# Eigenfaces and Music Genre Identification

Andrei Karavanov

Friday, March 8, 2019

**Abstract**

In the first half of this project I will talk about usage of SVD to compress the data. In the secod half we will create an algorithm that will be able to predict different songs.

# 1 Introduction and Overview

## 1.1 Yale Faces B

For this part of our project we were given two data sets of pictures of peoples' faces. In the first data set all pictures were cropped, such that the faces on those pictures were centered in the middle. The second data set also had pictures of the faces, but in this case those pictures were not cropped and faces were not centered in the middle. Our goal is to perform a Singular Value Decomposition analysis on these data sets and compare results of this analysis between them.

## 1.2 Music Classification

For this part of our project we had to build a statistical testing algorithm that could be used to classify music. Specifically we will work on three different cases: classification of bands from different genre, classification of bands from within the same genre, and lastly genre classification. For each case we would select several artists that we like and chose a variety of their songs to create our training set. The way that works is that we would take a one minute sample from every song in our data set and the split that sample onto 15 sub-samples, each 4 seconds long. Then we would split our collection of sub-samples on training and testing sets. Then we would create spectograms of each sub-sample. After that we would perform and SVD on our training data set of spectograms. Then we would train our model using K Nearest Neighbors algorithm and projections that we got from the previous step. Finally after our model is trained. We can now take our spectograms from testing set, project each spectogram individually on principal component basis, and then pass that projection into our model, which if implemented correctly should return the correct class of the sub-sample.

To make sure that our model was not trained on bias data set or that it was not over trained we will preform cross validation to find the average accuracy of our model.

# 2 Theoretical Background

## 2.1 Yale Faces B

Since in this part of our project in solely focuses on SVD lets recap a little bit. Imagine we have a matrix A. Then we can express that matrix in the following format: $A = USV'$, where $U$ is a set of orthogonal eigen-vectors of $AA^*$ and V is a set of orthogonal eigen-vectors of $A^*A$. Lastly, $S$ is a strictly diagonal matrix of ordered decreasing singular values.

Let's talk about applications of SVD and our specific example - image compression. First of all lets define what U, S and V represent in our situation. Let us start with U. In our case U represents a sent of singular modes also know as eigen faces. Each mode represents a specific feature that differentiates two or more images. Important thing to notice is that each feature has its own importance score, which can be used to identify which features are playing important role in differentiating different pictures and which don't. Interestingly enough those scores are represented in our matrix S. Therefore, we can conveniently see a the relationship between different eigen-faces just by looking at the scatter plot of values of S. Lastly, V represents projections of our original data onto principal component basis.

If a data set has a lot of redundancy, what is the point of storing it? Luckily we can use SVD to decompose out data set on principal components and see which modes contribute the most. Then we can set a specific boundary of tolerance of how much information are we willing to loose, and find a subset of U,S,V that we have to keep so that we would be able to reconstruct our original data (with some losses) in the future. The rest of information that we don't need in reconstruction can be erased. This is the basic idea of how one can compress the data using SVD.

## 2.2 Music Classification

The idea of music classification sound very close to the idea of photo classification, which we have already seen in one of our lectures. Therefore, we will try to manipulate our musical data set in such way that in the end it would look like we are just creating a classifier for cats/dog problem. To do so we will transform our raw data initially split our data set on training and testing subsets (approximately 80% to 20%). We will do that due to a fact that we want our training data to be independent from testing data. Since there is no way we can meaningfully compare our raw data between one another, we will instead create spectograms for each sample. That will now allow us to analyze our samples in frequencies domain. Since songs in the same genre and the same artist typically have same frequencies in common, we have a good reason to use and SVD on those spectograms. Doing that will allow to capture main features that differentiate songs from each other. After taking SVD we can now use produced projections to train our model using K Nearest Neighbors, since any linear model will fail in our case. Then we can just project testing samples onto principal component basis and pass these projections to our trained model.

# 3 Algorithm Implementation

## 3.1 Yale Faces B

We will use the same algorithm both for the uncropped and the cropped data sets. We will begin with loading our faces in one single matrix and then finding the average face of each human:

```
For each directory that represent a single person:
    subfaces = zeros(1,32256) -initialize empty array
    sizeSubfaces = 0
    For each image file that represent a single photo:
        read a photo using imread() and convert it to a double format
        reshape the matrix of a single face to vector faceA
        subfaces = subfaces+faceA;
        sizeSubfaces = sizeSubfaces + 1; - increment the number of faces
        faces = [faces; faceA]; - add a face into a matrix
    averageFaces = [averageFaces; subfaces/sizeSubfaces]; - add average face into a matrix
```

Now that we have our data matrices ready we can proceed with SVD:

```
    [m, n] = size(faces);
    mn = mean(faces, 2); 2 = faces - repmat(mn, 1, n);
    [U, S, V] = svd(faces2'/sqrt(n-1), 'econ');
```

Now we can find the number of nodes needed to reconstruct our original data with precision p.

```
    p = 0.8;
    for i = 1:size(U,2)
        if (sum(diag(S(1:i, 1:i)))/trace(S) ≥ p)
            break;
        end
```

Now we can use that index - i, to reconstruct our data and compare it with original quality:

```
    n = size(face,1);
    m = size(face,2);

    ff= U(:,1:i)S(1:i,1:i)V(:,1:i)'; - modal projections

    averageFacesRecon = zeros(size(averageFaces, 1),size(averageFaces, 2));

    for j = 1:size(averageFacesRecon,1)
        averageFacesRecon(j,:) = sum(ff(:,(j-1)*sizeSubfaces + 1:j*sizeSubfaces)')/sizeSubfaces;
```

```
end
```

## 3.2 Music Classification

We will use the same algorithm for all three cases. The only two things that will be changing is the constant for directory path of the data set and the constant corresponding to number of artists. We will begin with loading our data set. The algorithm will be mostly the same as from previous part except from two small datails. We will not be storing average and we will process audio files instead. Here is our updated the most inner loop:

```
[song,Fs] = audioread(currentSongPath);
for k = 1:numberOfSamples
    sample = song((k-1)*Fs*timeSample+1:k*Fs*timeSample,1);
    songs = [songs; sample'];
    classification = [classification; string(currentArtist)];
end
```

Then since we want to conduct a cross-validation we introduce the following outer for loop:

```
numberOfRepetitions = 5;
err = zeros(numberOfRepetitions,1);
for ind = 1:numberOfRepetitions
```

Now we for each run we will randomly generate test and train sets:

```
for i = 1:numberOfArtists
    q = randperm(numberOfSongs*numberOfSamples);
    data = songs((i-1)*numberOfSongs*numberOfSamples+1:i*numberOfSongs*numberOfSamples,
:);
    testX = [testX; data(q(1:numberOfTest),:)];
    testY = [testY; i * ones(numberOfTest, 1)];
    data(q(1:numberOfTest),:) = [];
    trainX = [trainX; data];
    trainY = [trainY; i * ones(size(data,1), 1)];
end
```

Now lets create spectograms for our samples. We are gonna be using a Gaussian filter to filter each time slice and a fixed width.

```
Fs = 32000;
n = 128000;
t = (1:n)/Fs;
L = 4;
k=(2*pi/L)*[0:n/2-1 -n/2:-1];
ks=fftshift(k);
```

tslide = 0:0.5:L;
width = 0.1;
Declare empty matrix to store spectograms of training set
for each song:
For each slice in time:
Create a filter centered around that time slice
Filter the signal using each filter
Take the fftshift of the Fourier transform
Shift the abs(frequency) and store it to the temporary matirx
Reshape the temporary spectogram matrix to a vector
Add spectogram vecor to the matrix of spectograms
Repeat the same for the testing set
Perform SVD on training set: [u,s,v] = svd(abs(trainSpectograms'), 'econ');
Create model using KNN and test it out:

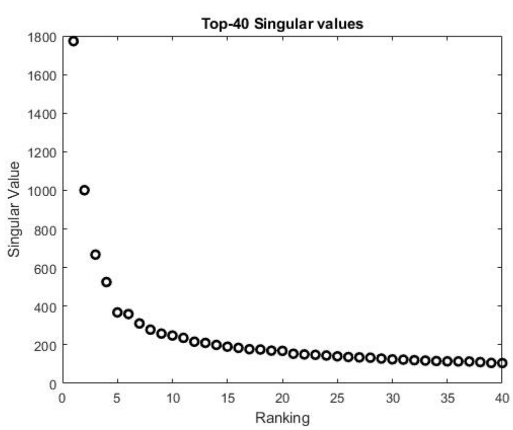
md = fitcknn(v(:,:),trainY,'NumNeighbors',5,'Distance','euclidean','DistanceWeight','squaredinverse');
testX = u(:,:).'*testSpectograms.'; - Project test spectograms singular values basis
res = predict(md, testX'); - Make a prediction
err(ind) = nnz(res-testY); - Compute error
We ended up using 5 nearest neighbours since that resulted in the most accurate model.

# 4 Computational Results

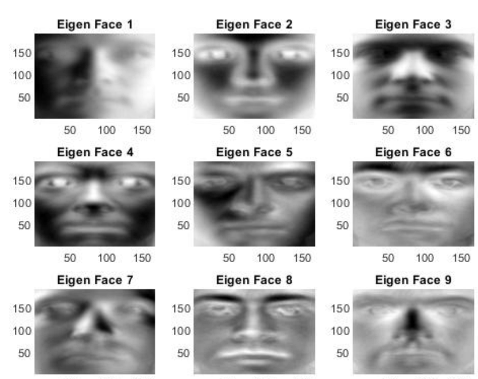
## 4.1 Yale Faces B

### 4.1.1 Cropped

To start with the following Figure 1 represents top 40 singular values as well as 9 most important singular modes:

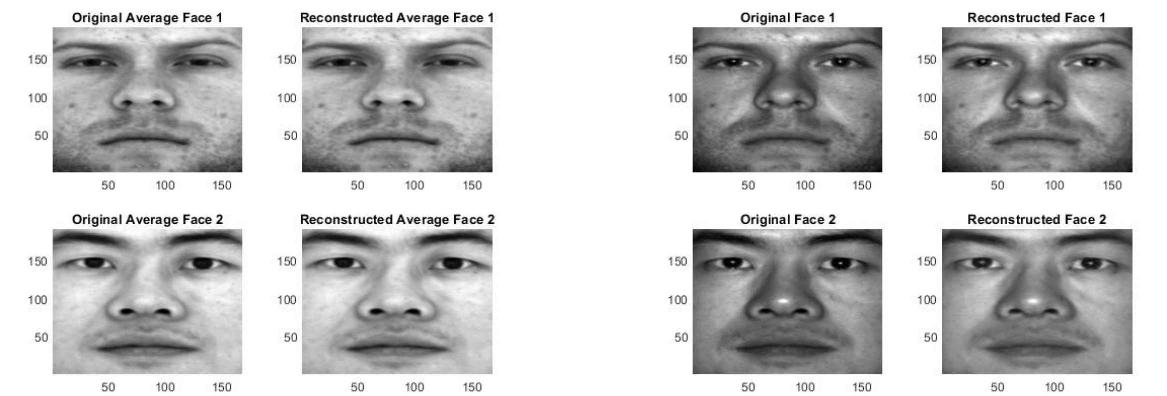



(a) Dominating Singular Values




(b) Dominating Eigen Faces

Figure 1: Key features of the SVD for cropped data set.




(a) Reconstructed Average Faces Comparison

(b) Reconstructed Original Faces Comparison

Figure 2: Performance of our reconstructions for cropped data set

Then using Figure 2 we see how well we reconstruct our data:

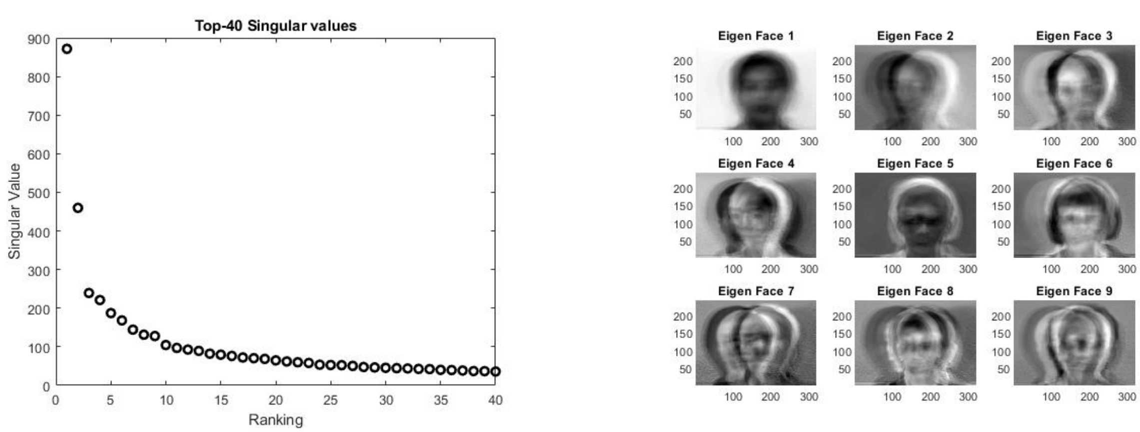
When doing this part I though that in order for reconstructions to be "good" they should be able provide 80 percent of our information back. After calculations that were mentioned in the algorithm analysis part, I found out that the required number of modes for that purpose is 677. That means that in order to keep 80 percent of our information we need 677 out of 2435 singular values, which is only 27.8 percent of our modes, which is quite impressive.

### 4.1.2 Uncropped

To start with the following Figure 3 represents top 40 singular values as well as 9 most important singular modes:

Then using Figure 4 we see how well we reconstruct our data:

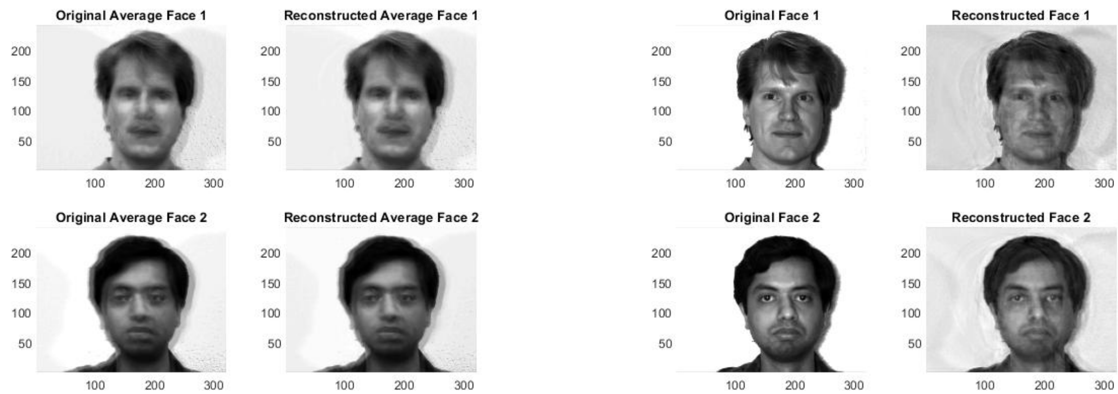
When doing this part I though that in order for reconstructions to be "good" they should be able provide 80 percent of our information back. After calculations that were mentioned in




(a) Dominating Singular Values

(b) Dominating Eigen Faces

Figure 3: Key features of the SVD for uncropped data set.




(a) Reconstructed Average Faces Comparison

(b) Reconstructed Original Faces Comparison

Figure 4: Performance of our reconstructions for uncropped data set

the algorithm analysis part, I found out that the required number of modes for that purpose is 70. That means that in order to keep 80 percent of our information we need 70 out of 165 singular values, which is only 42.4 percent of our modes, which is still quite impressive.

## 4.2 Music Classification

### 4.2.1 Sampling

In the theoretical part we have discussed how different artists have different signature. The following Figure 5 we can see how different each sample is.

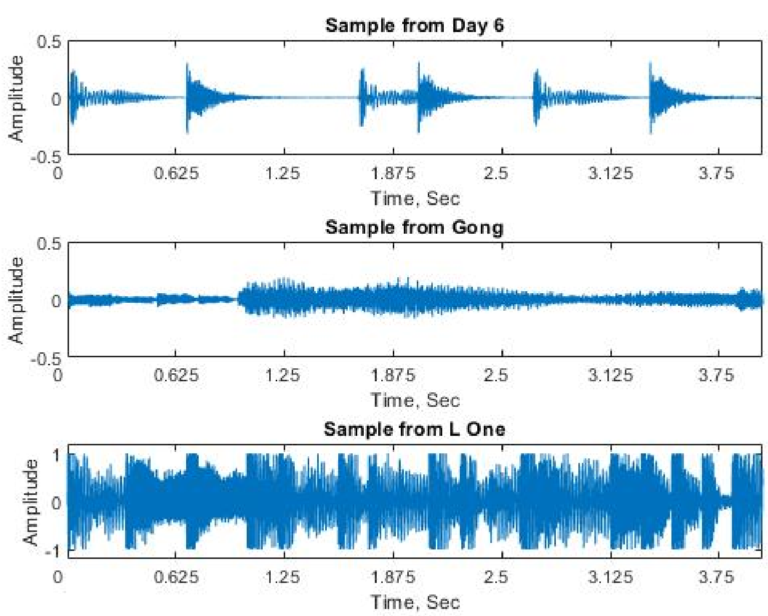



Figure 5: Differences in signatures for different artists

#### 4.2.2 Band Classification (Different Genre)

For this part I chose the following artists: Day6 (K-Pop), Gong (Chinese Hip-Hop), L'One (Russian Hip-Hop). After training our model on 5 different subsets of our original data we got the average accuracy score of 95% with the following error array: [8.33%, 4.17%, 8.33%, 4.17%, 0.00%].

#### 4.2.3 Band Classification (Same Genre)

For this part I chose the following artists: L'One (Russian Hip-Hop), Mot (Russian Hip-Hop), Jah Khalib (Russian Hip-Hop). After training our model on 5 different subsets of our original data we got the average accuracy score of 99% with the following error array: [0.00%, 4.17%, 0.00%, 0.00%, 0.00%].

#### 4.2.4 Genre Classification

For this part I chose the following artists: L'One (Russian Hip-Hop), Mot (Russian Hip-Hop), Jah Khalib (Russian Hip-Hop), Day6 (K-Pop), The Rose (K-Pop), Drug Resraurant (K-Pop), Gong (Chinese Hip-Hop), Jony J (Chinese Hip-Hop), Vava (Chinese Hip-Hop). After training our model on 5 different subsets of our original data we got the average accuracy score of 99% with the following error array: [0.26%, 0.00%, 0.17%, 0.00%, 0.00%].

# 5 Summary and Conclusions

## 5.1 Yale Faces B

After performing SVD analysis on both the cropped and uncropped data sets we can arrive to the same conclusion as in the previous project. When using SVP on clean, cropped and aligned data set, in order to restore reasonable 80% of information back we need only 677 out of 2435 singular values, which is 27.8 % of our total modes count. That means that if the rest 72.2% were discarded we would still be able to reconstruct our original images in quite good resolution. On the other had, the introduction of noise to our system makes the results of SVD not as effective. Using SVD on uncropped data set shows that we would need only 70 out of 165 singular values to achieve the same reconstruction of 80% of original information. One might foolishly say: "Well, 70 is smaller then 677 that means applying SVD on noisy data set is more beneficial". That would be a total nonsense, since two data sets have different amount of element, and if would actually find the percentage number of required modes we would see that for the uncropped data set we would need 42.4% of our modes. Although it is still far less then original 80%, it is 14.6% worse then needed percentage of modes for the cropped data set. And since now days companies are dealing with massive amounts of data, this 15% difference could lead to huge time variation. Not-only that, but also if we would compare the quality of our reconstructed images, we could see a great difference in clarity between reconstructions of uncropped data and reconstructions of cropped data. Using SVD on the noisy data will lead to a noisy projections. Afterall SVD was not inverted for the purpose of demonising the data.

### 5.2 Music Classification

During the process of finding the most optimal model I came upon several important outcomes. Initially I've tried to train my model of pure raw data - that approach did not yield any good outcomes. Then I've tried to use SVD directly on sampled segments and feed that output for the KNN to train on. That as well did not result in any extraordinary results. Then, I've tried creating spectograms, and using them to train our model. The results were still in that region of 60%-70%, which is better then randomly guessing, but not nowhere near to a 100%. Lastly, I've used SVD on spectograms and trained using output projections. In that situation I saw dramatic spike in accuracy, which lead me to my conclusion. In order to achieve great results one should design his/her model with a very great effort. A small deviation on the path of getting raw data ready to be used into training could lead to a completely different outcome. Also, using cross-validation is very important, since selected partitions for training could be bias.

# Appendices

## A MATLAB commands

dir(): Used to creates a list of folders contained in the current directory.
strcat(): Used to concatenate strings together.
imread(): Used to read in an image as a matrix of uint8.
reshape(): Used to reshapes a matrix to new dimensions.
svd(): Used to perform the SVD
pcolor(): Used to create a plot in pseudo color
audioread(): Used to reads in a music files
zeros(): Used to create empty matricies.
fft(): Used to take the Fourier transform.
fftshift(): Used to shift zero-frequency component to center of spectrum.
fitcknn(): Used to fit Data into KNN classification model
predict(): Used to get the output of our model

## B MATLAB code

```
% HW4 - PCA
%% Part 1 - Yale Faces B
%% Data Extraction

clear all; close all; clc

directionPath = './data/yalefaces_uncropped/yalefaces/';
direction = dir(directionPath);
```

```
faces = [];
averageFaces = [];

tic
for i = 3 : length(direction)
    currentDirection = direction(i).name;
    fileList = dir(strcat(directionPath, currentDirection));
    subfaces = zeros(1,77760);
    sizeSubfaces = 0;
    for j = 3 : length(fileList)
        currentFilePath = strcat(strcat(directionPath, currentDirection), ...
            '/',fileList(j).name);
        face = double(imread(currentFilePath));
        faceA= reshape(face, [1, size(face, 1)*size(face, 2)]);
        faces = [faces; faceA];
        subfaces = subfaces+faceA;
        sizeSubfaces = sizeSubfaces + 1;
    end
    averageFaces = [averageFaces; subfaces/sizeSubfaces];
%      imshow(uint8(reshape(averageFaces(i-2,:), [size(face, 1), size(face, 2)])));
end
save("averageFaces2.mat", "averageFaces", "-mat");
toc

%% SVD of data
[m, n] = size(faces);
mn = mean(faces, 2);
faces2 = faces - repmat(mn, 1, n);

[U, S, V] = svd(faces2'/sqrt(n-1), 'econ');

%% Plot Singular Values
plot(diag(S(1:40, 1:40)), 'ko', 'Linewidth', [2])
title('Top-40 Singular values')
xlabel('Ranking')
ylabel('Singular Value')

%% Plot Eigenfaces
n = size(face,1);
m = size(face,2);

figure()
```

```
for i = 1:9
    subplot(3,3,i)
    face = reshape(U(:,i),n, m);
    pcolor(flipud(face)), shading INTERP, colormap(gray)
    title(sprintf('Eigen Face %d', i))
end

%% Find number of nodes needed to reconstruct with p precision.
p = 0.8;

for i = 1:size(U,2)
    if (sum(diag(S(1:i, 1:i)))/trace(S) >= p)
        break;
    end
end

%% Reconstruct the faces
ff= U(:,1:i)*S(1:i,1:i)*V(:,1:i)'; % modal projections

%% Plot Reconstructed Faces
figure()
subplot(2,2,1)
pcolor(flipud(reshape(faces(1,:), n, m))), shading INTERP, colormap(gray)
title('Original Face 1')
subplot(2,2,2)
pcolor(flipud(reshape(ff(:,1), n, m))), shading INTERP, colormap(gray)
title('Reconstructed Face 1')
subplot(2,2,3)
pcolor(flipud(reshape(faces(12,:), n, m))), shading INTERP, colormap(gray)
title('Original Face 2')
subplot(2,2,4)
pcolor(flipud(reshape(ff(:,12), n, m))), shading INTERP, colormap(gray)
title('Reconstructed Face 2')

%% Projections of Average Faces
averageFacesRecon = zeros(size(averageFaces, 1),size(averageFaces, 2));
for j = 1:size(averageFacesRecon,1)
    averageFacesRecon(j,:) = sum(ff(:,(j-1)*11 + 1:j*11)')/11;
end

```

```
%% Plot Average Faces
figure()
subplot(2,2,1)
pcolor(flipud(reshape(averageFaces(1,:), n, m))), shading INTERP, colormap(gray)
title('Original Average Face 1')
subplot(2,2,2)
pcolor(flipud(reshape(averageFacesRecon(1,:), n, m))), shading INTERP, colormap(gray)
title('Reconstructed Average Face 1')
subplot(2,2,3)
pcolor(flipud(reshape(averageFaces(2,:), n, m))), shading INTERP, colormap(gray)
title('Original Average Face 2')
subplot(2,2,4)
pcolor(flipud(reshape(averageFacesRecon(2,:), n, m))), shading INTERP, colormap(gray)
title('Reconstructed Average Face 2')

%% Music classification
%% Test 1 - Band classification
clear all; close all; clc

mainDirectionPath = "music-samples/test-2/sample-60-2/";
mainDirection = dir(mainDirectionPath);

numberOfArtists = 3;
numberOfSongs = 9;
timeSample = 4;
numberOfSamples = round(60/timeSample) - 1;

songs = [];
classification = [];
tic
for i = 3 : length(mainDirection)
    currentArtist = ma
    inDirection(i).name;
    currentArtistDirection = dir(strcat(mainDirectionPath, currentArtist));
    fprintf("Processing %s\n", currentArtist)
    for j = 3 : length(currentArtistDirection)
        fprintf("Processing %s\n", currentArtistDirection(j).name)
        currentSongPath = strcat(strcat(mainDirectionPath, currentArtist), ...
            '/',currentArtistDirection(j).name);
```

```
            [song,Fs] = audioread(currentSongPath);
            for k = 1:numberOfSamples
                sample = song((k-1)*Fs*timeSample+1:k*Fs*timeSample,1)
                    ;
                songs = [songs; sample'];
                classification = [classification; string(currentArtist
                    )];
            end
        end
end
toc

clearvars mainDirectionPath mainDirection currentArtist i j k
    currentSongPath song sample currentArtistDirection

% Plot three different plots
subplot(3,1,1)
plot(songs(1,:))
title('Sample from Day 6')
axis([0*Fs 4*Fs -0.5 0.5])
xt = get(gca, 'XTick');
set(gca, 'XTick', xt, 'XTickLabel', xt/Fs)
xlabel('Time, Sec')
ylabel('Amplitude')
subplot(3,1,2)
plot(songs(numberOfSamples*numberOfSongs+1,:))
axis([0*Fs 4*Fs -0.5 0.5])
xt = get(gca, 'XTick');
set(gca, 'XTick', xt, 'XTickLabel', xt/Fs)
title('Sample from Gong')
xlabel('Time, Sec')
ylabel('Amplitude')
subplot(3,1,3)
plot(songs(2*numberOfSamples*numberOfSongs,:))
axis([0*Fs 4*Fs -1.2 1.2])
xt = get(gca, 'XTick');
set(gca, 'XTick', xt, 'XTickLabel', xt/Fs)
title('Sample from L One')
xlabel('Time, Sec')
ylabel('Amplitude')

% Create Training and Testing Sets
clc

```

```
numberOfTest = round(numberOfSongs*numberOfSamples*0.2/
    numberOfArtists);

numberOfRepetitions = 5;
err = zeros(numberOfRepetitions,1);

for ind = 1:numberOfRepetitions

    testX = [];
    testY = [];
    trainX = [];
    trainY = [];

    disp("Creating Test and Train Data Sets")
    for i = 1:numberOfArtists
        q = randperm(numberOfSongs*numberOfSamples);
        data = songs((i-1)*numberOfSongs*numberOfSamples+1:i*
            numberOfSongs*numberOfSamples, :);
        fprintf("Current test set for %d is %s \n", i, mat2str(q
            (1:numberOfTest)))
        testX = [testX; data(q(1:numberOfTest),:)];
        testY = [testY; i * ones(numberOfTest, 1)];
        data(q(1:numberOfTest),:) = [];
        trainX = [trainX; data];
        trainY = [trainY; i * ones(size(data,1), 1)];
    end
    disp("Done Creating Sets")

    clearvars q data i

    % Creating spectogram constants
    clc

    Fs = 32000;
    n = 128000;
    t = (1:n)/Fs;
    %each sample is 4 seconds
    L = 4;
    k=(2*pi/L)*[0:n/2-1 -n/2:-1];
    ks=fftshift(k);
    tslide = 0:0.5:L;

    % Getting Gaussian Spectogram matrices for train
    clc
    testSpectograms = zeros(size(testX,1), length(tslide)*n);
```

```
    trainSpectograms = zeros(size(trainX,1),length(tslide)*n);

    width = 0.1;

    % Spectograms of test set
    for i = 1:size(testSpectograms, 1)
        fprintf("Processing the %dth sample out of %d\n", i, size(testSpectograms, 1))
        currentSpectogram = [];
        for j=1:length(tslide)
            g = exp(-width*(t - tslide(j)).^2); %Gaussian Filter
            currentSpectogramG = g.*testX(i,:); %filtered with gaussian
            currentSpectogramFG = fft(currentSpectogramG); %FFT gaussian
            currentSpectogram = [currentSpectogram;abs(fftshift(currentSpectogramFG))];
        end
        %     figure()
        testSpectograms(i,:) = reshape(currentSpectogram, 1, length(tslide)*n);
    end
    disp("I'm done with test spectogram!")

    % Spectograms of training set
    for i = 1:size(trainSpectograms, 1)
        fprintf("Processing the %dth sample out of %d\n", i, size(trainSpectograms, 1))
        currentSpectogram = [];
        for j=1:length(tslide)
            g = exp(-width*(t - tslide(j)).^2); %Gaussian Filter
            currentSpectogramG = g.*trainX(i,:); %filtered with gaussian
            currentSpectogramFG = fft(currentSpectogramG); %FFT gaussian
            currentSpectogram = [currentSpectogram;abs(fftshift(currentSpectogramFG))];
        end
        %     figure()
        %     pcolor(tslide, ks, Sgt_spec_train.'), shading interp, colormap(hot)
        trainSpectograms(i,:) = reshape(currentSpectogram, 1, length(tslide)*n);
    end
    disp("I'm done with train spectogram!")
```

```
    clearvars tslide t currentSpectogram currentSpectogramG i ks k
        g j n testX trainX width currentSpectogramFG

    % SVD
    clc

    disp("Starting SVD")
    tic
    [u,s,v] = svd(abs(trainSpectograms'), 'econ');
    toc
    disp("Done with SVD")

    disp("Finding 80% recreation")
    p = 0.8;
    for index = 1:size(u,2)
        if (sum(diag(s(1:index, 1:index)))/trace(s) >= p)
            break;
        end
    end

    clearvars trainSpectograms p

    % Machine Learning 1
    clc

    disp("Starting ML")
    disp("Creating Model")
    Model = fitcknn(v(:,:),trainY,'NumNeighbors',5,'Distance','
       euclidean','DistanceWeight','squaredinverse');
    disp("Evaluating results")
    testX = u(:,:).'*testSpectograms.';
    res = predict(Model, testX');
    err(ind) = nnz(res-testY);
    fprintf("Current precision is %d%% \n", round((1 - err(ind)/(
       size(testX, 2)))*100));
end
cvErr = 1 - sum(err)/(3*numberOfTest*numberOfRepetitions);
fprintf("Total precision is %d%% \n", round(cvErr*100))
```