AMATH 482 HW 4

Zoheb Siddiqui

March 9, 2019

Abstract

This assignment has two parts. The first part involves exploring SVD and information stored in the matrices produced by SVD. The second part involves performing classification to classify songs to artists and genres.

Introduction

Part 1

We are given a number of pictures of peoples' faces. We must perform SVD analysis on these face images and gain a better understanding of what is the result of this decomposition, its advantages and uses.

We have a set of images which is cropped to make the face more prominent and then aligned. We have another set in which these thing haven't been done. We must perform SVD on both these sets and compare the results.

Part 2

Part two consists of 3 cases however we can address the problem in all three cases with a general introduction. The task is to do music classification. We have three classes of music and each class has a number of 4 second snippets of songs pertaining to that class.

We also have a few 4 second snippets of song intended to be used as a test set to find the accuracy of our classification algorithms. We use two pre-processing techniques and two classification techniques for this part and compare accuracies on these techniques.

First we create a spectogram for each snippet, perform SVD on a matrix of spectograms of the training data and then create projections for the train and test data. Now that the pre-processing part is done, we average the projections for the training data and then use these to predict classes for the test data by comparing distances between the projected test data vector and the average of the projected training vectors.

For the second method we just create the spectogram matrix. Once this is done we use the K

Nearest Neighbors algorithm. We first train it on the training spectogram matrix and then use the testing spectogram matrix to find accuracy.

Theoretical background

Part 1

From the previous homework assignment we know that performing SVD on a matrix, M, gives three principal components U, S and V. U is a set of orthogonal eigen-vectors of MM^* and V is a set of orthogonal eigen-vectors of M^*M. S is a diagonal matrix consisting of the singular values of M which are the square root of the eigenvalues of both MM^* and M^*M. We can reconstruct M from U, S and V as $M = USV^*$.

Since we know that the matrix M is a matrix of face images we can talk about U, S and V in those terms.

Let $M \in R^{m,n}$ s.t m is the number of pixels in each face and n is the number of faces. Thus, each face is a column vector. Now SVD(M) gives $U \in R^{m,n}$, $S \in R^{n,n}$ and $V \in R^{n,n}$.

Now U represents the eigen-faces. The eigen-faces can be thought of as features that are responsible for the construction and identification of a face. The number of features is the same as the number of faces however not all of these features are equally important.

The diagonal values in S represent the importance of each eigen-face (column in U). Thus, suppose S_{ii} is extremely large, then the column vector U_i will be an extremely important eigen-face and vice versa. Thus, once we have the eigen-faces we multiply them with S which gives a $R^{m,n}$ matrix. This matrix is composed of the weighted eigen-faces and the weights are the singular values.

The ith column of V^* represents the coefficients of the linear combination of these weighted eigen-faces responsible for re constructing the ith face. Thus, $M(:, i) = USV^*(:, i)$.

Now since not all eigen-faces are important, we don't need to use all of them to get an accurate reconstruction. This is great as we reduce the size of the matrices and save on computational space and time.

Suppose k eigen-faces are sufficient s.t $k < n$. The reconstruction $M_{reconstructed} = U(:, 1 : k)S(1 : k, 1 : k)V^*(1 : k, :)$. $M_{reconstructed} \in R^{m,n}$ and again each column represents a face.

Part 2

The idea of classification is as follows: given a vector representing 1 data point, we must figure out which class it belongs to. In the case of music classification the problem is that we have a vector representing 4 seconds of a song and we need to figure out which artist/genre that snippet of song belongs to.
We must first process the data. Since the data is in the form of sound waves, we would like to extract the frequency at each instant in time and use it for classification. For this we construct a Gaussian filtered spectogram.

For the 1st classification algorithm, we further process the data by performing SVD on it and projecting the training spectogram onto the U matrix. We also project the testing spectogram onto the U matrix. Now our pre processing phase is complete.

Now we take the average of the training projection matrix to generate n vectors where n is the number of classes. Now for each vector, v_i, in the testing projection matrix, we find the vector v_j in the training projection matrix s.t $||v_i - v_j||_1$ is minimized. Then we classify v_i to the came class as v_j. Since there are only 3 classes and the avg training projection matrix has 3 vectors, the class of $v_j = j$

For the 2nd classification algorithm we use the spectograms themselves.

Now we use the K Nearest Neighbors algorithm to classify the data points. The way this algorithm works is that, given a song vector v_i, we find the k closest vectors to it through euclidean distance (2 norm). The class that v_i belongs to is a weighted average of the classes of the k nearest neighbors of v_i.

Algorithm

Note: In the first part, the algorithm for both cropped and uncropped is the same. The only difference is how the data is loaded as there is one more directory in the cropped folder.

Note: In the second part, the algorithms for test 1, test 2 and test 3 are the same.

Part 1

```
faces = []
AvgFaces = []

For each directory:
    sum of faces = zeros(1,32256)
    number of faces = 0
    For each image file:
        read image through imread() and convert to double
        reshape image matrix to vector v
        sum of faces += v
        number of faces += 1
        faces = [faces; v]
    avg face = sum of faces/number of faces
    AvgFaces = [AvgFaces; avg face]
```

Now we have the face matrix containing all faces and the avg faces matrix.
Plot the first 9 avg faces and save them.
De mean the faces matrix.
Perform SVD on the transpose of the de meaned faces matrix to get U, S, V.
Plot the first 9 eigen-faces (9 columns of U).
Plot the first 50 singular values to see trend.
Reduce the size of U, S, V to remove redundant modes.

Call these matrices U_small, S_small, V_small.
reconstruction = transpose((U_small)(S_small)(V_small*))
Take the average of these reconstructed faces so we have the avg reconstructed face for each person
Plot the first 9 avg reconstructed faces and compare to true average faces plotted earlier.

Part 2

Read in 9 music files in order. 3 for each class. Thus, the first 3 songs are for class 1 and so on.
Convert each music file to one dimensional vector.
Make the length of each file uniform.
Break each song vector into 40 snippets of 4 seconds each. Take parts 5 to 34 (30 for each song) for training and 36 to 38 (3 for each song) for testing.
Note: This is done to remove the silence in the start and end to get more accurate results.
Thus, TrainingMatrix $\in R^{270,amplitudes}$ s.t the first 90 entries belong to class 1 and so on.
TestingMatrix $\in R^{27,amplitudes}$ s.t the first 3 entries belong to class 1, the next 3 to class 2 and the next 3 to class 3. This pattern repeats till we have 27 entries.

```
Fs = 44100; L = number of seconds in a snippet (4)
t = (1:amplitudes)/Fs
k = (2*pi/L)*[0:n/2-1 -n/2:-1]; ks = fftshift(k)
declare empty SpectogramTrainMatrix and empty SpectogramTestMatrix.

For each 4 second snippet named s in TrainingMatrix:
    Declare empty spectogram matrix
    For tslide = 1:0.5:L:
        create Gaussian filter around t - tslide.
        Filter s using Gaussian filter.
        fft() the filtered signal
        add the abs(fftshift(filtered signal)) to spectogram matrix
    flatten the spectogram matrix to a vector
    add reshaped spectogram vector to the SpectogramTrainMatrix.

Do the same for SpectogramTestMatrix.
```

SpectogramTrainMatrix, SpectogramTestMatrix $\in R^{snippets,frequencies}$

Note: The above part is same for both classification techniques.

For 1st classification technique:
Perform SVD on the transpose of SpectogramTrainMatrix to get U, S, V.
create TrainProjectionMatrix = (transpose(U))(transpose(SpectogramTrainMatrix))
create TestProjectionMatrix = (transpose(U))(transpose(SpectogramTestMatrix))
Now we find the AvgTrainProjection for each class. Each class's projection will consist of 1 row vector with 270 values (number of modes).
AvgTrainProjection = zeros(270,3)

```
For i= 1:3 //3 classes
    For j = 1:90 //90 snippets for each class
        AvgTrainProjection(:,i) = AvgTrainProjection(:,i) + TrainProjectionMatrix(:,(i-1)*90 + j);
    AvgTrainProjection(:,i) = AvgTrainProjection(:,i)/90

For each vector, v$_i$ in TestProjectionMatrix:
    calculate L1 norm of v$_i$ with each vector in AvgTrainProjection
    print the index of the vector v$_j$ $\in$ AvgTrainProjection with the smallest L1 norm.
    that is the prediction for the class of v$_i$.
```

Note that the output can be 1,2 or 3. This will hold true for the 2nd classification technique as well.

For 2nd classification technique:
We shall perform K nearest neighbors classification with K = 5.

```
For each vector, v$_i$ in SpectogramTestMatrix:
    Find 5 vectors in SpectogramTrainMatrix s.t. L2 norm of v$_i$ with those vectors is minimum.
    Find the class values of these 5 vectors i.e. 1,2 or 3.
    Take a weighted average of these class values s.t. the weights are $\frac{1}{L2\ norm}$
    Thus, avg = $\frac{\sum_{j=1}^{5}(\frac{1}{d_j}*class(v_j))}{\sum_{j=1}^{5}\frac{1}{d_j}}$
    Here $class(v_j)$ = 1,2, or 3
    $d_j = ||v_i - v_j||_2$ where $v_j$ is the jth closest vector to $v_i$ in SpectogramTrainMatrix.
    PredictedClass = round(avg)
```

Note: In this technique, more similar vectors get more weight/importance.

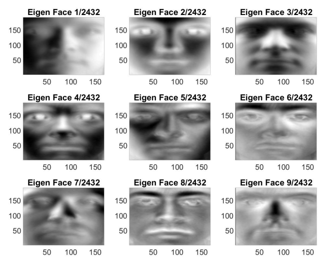
For both classification techniques:
The desired output is:

```
    [1 1 1 2 2 2 3 3 3]
    [1 1 1 2 2 2 3 3 3]
    [1 1 1 2 2 2 3 3 3]
```

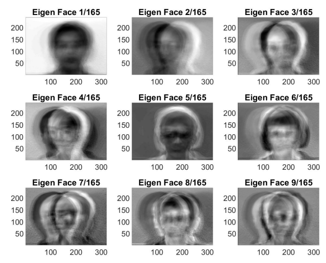
Computational Results

Part 1

The plot of the 9 most important singular modes(eigen-faces) looks like:



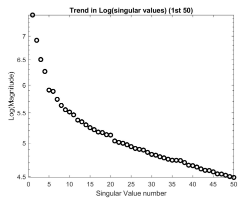
(a) Cropped eigen faces



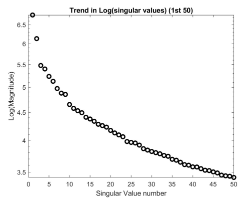
(b) Un Cropped eigen faces

Figure 1: First 9 eigen faces

The plot of the log of first 50 singular values looks like:



(a) Log of Cropped Singular Values

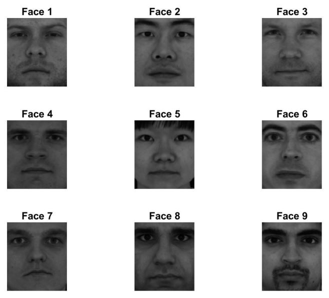


(b) Log of Un Cropped Singular Values

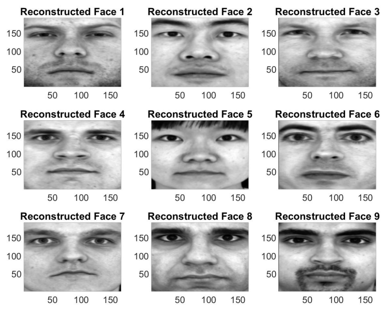
Figure 2: Log of first 50 singular values

My hypothesis is that the number of modes necessary are the number of singular values that provide 75% of the information. Thus I use 500 singular values out of 2432 for the cropped images and 60 singular values out of 165 for the un-cropped images.

The comparison in true and reconstructed average faces for cropped images:
The comparison in true and reconstructed average faces for un cropped images:

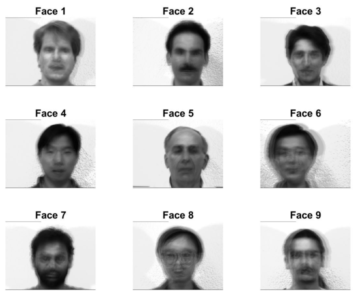


(a) True Avg Faces



(b) Reconstructed Avg Faces

Figure 3: Comparison of true and reconstructed avg faces for cropped images


(a) True Avg Faces

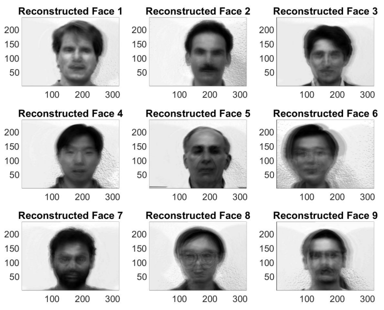

(b) Reconstructed Avg Faces

Figure 4: Comparison of true and reconstructed avg faces for un cropped images

Part 2

For test 1:
Class 1: Taylor swift (pop singer), Class 2: Kendrick Lamar (rap singer), Class 3: Caravan Palace (male jazz band)
With the 1st classification technique we get 100% testing accuracy.
The predictions are:

```
1 1 1 2 2 2 3 3 3
1 1 1 2 2 2 3 3 3
1 1 1 2 2 2 3 3 3
```

With the 2nd classification technique we get 77.78% testing accuracy.
The predictions are:

```
2. 1. 1. 2. 2. 2. 1. 3. 3.
2. 1. 1. 2. 2. 2. 1. 3. 3.
2. 1. 1. 2. 2. 2. 1. 3. 3.
```

For test 2:
Class 1: Taylor swift, Class 2: Katy Perry, Class 3: Maroon 5 (male band). These are all pop artists.
With the 1st classification technique we get 88.89% testing accuracy.
The predictions are:

```
1 1 1 2 2 2 3 2 3
1 1 1 2 2 2 3 2 3
1 1 1 2 2 2 3 2 3
```

With the 2nd classification technique we get 77.78% testing accuracy.
The predictions are:

```
3. 3. 1. 2. 2. 2. 2. 3. 3.
3. 3. 1. 2. 2. 2. 2. 3. 3.
3. 3. 1. 2. 2. 2. 2. 3. 3.
```

For test 3:
Class 1: Pop(3 different artists), Class 2: Rap(3 different artists), Class 3: Heavy Metal(3 different artists).
With the 1st classification technique we get 77.78% testing accuracy.

The predictions are:

```
1 3 1 2 3 2 3 3 3
1 3 1 2 3 2 3 3 3
1 3 1 2 3 2 3 3 3
```

With the 2nd classification technique we get 66.67% testing accuracy.
The predictions are:

```
3. 3. 1. 2. 2. 2. 2. 3. 3.
3. 3. 1. 2. 2. 2. 2. 3. 3.
3. 3. 1. 2. 2. 2. 2. 3. 3.
```

Conclusion

Part 1

The percentage of modes required to gain 75% of the information is 20.6% for the cropped images and 36.4% for the uncropped images.
More modes are required for he uncropped images as there is more noise in these images.

The log(singular values) graph looks similar to the exponential distribution. The graph without the log would descend even more steeply. Thus, we can say that the importance of the modes decays extremely quickly.

The uncropped images have more noisy eigen faces and more noisy reconstruction. Thus, more modes are required to get a reconstruction of the same quality as the cropped ones.

Thus, finally we can say that SVD is quite sensitive of noise and requires a fair amount of pre processing before it can be used effectively.

Part 2

The first clear conclusion is that we can draw is that using SVD as a part of the pre processing gives better results.
This is because KNN is a better classification technique than the 1st one and yet the first technique yeilds better results.

The second conclusion is that accuracy is better when we classify artists and even better when the artists belong to different genres. Accuracy is worst when classifying genres as there is too much varience within genres.

The third conclusion is that things are generally mis-classified when the artists singing the songs have similar pitch.

Appendices

Note: The first part and the classification through the average projection are written in Matlab whereas the K Nearest Neighbors classification is written in python.

Function Used

Matlab

dir(): 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(): Reshapes a matrix to new dimensions.
svd(): Does the SVD decomposition to generate [U,S,V].
pcolor(): Used to create a plot in pseudo color
bar(): Used to create a histogram.
audioread(): reads in a music file and outputs [amplitude, Fs].
zeros(): creates a matrix filled with zeros.
fft(): Used to take the Fourier transform.
fftshift(): Shifts the Fourier transform so the largest magnitudes are in the center.
csvwrite(): Writes a matrix to csv

Python

numpy.genfromtxt(): a function in the numpy package that reads csv files.
numpy.zeros(): has the same functionality as zeros() in matlab
sklearn.neighbors.KNeighborsClassifier(): a function in the sklearn package that performs KNN classification.
The hyper-parameters we pass in this function are k = 5, distance_metric = 'Eculidean', weights = 'distance'.
KNeighborsClassifier.fit(): used to train the model.
KNeighborsClassifier.score(): outputs the accuracy for a gived data set.
KNeighborsClassifier.predict(): outputs the predictions for a given dataset.

Code

Part 1

Cropped

```
clear all; close all; clc;
%% constructing avg face matrix

directory_Path = 'Data/Cropped/CroppedYale/';
```

```
directory = dir(directory_Path);

faces = [];
averageFaces = [];

for i = 3:length(directory)
    %finds each folder in CroppedYale
    current_directory = directory(i).name();
    %list of all file names in current folder
    file_list = dir(strcat(directory_Path, current_directory));
    %store sum of all faces of 1 type
    sum_faces_type_1 = zeros(1,32256);
    number_of_faces_type_1 = 0;

    for j = 3 : length(file_list)
        current_folder_path = strcat(directory_Path, current_directory);
        %find each file for each face
        currentFilePath = strcat(current_folder_path,...
            '/',file_list(j).name);

        %convert face from uint8 to double
        face = double(imread(currentFilePath));
        %reshape face to make it linear
        face_to_add = reshape(face, ...
            [1, size(face, 1)*size(face, 2)]);
        %add faces to matrix
        faces = [faces; face_to_add];

        %add face to sub matrix
        sum_faces_type_1 = sum_faces_type_1+face_to_add;
        number_of_faces_type_1 = number_of_faces_type_1 + 1;
    end
    %add average face to avg face matrix
    averageFaces = [averageFaces; ...
        sum_faces_type_1/number_of_faces_type_1];
end

%% De mean data
mn = mean(faces, 2);
for i=1:size(faces,1)
    faces(i,:) = faces(i,:) - mn(i,1);
end
%% SVD
[U, S, V] = svd(faces'/sqrt(32256-1), 'econ');
%% Drawing Eigen Faces
```

```
figure(1)
for i = 1:9
    subplot(3,3,i)
    eigenface = reshape(U(:,i),size(face, 1), size(face, 2));
    pcolor(flipud(eigenface)), shading INTERP, colormap(gray)
    title(sprintf('Eigen Face %d/2432', i));
end
figure(2)
diags = diag(S);
plot(log(diags(1:50)), 'ko', 'Linewidth', [2])
xlabel('Singular Value number')
ylabel('Log(Magnitude)')
title('Trend in Log(singular values) (1st 50)')
%% Projecting Avg faces
%plotting projections on bar graph
figure(3)
for i = 1:9
    projection = U(:,1:500).'*averageFaces(i, :).';
    subplot(3, 3, i)
    bar(projection), set(gca, 'Xlim', [0 100], 'Ylim', [-4000 4000]);
    title(sprintf('Projection of Avg Face %d on U_{short}', i));

end
%% Drawing Avg faces
figure(4)
for i = 1:9
    subplot(3, 3, i)
    imshow(uint8(reshape(averageFaces(i, :),...
        [size(face, 1), size(face, 2)])));
    title(sprintf('Face %d', i));
end
%% Reconstructing faces
V_T = V.';
V_T = V_T(1:500,:);
reconstructed = (U(:,1:500)*S(1:500,1:500)*V_T).'*sqrt(32256-1);
figure(5)
%plotting 9 reconstructed average faces
for i = 1:9
    subplot(3, 3, i)
    avg_face_reconstructed = zeros(1,32256);
    for j = 1:64
        avg_face_reconstructed = avg_face_reconstructed + ...
            reconstructed(j + (i-1)*64,:);
    end
    pcolor(flipud(reshape(avg_face_reconstructed, ...
        [size(face, 1), size(face, 2)]))), ...
        shading INTERP, colormap(gray);
    title(sprintf('Reconstructed Face %d', i));
end
```

UnCropped

```
clear all; close all; clc;
%% constructing avg face matrix

directory_Path = 'Data/UnCropped/yalefaces/';
directory = dir(directory_Path);

faces = [];
averageFaces = [];

for i = 1:15

    sum_faces_type_1 = zeros(1,77760);
    number_of_faces_type_1 = 0;

    for j = 1:11
        %find each file for each face
        currentFilePath = strcat(directory_Path, '/',...
            directory((i-1)*11 + j + 2).name);

        %convert face from uint8 to double
        face = double(imread(currentFilePath));
        %disp(face);
        %reshape face to make it linear
        face_to_add = reshape(face, ...
            [1, size(face, 1)*size(face, 2)]);
        %add faces to matrix
        faces = [faces; face_to_add];

        %add face to sub matrix
        sum_faces_type_1 = sum_faces_type_1+face_to_add;
        number_of_faces_type_1 = number_of_faces_type_1 + 1;
    end
    %add average face to avg face matrix
    averageFaces = [averageFaces; ...
        sum_faces_type_1/number_of_faces_type_1];
end

%% De mean data
```

```
mn = mean(faces, 2);
for i=1:size(faces,1)
    faces(i,:) = faces(i,:) - mn(i,1);
end
%% SVD
[U, S, V] = svd(faces'/sqrt(77760-1), 'econ');
%% Drawing Eigen Faces
figure(1)
for i = 1:9
    subplot(3,3,i)
    eigenface = reshape(U(:,i),size(face, 1), size(face, 2));
    pcolor(flipud(eigenface)), shading INTERP, colormap(gray)
    title(sprintf('Eigen Face %d/165', i));
end
figure(2)
diags = diag(S);
plot(log(diags(1:50)), 'ko', 'Linewidth', [2])
xlabel('Singular Value number')
ylabel('Log(Magnitude)')
title('Trend in Log(singular values) (1st 50)')
%% Projecting Avg faces
%plotting projections on bar graph
figure(3)
for i = 1:9
    projection = U(:,1:60).'*averageFaces(i, :).';
    subplot(3, 3, i)
    bar(projection), set(gca, 'Xlim', [0 100], 'Ylim', [-4000 4000]);
    title(sprintf('Projection of Avg Face %d on U_{short}', i));

end
%% Drawing Avg faces
figure(4)
for i = 1:9
    subplot(3, 3, i)
    imshow(uint8(reshape(averageFaces(i, :), ...
        [size(face, 1), size(face, 2)])));
    title(sprintf('Face %d', i));
end
%% Reconstructing faces
V_T = V.';
V_T = V_T(1:60,:);
reconstructed = (U(:,1:60)*S(1:60,1:60)*V_T).'*sqrt(77760-1);
figure(5)
%plotting 9 reconstructed average faces
for i = 1:9
```

```
    subplot(3, 3, i)
    avg_face_reconstructed = zeros(1,77760);
    for j = 1:11
        avg_face_reconstructed = avg_face_reconstructed + ...
            reconstructed(j + (i-1)*11,:);
    end
    pcolor(flipud(reshape(avg_face_reconstructed, ...
        [size(face, 1), size(face, 2)]))), ...
        shading INTERP, colormap(gray);
    title(sprintf('Reconstructed Face %d', i));
end
```

Part 2

Note: the code is the same for all 3 test cases. The data is simply stored in a new folder for each case, thus I shall only be listing the code for the first case.

Matlab

```
clear all; close all; clc;
%% Loading songs

%Fs can be found by doing [song, Fs] = audioread();
Fs = 44100;

TS_1 = audioread("Data/Music_Part_1\TS1.mp3");
TS_2 = audioread("Data/Music_Part_1\TS2.mp3");
TS_3 = audioread("Data/Music_Part_1\TS3.mp3");
KL_1 = audioread("Data/Music_Part_1\KL1.mp3");
KL_2 = audioread("Data/Music_Part_1\KL2.mp3");
KL_3 = audioread("Data/Music_Part_1\KL3.mp3");
CP_1 = audioread("Data/Music_Part_1\CP1.mp3");
CP_2 = audioread("Data/Music_Part_1\CP2.mp3");
CP_3 = audioread("Data/Music_Part_1\CP3.mp3");

%% Taking avg of left and right stereo and transposing
TS_1 = ((TS_1(:,1) + TS_1(:,2))/2).';
TS_2 = ((TS_2(:,1) + TS_2(:,2))/2).';
TS_3 = ((TS_3(:,1) + TS_3(:,2))/2).';
KL_1 = ((KL_1(:,1) + KL_1(:,2))/2).';
KL_2 = ((KL_2(:,1) + KL_2(:,2))/2).';
KL_3 = ((KL_3(:,1) + KL_3(:,2))/2).';
CP_1 = ((CP_1(:,1) + CP_1(:,2))/2).';
CP_2 = ((CP_2(:,1) + CP_2(:,2))/2).';
CP_3 = ((CP_3(:,1) + CP_3(:,2))/2).';
```

```
%% Finding smallest length to make all songs uniform
min_length_1 = min(size(TS_1, 2), size(TS_2, 2));
min_length_2 = min(size(TS_3, 2), size(KL_1, 2));
min_length_3 = min(size(KL_2, 2), size(KL_3, 2));
min_length_4 = min(size(CP_1, 2), size(CP_2, 2));
min_length_5 = min(min_length_1, size(CP_3, 2));
min_length_6 = min(min_length_2, min_length_3);
min_length_7 = min(min_length_1, min_length_4);

min_length = min(min_length_6, min_length_7);
%% Creating song matrix
%removing 5 seconds from start and end of each song
%as it is mostly silence
%we remove a bit more than that to make segments uniform
songs = [TS_1(1, 178657:min_length - 178657 - 7)
    TS_2(1, 178657:min_length - 178657 - 7)
    TS_3(1, 178657:min_length - 178657 - 7)
    KL_1(1, 178657:min_length - 178657 - 7)
    KL_2(1, 178657:min_length - 178657 - 7)
    KL_3(1, 178657:min_length - 178657 - 7)
    CP_1(1, 178657:min_length - 178657 - 7)
    CP_2(1, 178657:min_length - 178657 - 7)
    CP_3(1, 178657:min_length - 178657 - 7)];
%% Dividing the songs 40 into snippets of 4 seconds each
%for each song put segment 5 to 34 in train
% Putting 9*30 into train
%for each song put segment 35 to 37 in test
% and 9*3 into test
%thus, we have 270 train and 27 test points
train_snippets = zeros(30*9,178656);
test_snippets = zeros(27,178656);
for i = 1:9
    for j = 5:34
        train_snippets((i-1)*30 + j-4,:) = ...
            songs(i,(j-1)*178657 + 1:j*178657 - 1);
    end
end
for j = 35:37
    for k = 1:3
        for i = 1:9
                test_snippets(i + (k-1)*9,:) = ...
                    songs(i,(j-1)*178657 + 1:j*178657 - 1);
        end
    end
```

```
end
%% Creating spectogram shit
t = (1:178656)/Fs;
n=178656;
%each snipped is 4 seconds
L = 4;
k=(2*pi/L)*[0:n/2-1 -n/2:-1];
ks=fftshift(k);
%larg-ish t-slide cause else it matrix becomes to large to process
tslide = 0:0.5:L;

%% Getting Gaussian Spectogram matrices for train
Sgt_spec_train_matrix = zeros(270,length(tslide)*n);

width = 25;
for i = 1:270
    Sgt_spec_train = [];
    for j=1:length(tslide)
        g = exp(-width*(t - tslide(j)).^2); %gaussian filter
        Sg = g.*train_snippets(i,:); %filtered with gaussian
        Sgt = fft(Sg); %fft gaussian
        Sgt_spec_train = [Sgt_spec_train;abs(fftshift(Sgt))];
    end
    Sgt_spec_train_matrix(i,:) = ...
        reshape(Sgt_spec_train, 1, length(tslide)*n);
end
%% Getting Gaussian Spectogram matrices for test
Sgt_spec_test_matrix = zeros(27,length(tslide)*n);
for i = 1:27
    Sgt_spec_test = [];
    for j=1:length(tslide)
        g = exp(-width*(t - tslide(j)).^2); %gaussian filter
        Sg = g.*test_snippets(i,:); %filtered with gaussian
        Sgt = fft(Sg); %fft gaussian
        Sgt_spec_test = [Sgt_spec_test;abs(fftshift(Sgt))];
    end
    Sgt_spec_test_matrix(i,:) = ...
        reshape(Sgt_spec_test, 1, length(tslide)*n);
end
%% SVD
[U, S, V] = svd(Sgt_spec_train_matrix', 'econ');
%% Creating the projection matrix
projection_matrix = U.'*Sgt_spec_train_matrix.';
%% Creating test_projection vector
test_projection = U.'*Sgt_spec_test_matrix.';
```

```
%% Finding 3 avg projections
avg_projections = zeros(270,3);
for i = 1:3
    for j = 1:90
        avg_projections(:,i) = avg_projections(:,i) + ....
            projection_matrix(:,(i-1)*90 + j);
    end
    avg_projections(:,i) = avg_projections(:,i)/90;
end
%% Classifying test data
%for each row the expected out is [1,1,1,2,2,2,3,3,3]
indices = zeros(3,9);

for k = 1:3

    for i = 1:9

        min_index = 0;
        min_dist = Inf;

        for j = 1:3
            dist = sum(abs(avg_projections(:,j) - ...
                test_projection(:,i + (k-1)*9)));
            if dist < min_dist
                min_dist = dist;
                min_index = j;
            end
        end
        indices(k,i) = min_index;
    end

end
disp(indices);

%% saving spectogram train and test matrices as .csv
csvwrite("Spectogram_train_1.csv",Sgt_spec_train_matrix);
csvwrite("Spectogram_test_1.csv",Sgt_spec_test_matrix);
```

Python

Note: The spectograms uploaded are the ones saved from the matlab file using the csvwrite() command.

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier as KNN
```

```
X_train = np.genfromtxt('Spectogram_train_1.csv',delimiter = ',')
X_test = np.genfromtxt('Spectogram_test_1.csv',delimiter = ',')
#X_train = np.genfromtxt('Spectogram_train_2.csv',delimiter = ',')
#X_test = np.genfromtxt('Spectogram_test_2.csv',delimiter = ',')
#X_train = np.genfromtxt('Spectogram_train_3.csv',delimiter = ',')
#X_test = np.genfromtxt('Spectogram_test_3.csv',delimiter = ',')
print('Done loading data')

y_train = np.zeros((270,1));
y_test = np.zeros((27,1));

for i in range(3):
        for j in range(9):
                if j < 3:
                        y_test[9*i + j,0] = 1
                elif j < 6:
                        y_test[9*i + j,0] = 2
                else:
                        y_test[9*i + j,0] = 3


for j in range(270):
        if j < 90:
                y_train[j,0] = 1
        elif j < 180:
                y_train[j,0] = 2
        else:
                y_train[j,0] = 3


classifier =KNN(n_neighbors = 5, weights = 'distance', metric = 'euclidean')

classifier.fit(X_train, y_train.ravel())
train_accuracy = classifier.score(X_train, y_train.ravel())
print('Train accuracy: ' + str(train_accuracy))

predictions_for_test = classifier.predict(X_test)
print(predictions_for_test)
test_accuracy = classifier.score(X_test, y_test.ravel())
print('Test accuracy: ' + str(test_accuracy))
```