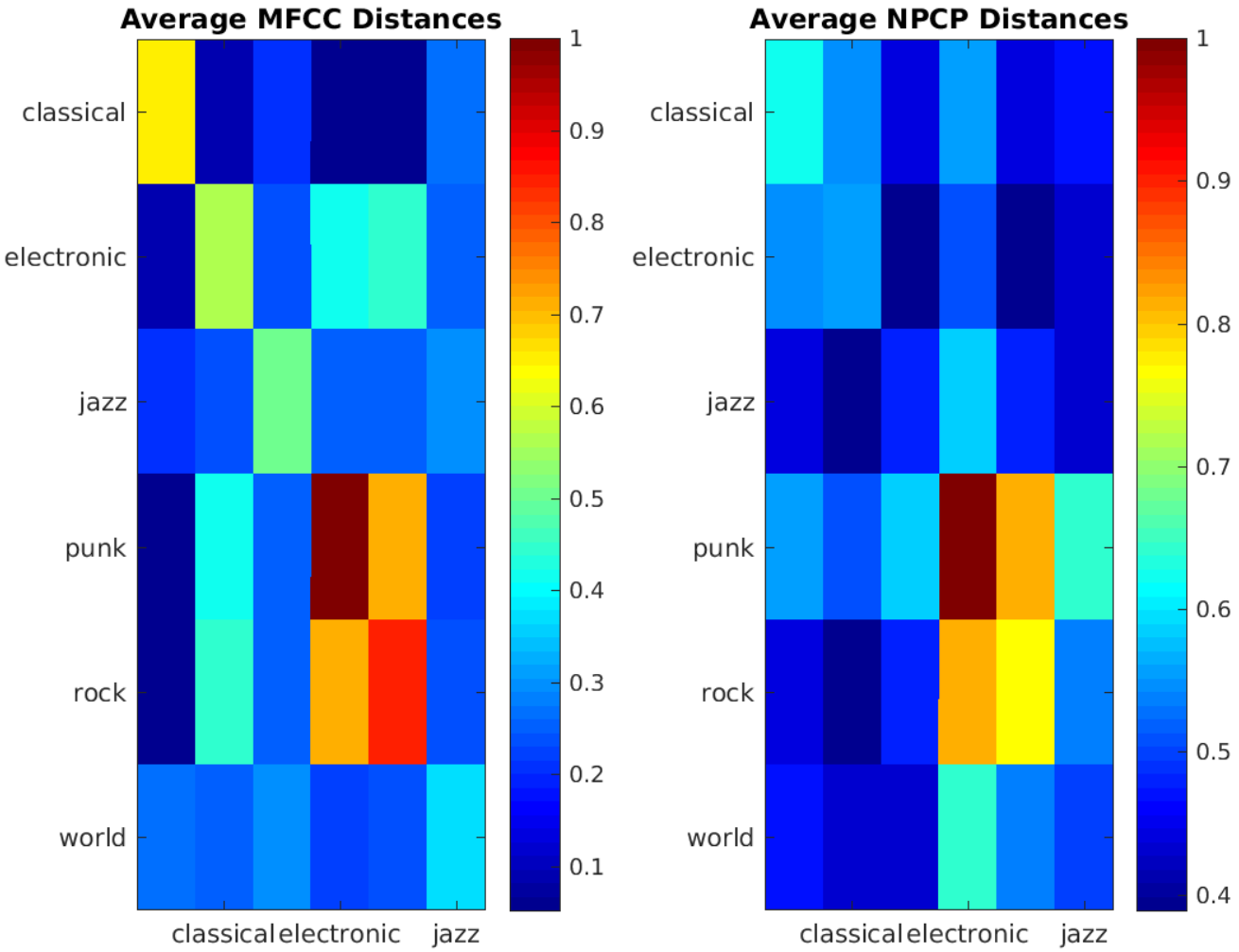

Figure 11: $\gamma = 10$

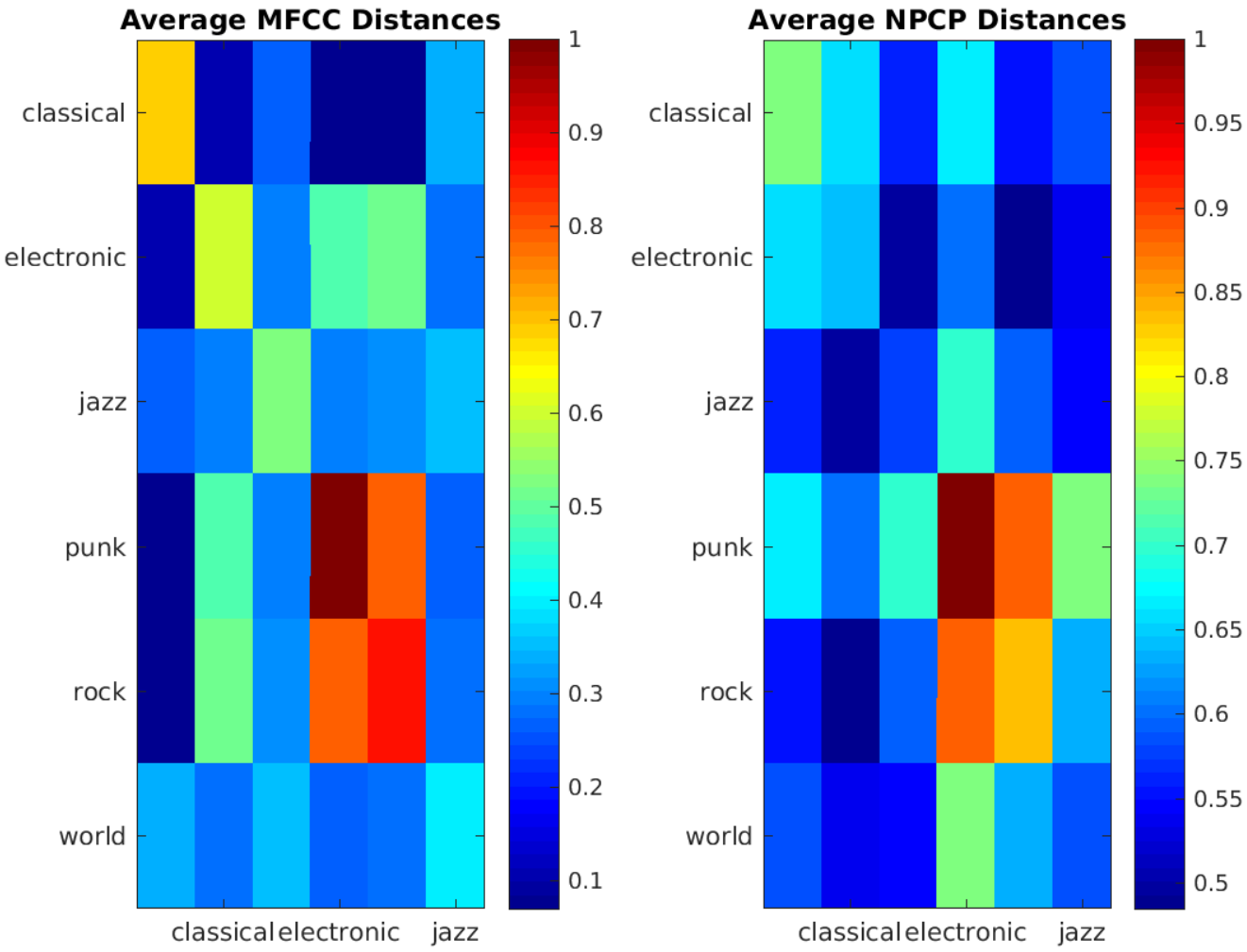

Figure 12: $\gamma = 20$

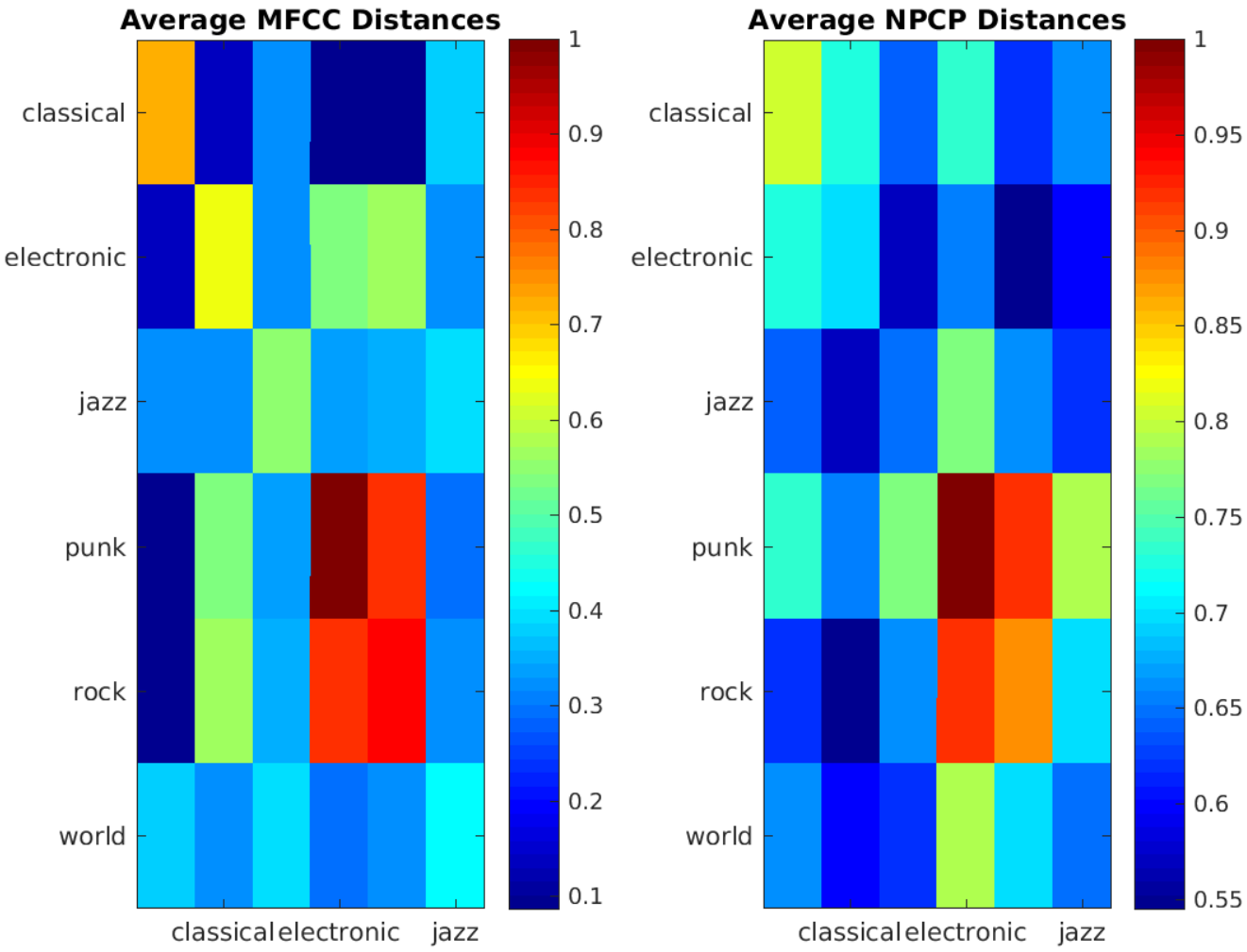

Figure 13: $\gamma = 30$

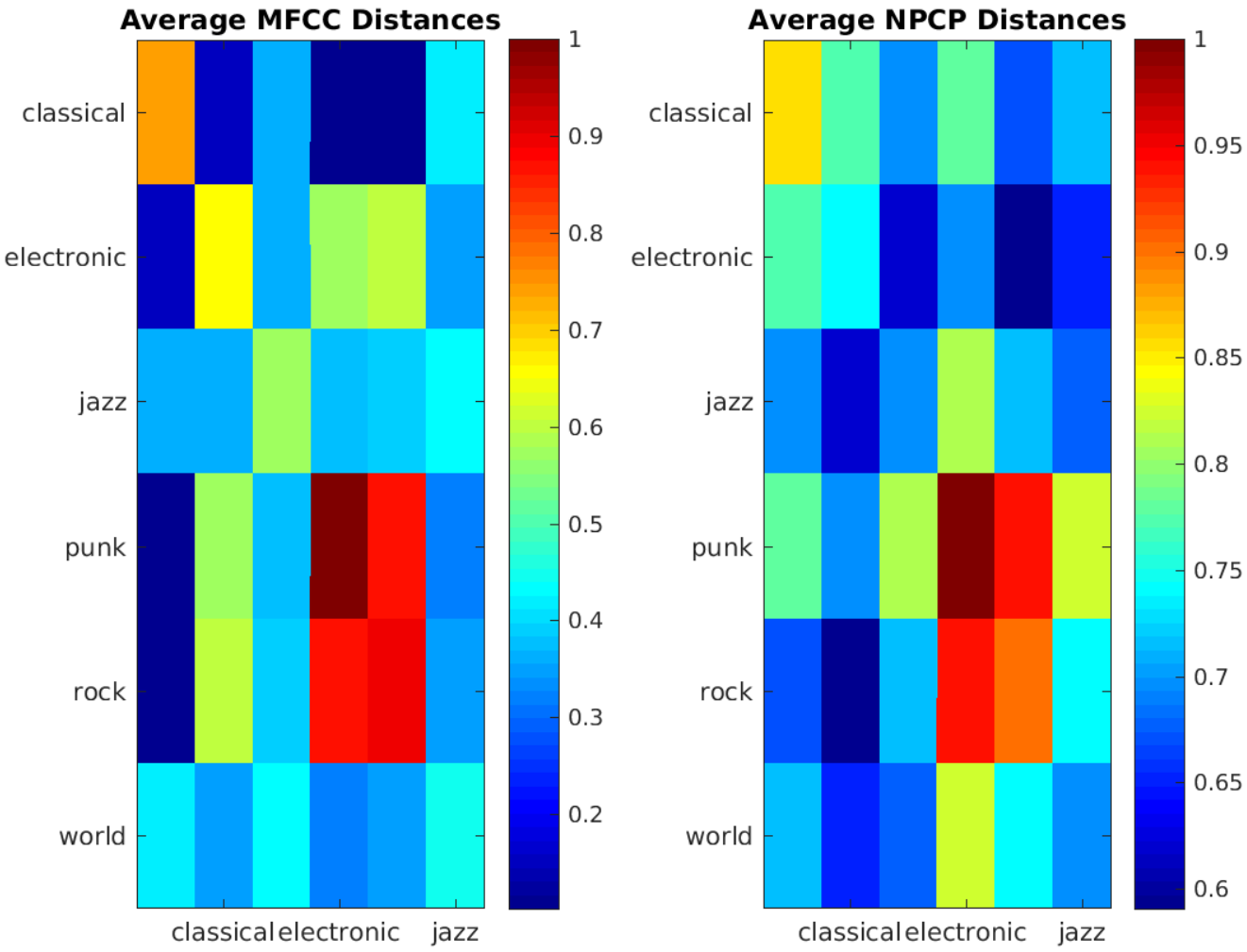

Figure 14: $\gamma = 40$

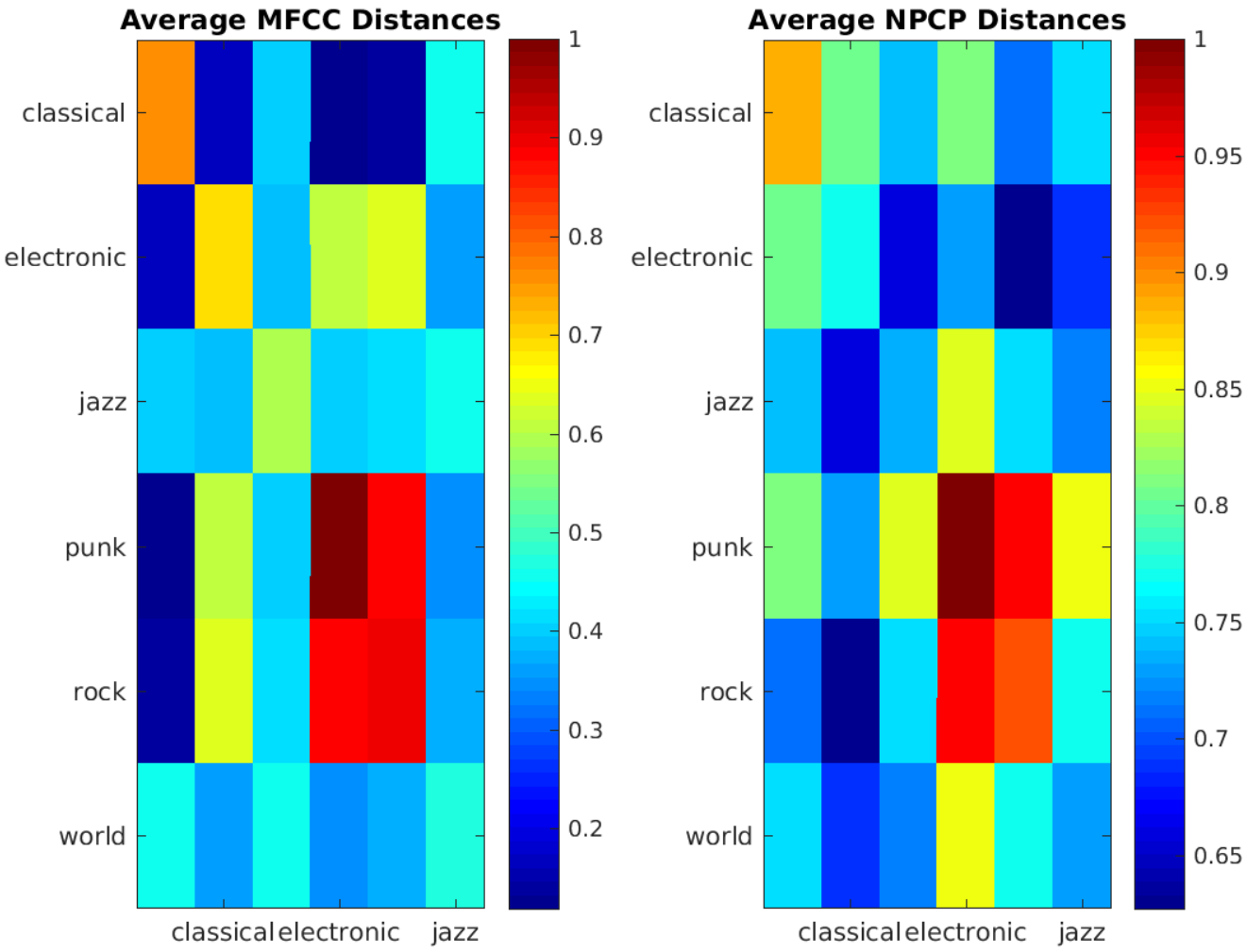

Figure 15: $\gamma = 50$

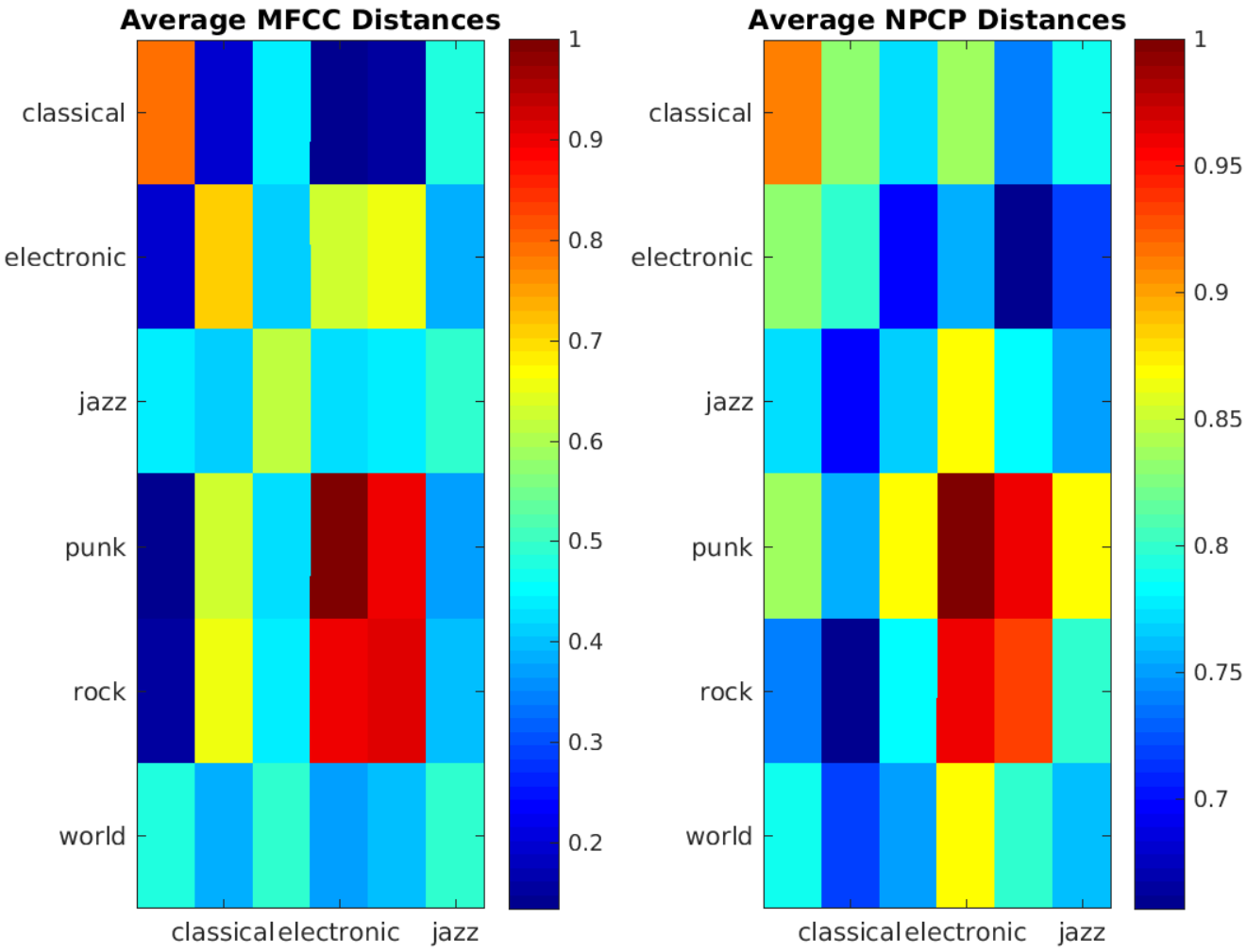

Figure 16: $\gamma = 60$

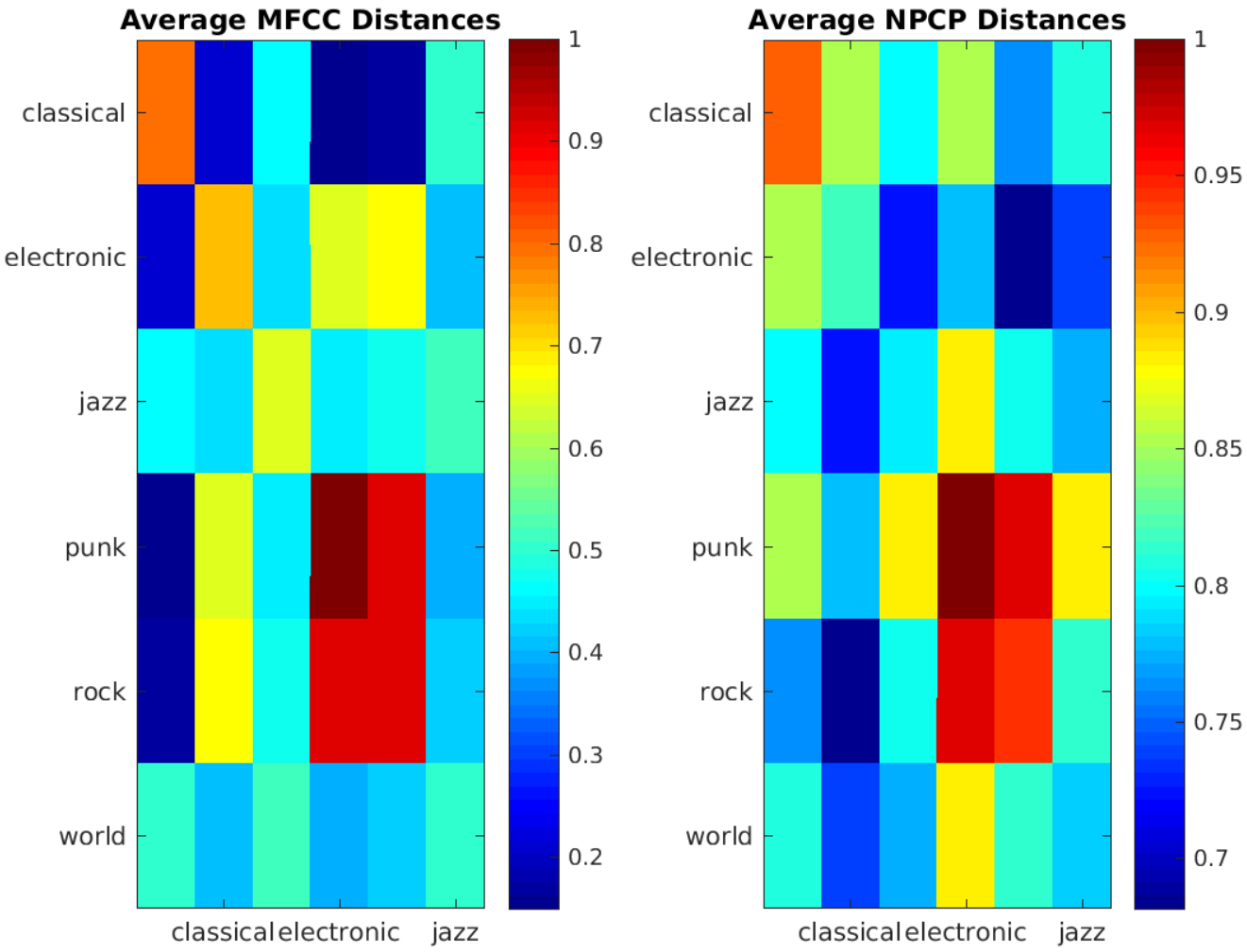

Figure 17: $\gamma = 70$

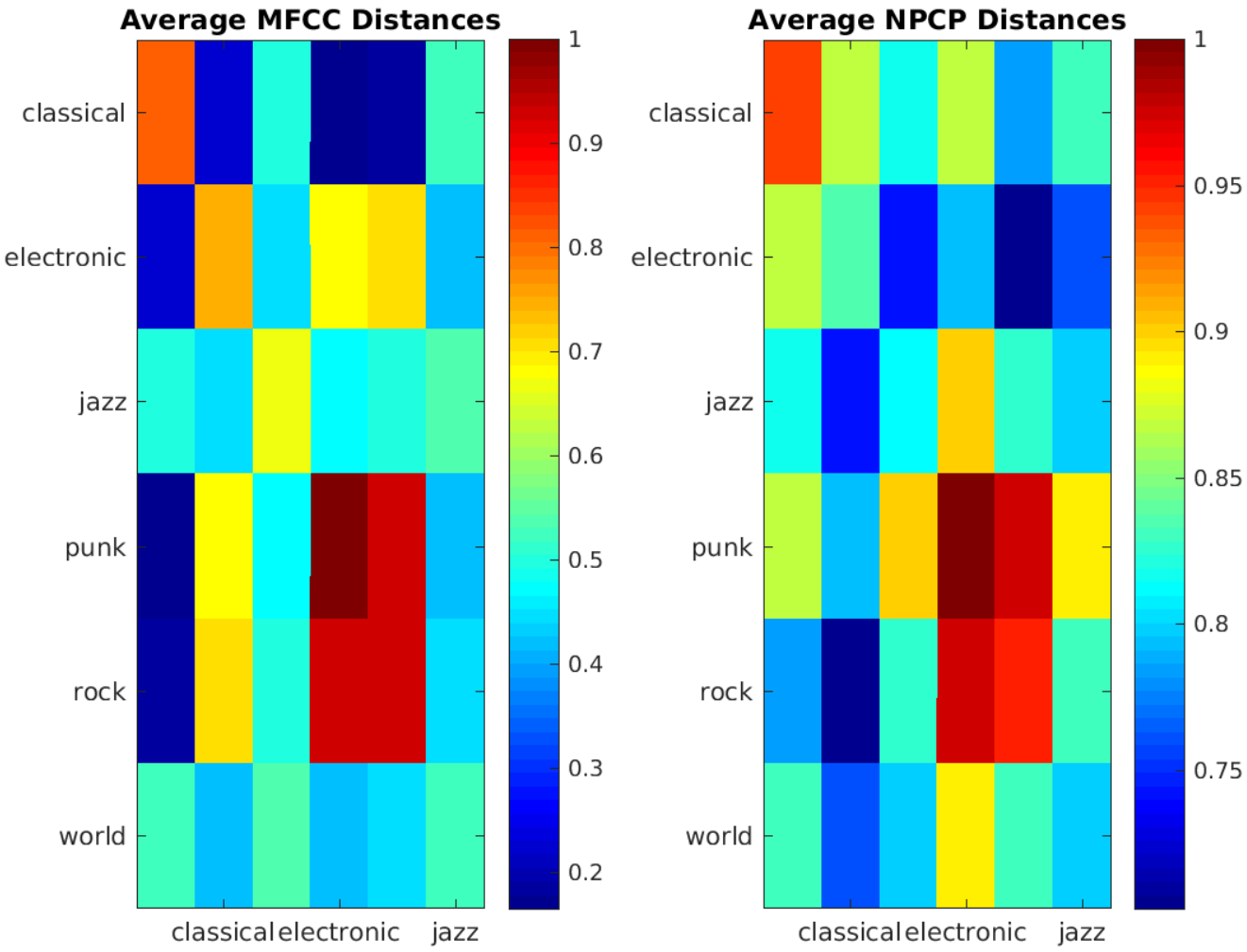

Figure 18: $\gamma = 80$

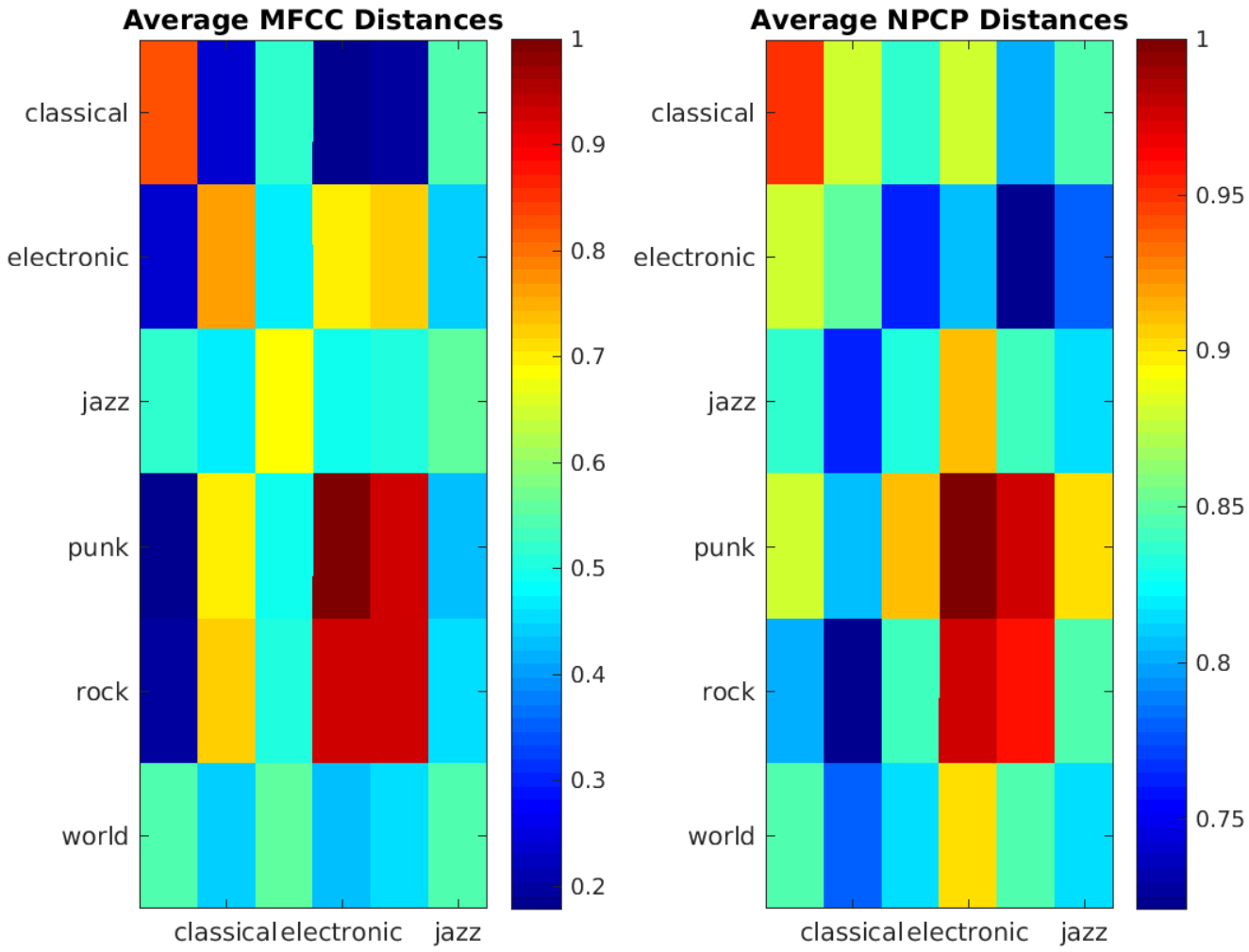

Figure 19: $\gamma = 90$

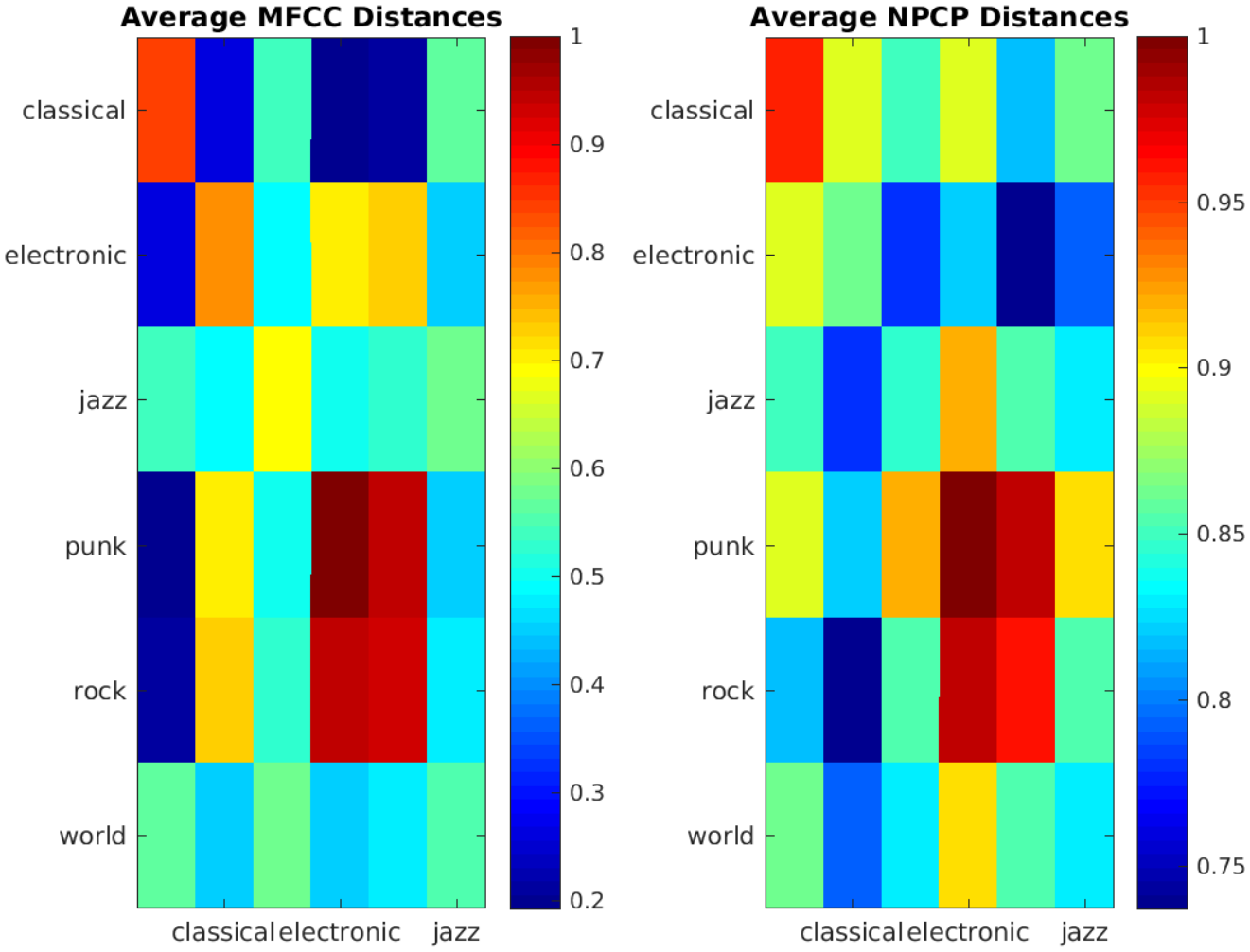

Figure 20: $\gamma = 100$

It is not efficient to determine the best γ value using eyes, so the maximum distance value, always when classifying a genre with itself, is compared to the second maximum distance. The difference was stored into a matrix with the following results. The second maximum distance is picked because the classification needs to be sensitive and needs to minimize the difference between the largest value and the second largest. Each row represents the genre, and each column represents the γ value, with increments of 10.

For the mfcc:

$$\begin{pmatrix} 0.148 & 0.203 & 0.236 & 0.256 & 0.268 & 0.275 & 0.279 & 0.28 & 0.28 & 0.279 \\ 0.0437 & 0.0427 & 0.0406 & 0.0387 & 0.0374 & 0.0365 & 0.036 & 0.0357 & 0.0356 & 0.0355 \\ 0.0614 & 0.0616 & 0.0589 & 0.0566 & 0.0552 & 0.0546 & 0.0546 & 0.055 & 0.0557 & 0.0565 \\ 0.131 & 0.138 & 0.128 & 0.115 & 0.102 & 0.0918 & 0.0829 & 0.0756 & 0.0696 & 0.0646 \\ 0.0478 & 0.0346 & 0.0214 & 0.0109 & 0.00288 & 0.00319 & 0.0078 & 0.0113 & 0.0141 & 0.0162 \\ 0.0432 & 0.0469 & 0.047 & 0.046 & 0.0447 & 0.043 & 0.0411 & 0.038 & 0.0352 & 0.0325 \end{pmatrix}$$

Initially looking at classical, or the first row, it is evident that the lowest difference is at $\gamma = 10$, or the first column. As one increase the γ value, it is increasing the difference values, however, this is a unique characteristic of classical. When looking at electronic right below it, the lowest

value is actually produced when $\gamma = 100$, but once again, it is different for Jazz with $\gamma = 60$ or 70 being the most optimal.

For the npcp:

$$\begin{pmatrix} 0.0232 & 0.0133 & 0.00923 & 0.00812 & 0.00842 & 0.00931 & 0.0104 & 0.0115 & 0.0124 & 0.0132 \\ 0.00412 & 0.00773 & 0.0158 & 0.0227 & 0.0283 & 0.0328 & 0.0365 & 0.0395 & 0.0419 & 0.0429 \\ 0.0371 & 0.0772 & 0.0988 & 0.11 & 0.114 & 0.11 & 0.104 & 0.099 & 0.0935 & 0.0882 \\ 0.243 & 0.235 & 0.199 & 0.165 & 0.137 & 0.114 & 0.0952 & 0.0803 & 0.0683 & 0.0585 \\ 0.0273 & 0.0576 & 0.0712 & 0.0767 & 0.078 & 0.077 & 0.0748 & 0.0718 & 0.0686 & 0.0652 \\ 0.0499 & 0.0691 & 0.0759 & 0.0772 & 0.0761 & 0.0738 & 0.0709 & 0.0678 & 0.0646 & 0.0614 \end{pmatrix}$$

Once again, it is not as clear which γ value is efficient. Overall, they are not necessarily consistent. As a result, the average of each column is calculated and the largest average is used because the larger average represents a larger difference.

For mfcc, the value is $\gamma = 30$, with the average difference being 0.0886.
For npcp, the value is $\gamma = 30$, with the average difference being 0.0784.

This value of γ will be used for the rest of the lab.

3.1 Analysis

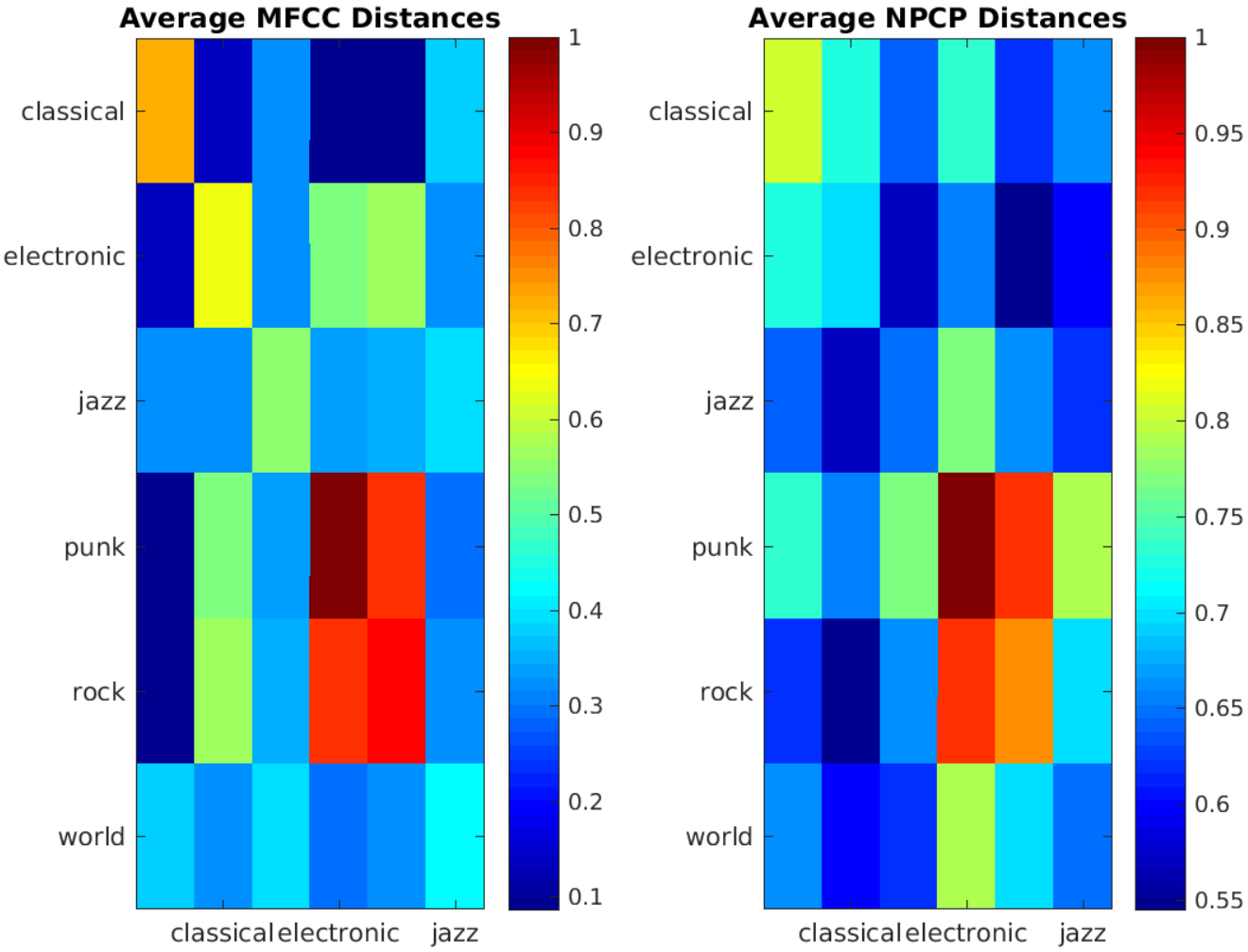

Figure 21: $\gamma = 30$

Its very interesting to note that both values point to the same gamma value to be the most efficient value to be used, but the mfcc has a larger average difference than npcp, and with this, the value of the most efficient γ is 30. The mfcc values also give a better average distance than the npcp values as evident in the magnitudes of the diagonal in comparison to the other genres. In the npcp values, it is evident there are several misclassification such as jazz and world.

In the figure, the average mfcc distance shows that for classical music, it is very obvious when a song is classical. However, for songs like jazz and world, it is very difficult to differentiate between the songs. Similar genres, like punk and rock are very similar, and thus those square are very large in value.

Electronic is another story as it is seen that it is very similar to punk and rock. This is another problematic genre.

Overall, classical, jazz, and punk will have the best prediction due to the large distances between

itself and the other songs.

Rock, world, jazz, and possibly electronic will have issues for prediction. Rock might have a lot of misclassification to punk, and world is not necessarily that different from any of the other genres.

3.2 Histogram

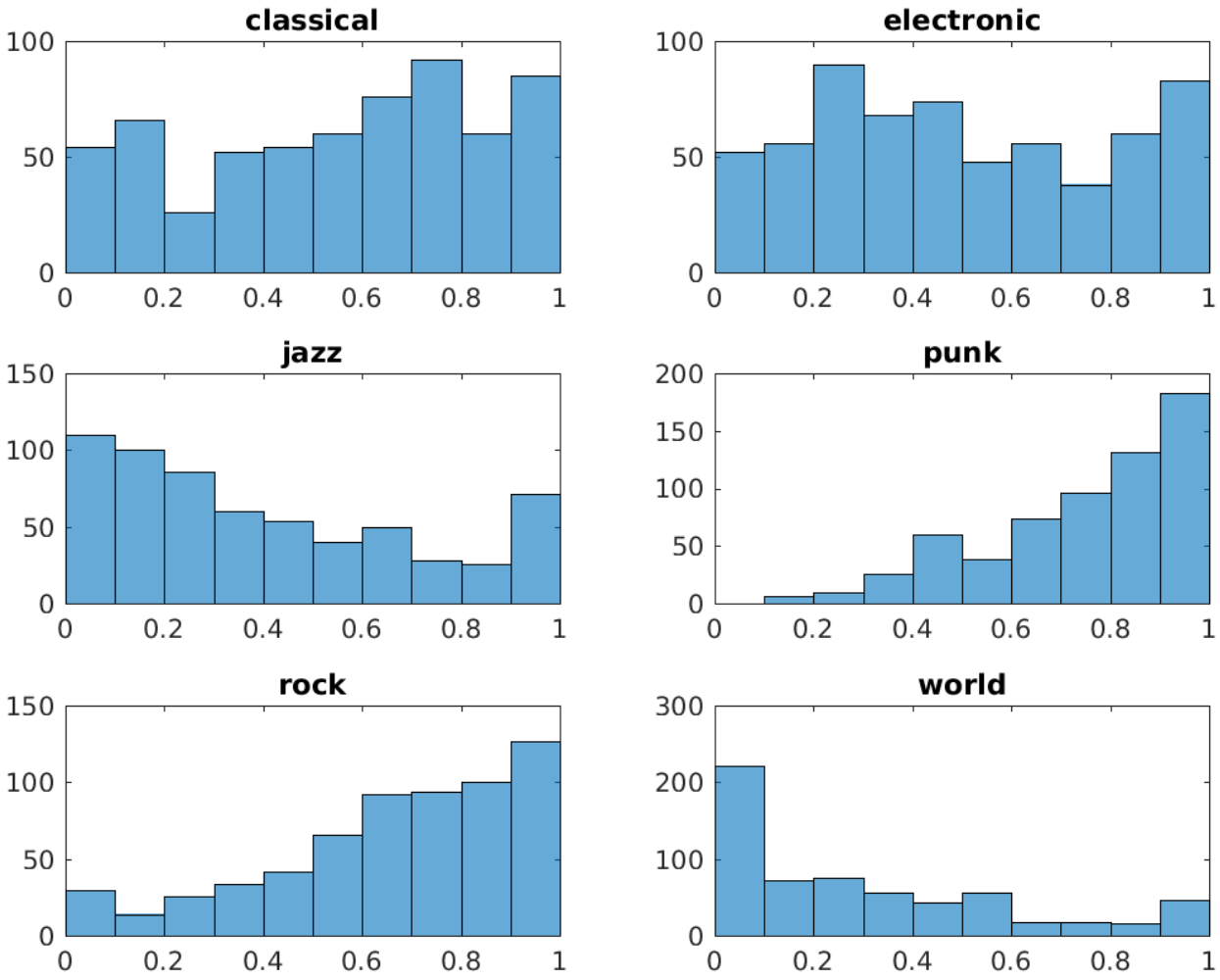

Figure 22: Histogram of the Distances

The histogram demonstrates the spread of distances. As evident in world histogram, it is very hard to classify world as most of the histogram is very low. Punk and rock are expected to have high recognizability since they have very large values at 1. Jazz will be difficult, but not as difficult as world, and electronic and classical will be fairly simple to classify. This histogram is essentially a prediction of what the classifier should be able to predict.

3.3 Song Length

For the next section, the song length is differed and the resulting averages are analyzed. The songs are analyzed at 30, 60, 120, and 240 seconds. The code used is similar to workspace_b.m. The code is to run workspace_a.m with the different song lengths, and then run workspace_b.m to generate new average plots.

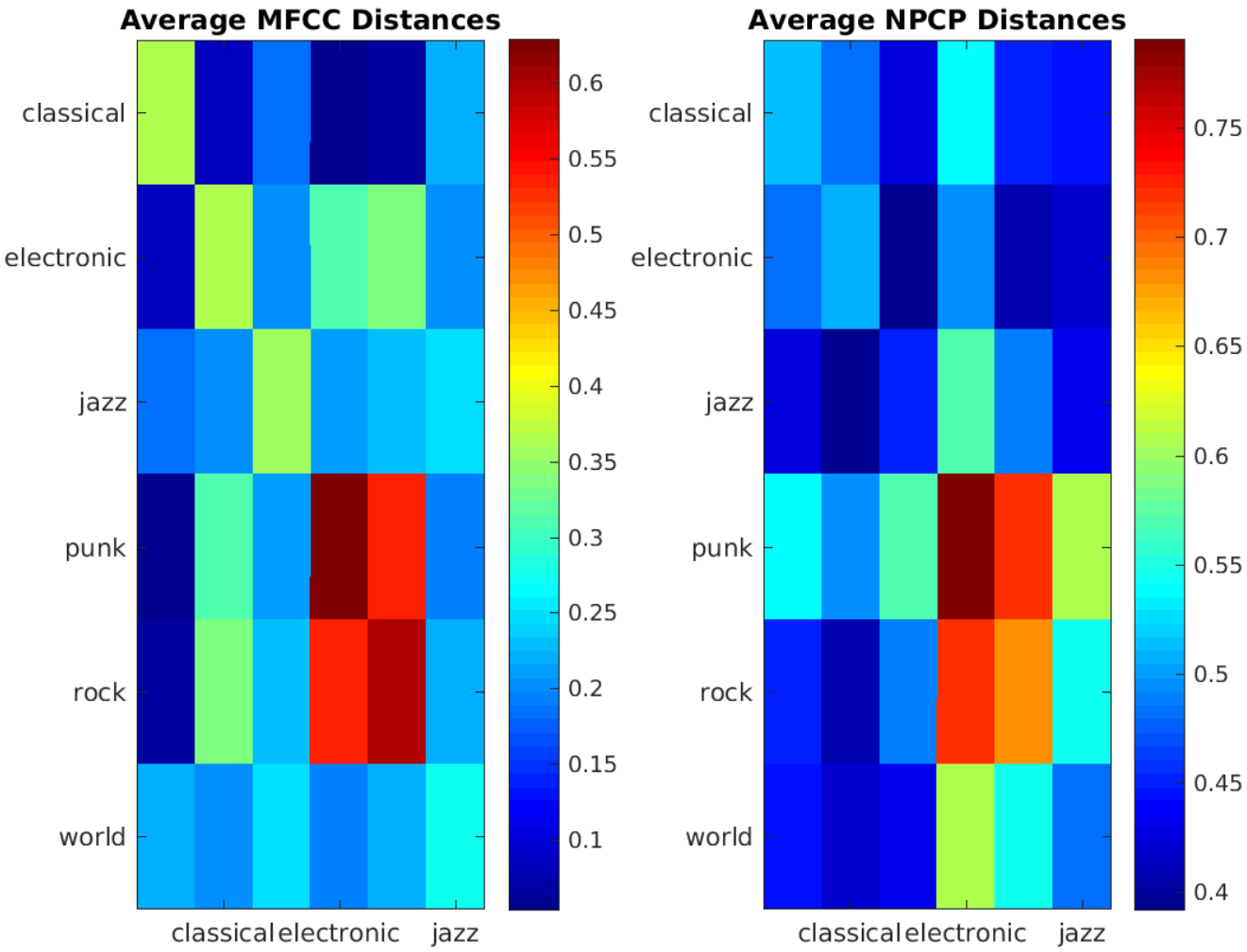

Figure 23: Song Length = 30

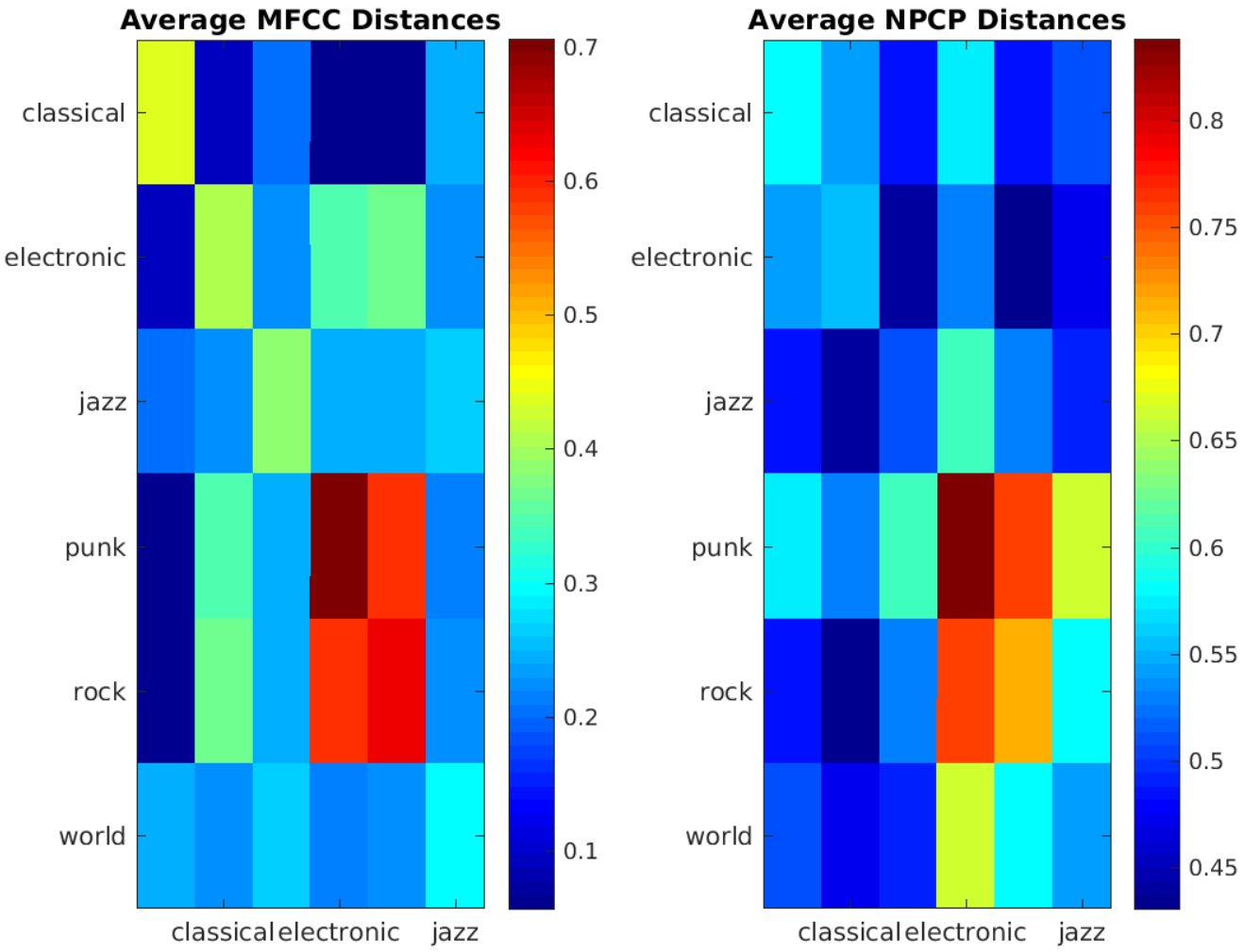

Figure 24: Song Length = 60

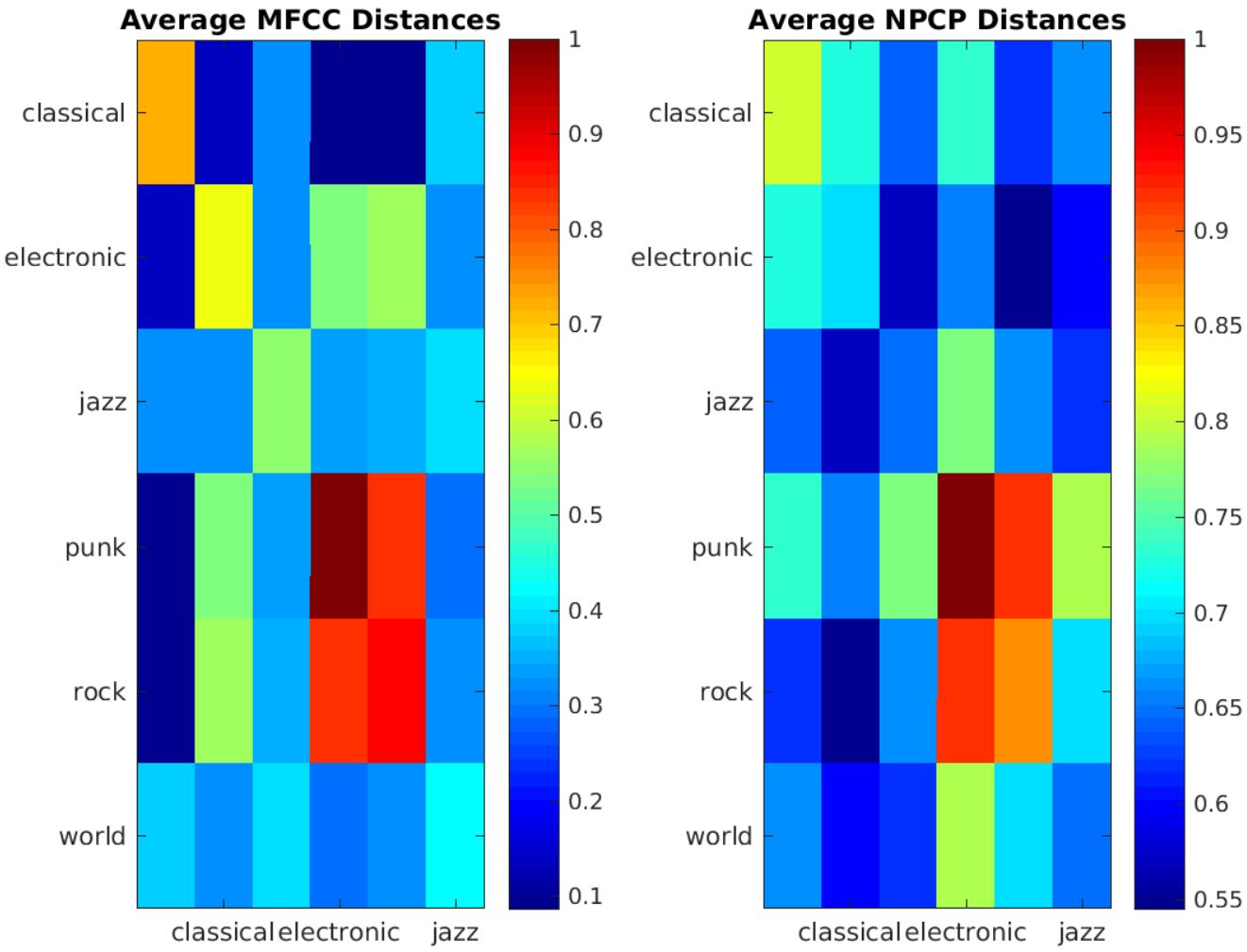

Figure 25: Song Length = 120

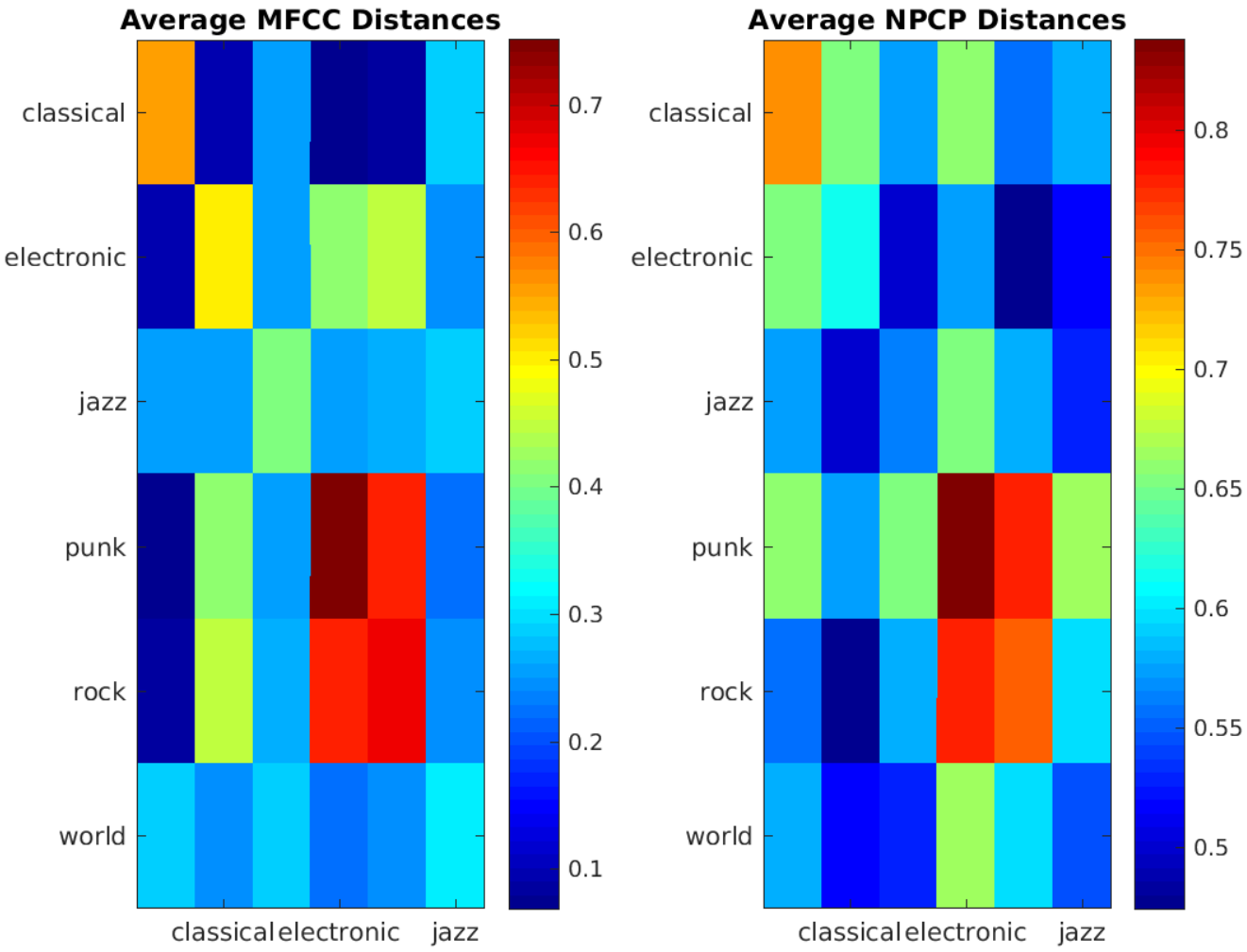

Figure 26: Song Length = 240

Overall, it seems that the length of song helps with the distance. This would make more sense because you obtain more song characteristics, but this can also be counterproductive because more characteristics may be too broad and not bring in enough specific details.

4 Classification

In order to test classification, there are two methods that are used: SVM and K Nearest Neighbor. The lab states that the SVM can be used to assist the original classifier, however, this is not necessarily possible to combine these two classifiers in Matlab. Instead, I will analyze the two different classifier methods.

```
1 %workspace to work with different classifier
2
3 %Genre names
4 genres = cell(1,150);
5 genres(1:25) = {'classical'};
```

```
genres(26:50) = {'electronic'};
genres(51:75) = {'jazz'};
genres(76:100) = {'punk'};
genres(101:125) = {'rock'};
genres(126:150) = {'world'};

%Pre Generation of confusion matrices
confusion = zeros(6, 6, 10);
confusionSVM = zeros(6, 6, 10);
meanConfusion =  zeros(6,6);
meanConfusionSVM = zeros(6,6);
stdConfusion =  zeros(6,6);
stdConfusionSVM = zeros(6,6);

for n = 1:10
    %Confusion matrix
    confusion(:,:,n) = genreClassifier(D_mfcc, genres, 5, 'knn');
    %Confusion matrix for SVM
    confusionSVM(:,:,n) = genreClassifier(D_mfcc, genres, 5, 'svm');
end

for i = 1:6
    for j = 1:6
        meanConfusion(i,j) =  mean(confusion(i,j,:));
        meanConfusionSVM(i,j) = mean(confusionSVM(i,j,:));
        stdConfusion(i,j) =  std(confusion(i,j,:));
        stdConfusionSVM(i,j) = std(confusionSVM(i,j,:));
    end
end
```

```
function [ confusion ] = genreClassifier(distance, genres, times, fitType)
%Genre Classifier

confusion = zeros(6, 6);

%For the number of times to do the k fold
for i = 1:times
        %Cross Valid by removing 5 entries per class
        [train, test] = crossvalind('LeaveMOut',genres, 5);

        %KNN or SVM
        if(strcmp(fitType, 'knn'))
            Mdl = fitcknn(distance(train,train), genres(train));
        elseif(strcmp(fitType, 'svm'))
             Mdl = fitcecoc(distance(train,train), genres(train));
        end

        %Predict using the test class
        class = predict(Mdl, distance(test, train));

        %Takes the predicted classes and organizes them
        L = 1;
        for j = 1:6
            for k = 1:5
                switch char(class(L))
                    case {'classical'}
                        input = 1;
                    case {'electronic'}
                        input = 2;
                    case {'jazz'}
                        input = 3;
                    case {'punk'}
```

```
                        input = 4;
                    case {'rock'}
                        input = 5;
                    case {'world'}
                        input = 6;
                end

                confusion(j, input) = confusion(j, input) + 1;
                L = L + 1;
            end
        end
end
```

4.1 Confusion Matrices

K Nearest Neighbor Averages

$$\begin{pmatrix} 22.0 & 0 & 0.1 & 0 & 0 & 2.5 \\ 0 & 17.0 & 2.4 & 1.5 & 1.6 & 2.4 \\ 1.4 & 1.8 & 18.0 & 2.0 & 0 & 2.1 \\ 0 & 1.5 & 0.9 & 18.0 & 3.1 & 1.5 \\ 0 & 5.6 & 0.1 & 3.5 & 14.0 & 1.6 \\ 4.4 & 3.5 & 3.2 & 0.7 & 0.9 & 12.0 \end{pmatrix}$$

SVM Averages

$$\begin{pmatrix} 22.0 & 0 & 0 & 0 & 0 & 3.3 \\ 0 & 17.0 & 2.4 & 0.5 & 3.7 & 1.8 \\ 0.1 & 2.3 & 14.0 & 2.0 & 0.1 & 6.2 \\ 0 & 2.4 & 0 & 21.0 & 1.8 & 0 \\ 0 & 4.2 & 0.4 & 4.2 & 14.0 & 2.6 \\ 4.5 & 3.6 & 3.4 & 0.9 & 0.8 & 12.0 \end{pmatrix}$$

K Nearest Neighbor Standard Deviation

$$\begin{pmatrix} 1.5 & 0 & 1.2 & 0 & 0 & 1.2 \\ 0 & 2.3 & 1.6 & 1.3 & 0.85 & 1.4 \\ 0.84 & 1.5 & 2.1 & 0.95 & 0.32 & 2.0 \\ 0 & 1.3 & 0.7 & 2.0 & 1.2 & 0.63 \\ 0 & 1.4 & 0.32 & 1.9 & 1.5 & 1.6 \\ 2.3 & 1.9 & 1.8 & 0.32 & 0.79 & 2.5 \end{pmatrix}$$

SVM Standard Deviation

$$\begin{pmatrix} 3.1 & 0 & 0.42 & 0 & 0 & 2.9 \\ 0 & 1.7 & 0.99 & 1.3 & 2.1 & 0.97 \\ 1.0 & 1.6 & 1.6 & 1.4 & 0.67 & 1.5 \\ 0 & 0.84 & 0.95 & 1.9 & 1.3 & 0 \\ 0 & 2.3 & 0.79 & 1.6 & 1.9 & 0.95 \\ 2.0 & 1.0 & 1.3 & 0.63 & 0.74 & 2.1 \end{pmatrix}$$

The values of the resulting confusion matrices demonstrates that the K nearest neighbor is the most efficient classifier overall. The values when compared to the histogram is as expected. The average mfcc images demonstrates how there are a lot of misclassification in the genres that were expected to have a lot of inconsistencies. The overall perentage of the KNN method is only 67.8% and for SVM it is 65.8%. There are a lot of misclassification between all the genres. This is strongly influenced by genres like world. For each genre, the accuracy is: 89%, 68.4%, 70.8%, 72%, 56.8%, and 49.2%. World, rock, and electronic give the classifier a much more difficult time to properly classify.

5 Conclusion

The project was a great learning process and demonstrated different DSP techniques that can be applied in order to analyze audio. Just this single category demonstrates the power of filters

and their use in the audio world.

It would be interesting to try and do a theoretical part of designing filters on our own and learning other types of techniques that are used to design these filter banks.

It would be interesting to be able to use two different representations to try and create a better distance matrix. The chroma was very lacking, as evident in the analysis of the distance matrices. Overall, this means that the chroma has some limitations in being able to represent different songs, most likely due to its limitations in representing different audio frequency ranges.

The classifier is a very difficult part of the project. In order to properly classify, it would require a lot more work to properly train and get working.