AMATH 582 Homework 4: Faces and Music Classification

Elise Bishoff
University of Washington
Department of Applied Mathematics
1410 NE Campus Parkway, Seattle WA 98105

Abstract

Singular Value Decomposition poses to be a powerful mathematical tool. In this paper we discuss how we use it to create eigenfaces and reconstruct images of faces based off the different eigenfaces. Also, we use it to perform music classification by taking the SVD of our spectrograms of 5 second clips of songs and then training our classification models on the signatures of the 5 second clips.

Sec. I. Introduction and Overview

Overall, we use Singular Value Decomposition to reconstruct images of faces and classify 5 second clips on song based off their band and music genre. We first discuss the necessary theoretical background needed to understand the problem at hand. We specifically explain Singular Value Decomposition, Support Vector Machines, K-Nearest Neighbors, and Naïve Bayes. Then, we go over the process of applying these methods to the specific problem at hand by exploring the algorithmic development and MATLAB code. This is followed by the computational results for our different scenarios. Next, we have our conclusion. This is followed by Appendices of MATLAB functions used and MATLAB code.

Sec. II. Theoretical Background

Singular Value Decomposition (SVD)

SVD is a very important mathematical tool that allows us to take a matrix and decompose it into three different components that each have specific applications and meaning. The SVD decomposition is illustrated in **Eq 2.1**.

$$A = U\Sigma V^* \quad (2.1)$$

From SVD we get $U \in \mathbb{C}^{m\times m}$ is unitary, $V \in \mathbb{C}^{n\times n}$ is unitary, $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal and $A \in \mathbb{C}^{m\times n}$. We can think of U and V^* as rotations and Σ as a stretch. So, we can think of A as a rotation, stretch(compress), and another rotation.

U is considered the left singular vectors and V^* is considered the right singular vectors. Σ is an ordered diagonal matrix of singular values that tells us what modes of A are most important in reconstructing the matrix A.

There are some important theorems that deal with SVD. We will briefly mention some of them. One of them is that every matrix A has a SVD. No matter what we can perform SVD on any matrix. This is an important point because there are a lot of decompositions that require some criteria to work, but for SVD, it always holds true for every matrix $A \in \mathbb{C}^{m\times n}$. The important differentiator is that we are using two orthogonal basis, U and V^*.

Now, the way to compute the SVD decomposition in practice is to first start with **Eq 2.2** and **Eq 2.3**. Now, since U and V are unitary then we know that $U^*U = I, V^*V = I$. This explains **Eq 2.4**.

$$A^TA = (U\Sigma V^*)^T(U\Sigma V^*) \quad (2.2)$$

$$= V\Sigma U^* U\Sigma V^* \quad (2.3)$$

$$= V\Sigma^2 V^* \quad (2.4)$$

This can be then be written into an eigenvalue problem as shown in **Eq 2.5**.

$$A^T AV = V\Sigma^2 \quad (2.5)$$

Now, with eigenvalue problems we must be careful because we don't always have a solution, but since $A^T A$ is a symmetric matrix, then we are guaranteed to have an eigenvalue decomposition. Thus, we can solve for V in this case. Similarly, we can solve for U by following a similar procedure with AA^T and we get the eigenvalue problem as shown in **Eq 2.6**.

$$AA^T U = U\Sigma^2 \quad (2.6)$$

This is how we extract the matrices U and V. To extract Σ we simply can find the eigenvalues of $A^T A$.

Machine Learning Algorithms: Support Vector Machines, K-Nearest Neighbors, and Naïve Bayes,

Support Vector Machines create hyperplanes that optimize the split between different classes or clusters of data. We can have linear SVM or nonlinear SVM which essentially the difference is between whether we create hyperplanes as our 'lines' to split the data or nonlinear classifiers as 'curves' to split our data. Support vectors are the points of data that touch the edge of our boundaries.

K-Nearest neighbors essentially takes all our training data and uses each labeled point within the model. Then depending on the value of k it classifies the new data based on the new data's k nearest neighbors based off distance. For example, if we use k-nearest neighbor where $k = 1$ then the training data is divided completely. Then, for every new point it finds it nearest neighbor and classifies it as that point. If $k = 5$ then the new data point is determined by a majority vote of the labels of the 5 points.

Naïve Bayes is a simple algorithm that relies on Bayes's theorem. It first computes the prior probability distributions of our training data. Then, when new data is received it estimates the label of the data based off that prior probability distribution.

Sec. III. Algorithm Implementation and Development

Overall, I knew the algorithm needed to be using SVD to find the eigenface for part 1. For part 2 I knew I needed to use SVD to decompose our spectrograms of the 5 second clips of songs and use the signature of the song to create a model that could accurate predict new data's label. I go over the important points of the code and discuss the algorithm implementation and development.

Part 1 Code

The important points from the code here are:

- In **lines 1-50** we are simply loading the data of the cropped and uncropped faces and then save the data as a .mat file to use for later use. In **lines 51-58** I simply take one picture from the cropped faces and one picture from the uncropped faces and plot it in grayscale to get a better understanding of what the pictures look like. In **line 60** I compute the SVD on the cropped faces.
- In **line 63-68** I create the plot of the normalized singular values to see which modes are most important for construction of the image. In **line 70** I create an array of values that tells me what

the percentage of each diagonal value has overall. In **lines 72-86** I plot the first 6 eigenfaces modes to get an understanding of the overall face structure from the pictures. In **lines 88-100** I create a loop to tell me the rank for the cropped images depending on the overall percentage of energy I want.

- In **lines 101-130** I create a plot that reconstructs the face depending on the rank given. In **lines 132-147** I create a plot that compares the original image with the reconstructed image of the face.
- In lines **150-209** we essentially do the same thing as above, but with uncropped images.

Part 2 Code

I have three different files for the three different cases, but they are essentially the same. I will explain case 1 and then give the differences between case 2 and 3 since case 2 and 3 are almost identical. Also, I created two functions that I explain as well.

From the test case 1 MATLAB code the important points are:

- In **lines 1-43** I read in the different song files. Each song I receive is in stereo mode so I convert it to mono mode by taking the mean between the two arrays after I take the transpose of the song. I calculate how long the song is in seconds as well. In **lines 44-69** I perform the two different functions I wrote on the songs. These functions are explained in greater detail below. Essentially, I split the song and find its spectrogram.
- In **lines 70-77** I combine the data together. Then I perform SVD on the songs. In **lines 81-90** I prep the data by find V' of the training set from the SVD and to project the test songs on the basis of the training set to find its V'. In **lines 91-120** I create an array that hold the true labels of the training and test data.
- In **lines 121-211** I create a KNN, SVM, and a Naïve Bayes model and classify the test data. Then I determine the accuracy of the test data depending on the number of modes used.
- In **lines 212-241** I create various plots to understand the data. I plot the singular values, and the accuracy of the three models together based off the number of modes used.

For the test case 2 and 3 MATLAB code the important points that are different from test case 1 are:

- In **lines 25-27** I chose to only use 400 seconds of each imported set of songs because I downloaded over an hour-long playlist of the artist or genre for each one. This took way too long to process so I extract the first 400 seconds.
- In **lines 53-64** I created the training and test set a little differently. I first put everything together and randomly split the data into 80% training data and 20% testing data.

From the function splitsong and spectransform the important points are:

- Splitsong: In **line 3** the function takes in the time of the song in seconds, the song's amplitude, and the sample rate. It spits out a matrix of 5 second clips of the song(s) pieced together. In **line 4** I create a placeholder for the 5 second clips. Then I determine how many 5 second clips can be extracted from the song in **line 6**. I then have a loop in **lines 12-18** that takes the song and splits it into 5 second clips.
- Spectransform: The function takes in a song or lots of songs in **line 3**. If it is lots of songs, then I loop through all the songs in **lines 5-9** and take the absolute value of their spectrogram in **line 6** and convert it into an vector in **line 7**. Then, I save that vector into a matrix where each column represents a different 5 second clip of a song.

Sec. IV. Computational Results

Part I: Yale Faces B

For the first part we were asked to perform SVD analysis on cropped and uncropped images. First, we are asked to interpret the meaning of $U, \Sigma,$ and V matrices for our SVD analysis. The U is essentially the basis for our images. This gives us a unique basis in being able to project all our images onto this basis. **Figure 4.1** are the first 6 modes (or basis) of our faces for our cropped and uncropped faces. We can see that they are the skeletons of the averages of all the faces. The cropped faces look like they have more defined facial features whereas the uncropped faces just look like outlines of a head. This is one difference between the cropped and uncropped images.

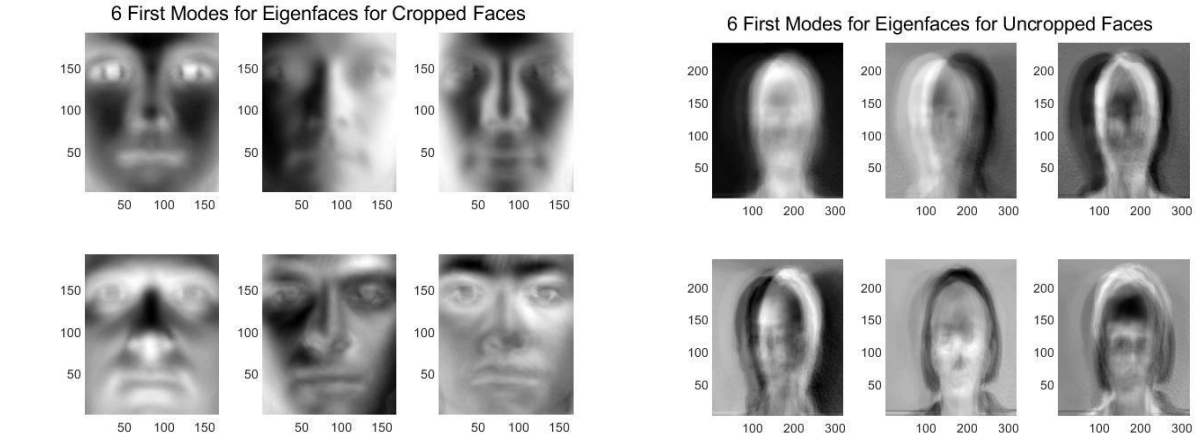

Figure 4.1 The first 6 modes of our faces. Left: Cropped Right: Uncropped

The Σ are the singular values and essentially are ordered from most important to least important. Essentially this tells us which modes are most important for reconstruction of our image. **Figure 4.2** are our Singular Values for our faces. V is the signature of each picture on the basis. It usually is a unique 'coordinate' plotted on the basis.

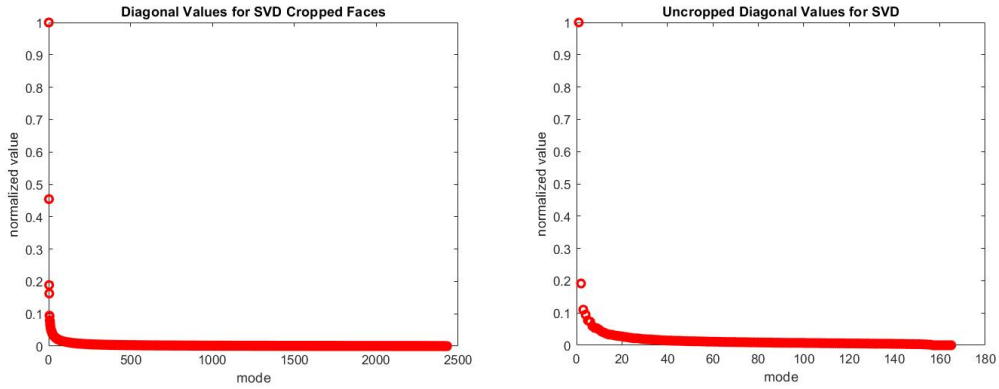

Figure 4.2 The Singular Values for our faces. Left: Cropped Right: Uncropped

Now, in order to get a good image reconstruction, I tested out different rank values. Rank really depends on what you consider to be a good reconstruction. For **Figure 4.3** top image I tried out different rank values depending on how much energy it explained or in other words how many normalized singular

values I needed to add up to get a certain percentage of the total sum of the singular values. **Figure 4.4** shows the number of modes used for each of the images in **Figure 4.3** top in order to get the reconstruction of the image. We can definitely see that the more modes used the better the image.

I decided that rank of 179 which explains 60% of it gives us a good reconstruction of the image which is shown in **Figure 4.3** bottom left. Also, I decided that rank of 94 for uncropped faces worked well to get a reconstruction which is also shown in **Figure 4.3** bottom right. Now, this shows another difference between the cropped and uncropped images. The cropped images I used 179 modes out of around 2400 to explain 60% of it whereas I used 94 modes out around 165 to explain 90% of the image and I still felt like it wasn't as nice of a reconstruction as the uncropped images. This just goes to show that having just the face works a lot better for the SVD than zooming out of the face.

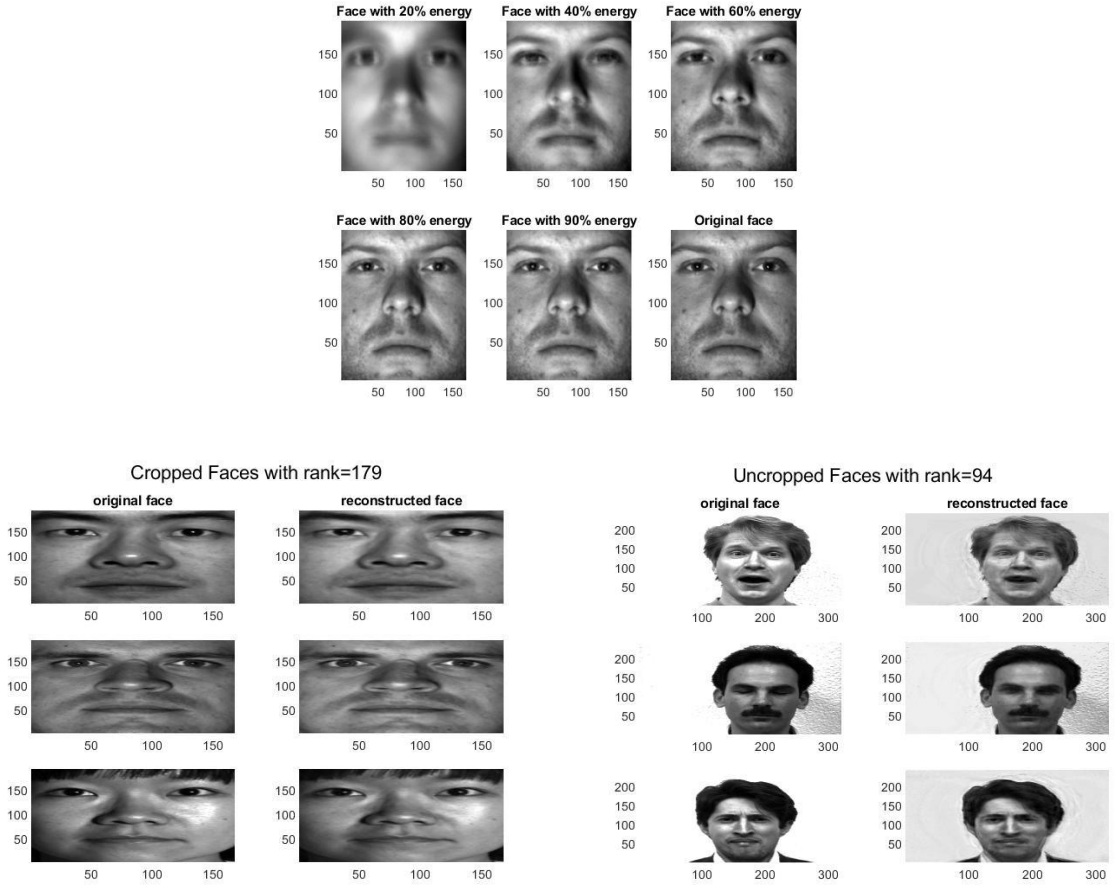

Figure 4.3 Top: Same cropped face reconstructed using different number of modes. Left Bottom: Original image compared to reconstructed cropped faces with 179 modes or rank=179. Right Bottom: Original image compared to reconstructed uncropped faces with rank=94.

Energy	20%	40%	60%	80%	90%
Rank	**5**	**47**	**179**	**605**	**1134**

Figure 4.4 Table of the number of modes needed(rank) in order to describe the energy level for Cropped Faces.

Part II: Music Classification

For part 2 we were asked to perform music classification based off three different test cases using SVD. The first case we were asked to perform music classification based off the band. I used Paper Kites, Daft Punk, and Beethoven. The second case we were asked to repeat the same experiment, but with three bands from the same music genre. I used Chopin, Liszt, and Shubert which are all classical composers from roughly the same time period. The third case we were asked to perform music classification based off the genre of music. I used country music, swing music, and classic rock. We use different classification algorithms to train our model and to predict the classes on the test set.

Figure 4.5 shows us the different normalized singular values from each test case. The first test case seems to have less prominent modes meaning that we can gather more information from less modes than the other cases. Whereas case 2 and case 3 seem to have a somewhat less 'exponential' cure and tend to be more linear.

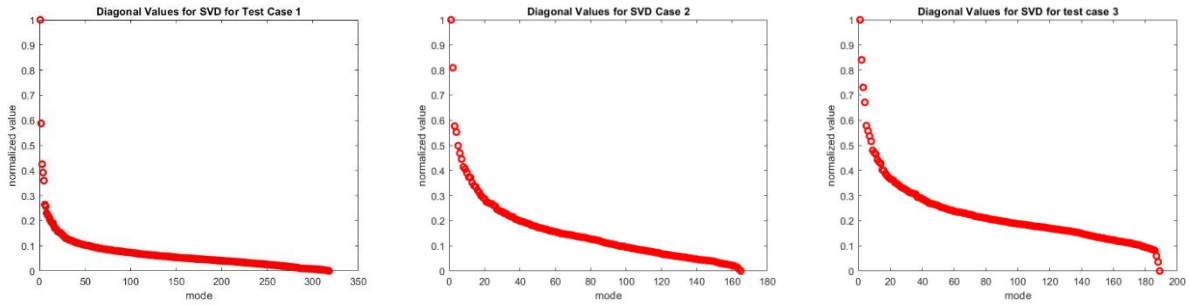

Figure 4.5 Singular Values for the different test cases. Left: Test Case 1 Middle: Test Case 2 Right: Test Case 3

Now, I trained with the signature of the songs (V') on three different classification algorithms, K nearest neighbor, Support Vector Machines, and Naïve Bayes. **Figure 4.6** shows the accuracy of classifying our test set based off the different algorithm used and based off how many modes were used in classification.

It was very interesting to see that depending on the type of classification task different classification algorithms worked better than others. For example, Naïve Bayes worked well on test case 1 with more modes but failed on test case 2 and test case 3. Also, SVM did well on test case 1, but flatlined for test case 2 and 3. I copied my code from test case 1 to do test case 2 and 3 so I figured it wasn't the code that was wrong. Then KNN seemed to perform consistently for all test cases. It especially performed well for test case 2 which I thought was odd since it was with three similar bands. All of this can be seen in **Figure 4.6**.

Overall, for test case 1 we were able to get the best accuracy with SVM with around 30 modes with around a 92% accuracy. KNN performed best for test case 2 with getting 100% accuracy with around 55 modes. This was very surprising. Test case 3 performed best with KNN with getting an overall accuracy of around 62% which was surprising as well because I thought it would perform similar to test case 1, but technically it performed the worst out of all the cases.

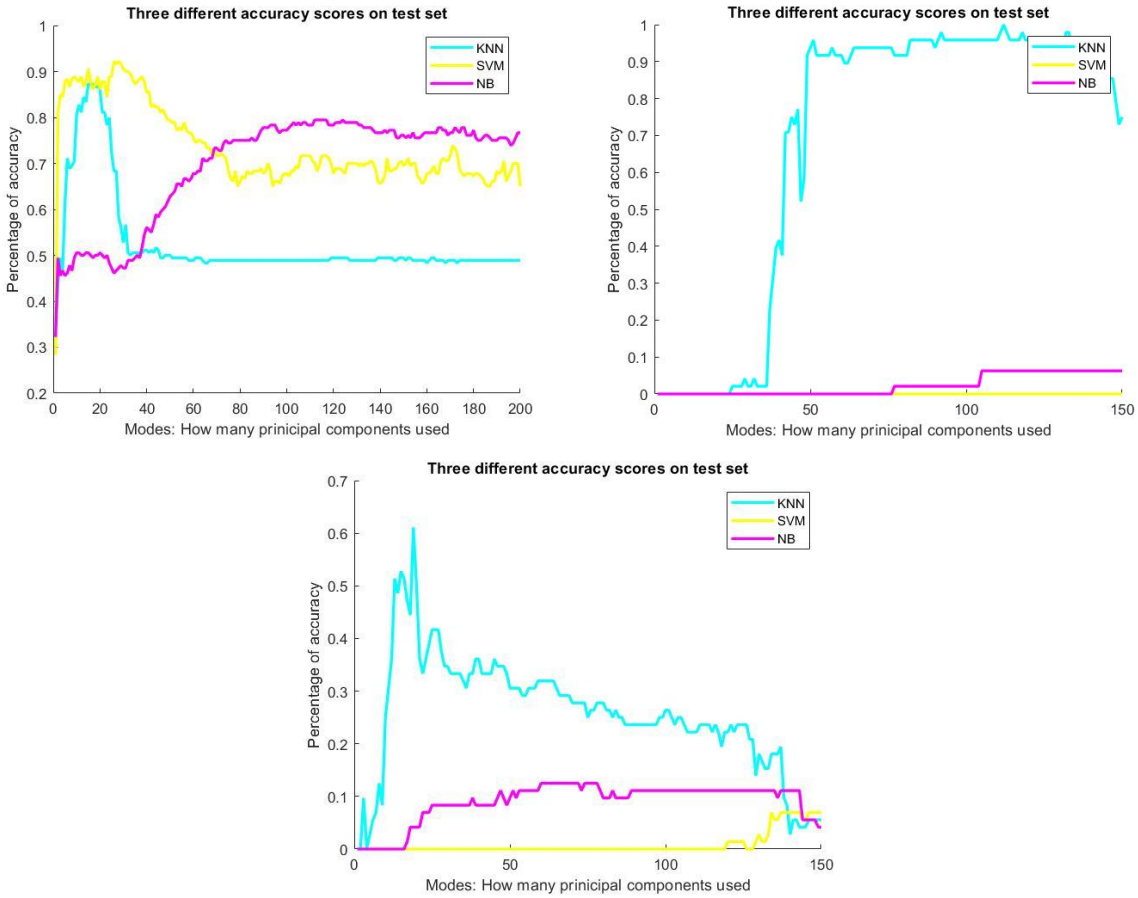

Figure 4.6 The accuracy scores on the test set based off the number of modes used for KNN, SVM, and Naïve Bayes

Sec. V. Summary and Conclusion

In conclusion, we can see that SVD is a powerful tool that allows us to analyze data in new and interesting ways. It allowed for us to decompose and reconstruct images of faces with less information. It allowed for us to construct the 'average' face of our dataset. It allowed for us to extract different music signatures and understand that each music signature is unique. It allows for us to classify music based off their band or music genre. These are all things that we were able to accomplish with SVD. Also, we were able to implement different classification algorithms that helped us understand that some algorithms work better than others depending on the problem. Overall, we were once again able to analyze data using powerful tools that allowed us to extrapolate new information from that data that we didn't see before.

Appendix A MATLAB functions used and brief implementation explanation

audioplayer(v,Fs): This creates an object for v where it uses Fs as the sample rate. This allowed us to play the audio recording of Handel's Messiah.

[p,Fs]=audioread('musicfile.wav'): This reads data from the file in a format that we can work with in MATLAB.

$model = fitcknn(input, input_y)$: This function allows us to put in our data and its corresponding classes and creates a model using KNN. We can then use this model to predict from other test sets.

$model = fitcecoc(input, input_y)$: This function allows us to put in our data and its corresponding classes and creates a model using SVM. We can then use this model to predict from other test sets.

$imshow(image)$: This displays the image or in our case a frame.

$labels = predict(model, test)$: This function allows us to predict the labels of our test set based off of the given model.

pcolor(0:ss:9,p,spc1.'),shading interp, colormap(hot): pcolor creates a pseudocolor plot. Shading determines how the plot will be colored and colormap defines the color scheme.

$y = reshape(s, [l1, l2])$: The reshape command allows us to reshape s into a matrix of l1 rows and l2 columns.

$s = spectogram(X)$: This returns the short-time Fourier transform of the signal X.

$[u, s, v] = svd(X)$: Performs a singular value decomposition on the given matrix, X. s is the diagonal matrix of singular values. Then u and v are either the left singular vectors and right singular vectors.

Appendix B MATLAB code

Part 1

```
%For this task we were asked to perform SVD analysis on a set of cropped
%and uncropped images.
%Here we are just reading in the cropped faces from my computer
A=dir('C:\Users\ebish\Documents\MATLAB\582 file 4\CroppedYale\*')
croppedfaces=[];
for j=3:length(A)
    d=fullfile(A(j).folder, A(j).name);
    getpictures=dir(d);
    for jj=3:length(getpictures)
        individualpics=fullfile(getpictures(jj).folder,getpictures(jj).name);
        indpicsfiles=double(imread(individualpics));
        makevector=indpicsfiles(:);
        croppedfaces=[croppedfaces makevector];
    end
end
%saving the original images dimesions and saving cropped faces so that way
%I don't have to run this code everytime and can just load these values
%since it takes forever to load.
mc=size(indpicsfiles,1);
```

```
nc=size(indpicsfiles,2);
save('croppedfaces.mat', 'croppedfaces');
save('mc.mat', 'mc');
save('nc.mat', 'nc');
%%
%Here we are just reading in the uncropped faces from my computer
B=dir('C:\Users\ebish\Documents\MATLAB\582 file 4\yalefaces\*')
uncropped=[];
for j=3:length(B)
    dd=fullfile(B(j).folder, B(j).name);
    unindpicsfiles=double(imread(dd));
    makevecun=unindpicsfiles(:);
    uncropped=[uncropped makevecun];
end
mu=size(unindpicsfiles,1);
nu=size(unindpicsfiles,2);
%saving the original images dimesions and saving cropped faces so that way
%I don't have to run this code everytime and can just load these values
%since it takes forever to load.
save('uncropped.mat', 'uncropped');
save('mu.mat', 'mu');
save('nu.mat', 'nu');
%%
%Load the values from above. Since I ran this code once all I have to do
%when i rerun this code is start here. It saves a bunch of time
 load('uncropped.mat')
 load('croppedfaces.mat')
 load('mc')
 load('nc')
 load('mu')
 load('nu')
%%
%this I am just looking at an image of the cropped and uncropped data
t=reshape(croppedfaces(:,10),[mc, nc]);
tu=reshape(uncropped(:,10),[mu,nu]);
figure
pcolor(flip(t)), shading interp, colormap(gray)
figure
pcolor(flip(tu)), shading interp, colormap(gray)
%%
%perfomr SVD on cropped images
[uc,sigmac,vc]=svd(croppedfaces, 'econ');
%%
%plot of the modes from most imporant to least important
figure
plot(diag(sigmac)/max(diag(sigmac)), 'ro', 'Linewidth',[2])
xlabel('mode')
ylabel('normalized value')
title('Diagonal Values for SVD Cropped Faces')
%This tells us how much each mode is contributing in terms of percentage
percentcropped=(diag(sigmac)/max(diag(sigmac)))/sum(diag(sigmac)/max(diag(sigmac)));

%first 6 eigenfaces
figure
```

```
subplot(2,3,1)
face1=reshape(uc(:,1),mc,nc); pcolor(flipud(face1)), shading interp, colormap(gray)
subplot(2,3,2)
face1=reshape(uc(:,2),mc,nc); pcolor(flipud(face1)), shading interp, colormap(gray)
subplot(2,3,3)
face1=reshape(uc(:,3),mc,nc); pcolor(flipud(face1)), shading interp, colormap(gray)
subplot(2,3,4)
face1=reshape(uc(:,4),mc,nc); pcolor(flipud(face1)), shading interp, colormap(gray)
subplot(2,3,5)
face1=reshape(uc(:,5),mc,nc); pcolor(flipud(face1)), shading interp, colormap(gray)
subplot(2,3,6)
face1=reshape(uc(:,6),mc,nc); pcolor(flipud(face1)), shading interp, colormap(gray)
sgtitle('6 First Modes for Eigenfaces for Cropped Faces')
%%
%this allows for me to find the number of modes needed to meet my energy
%threshold value of chosing. i chose to do 5 different threehold values
modestotal=[];
for threshold=[.2 .4 .6 .8 .9]
    sumperc=0;
    r=1;
    while sumperc<threshold
        sumperc=sumperc+percentcropped(r);
        r=r+1;
    end
    modestotal=[modestotal r];
end
%%
%reconstructing images based off a given number of modes
vcprime=vc';
modes=modestotal(3);
%%
%reconstructed images only with different number of modes
figure
reconstructedimages=uc*sigmac(:,1:modestotal(1))*vcprime(1:modestotal(1),:);
subplot(2,3,1)
face1=reshape(reconstructedimages(:,10),mc,nc); pcolor(flipud(face1)), shading interp,
colormap(gray)
title('Face with 20% energy')
reconstructedimages=uc*sigmac(:,1:modestotal(2))*vcprime(1:modestotal(2),:);
subplot(2,3,2)
face1=reshape(reconstructedimages(:,10),mc,nc); pcolor(flipud(face1)), shading interp,
colormap(gray)
title('Face with 40% energy')
reconstructedimages=uc*sigmac(:,1:modestotal(3))*vcprime(1:modestotal(3),:);
subplot(2,3,3)
face1=reshape(reconstructedimages(:,10),mc,nc); pcolor(flipud(face1)), shading interp,
colormap(gray)
title('Face with 60% energy')
reconstructedimages=uc*sigmac(:,1:modestotal(4))*vcprime(1:modestotal(4),:);
subplot(2,3,4)
face1=reshape(reconstructedimages(:,10),mc,nc); pcolor(flipud(face1)), shading interp,
colormap(gray)
title('Face with 80% energy')
reconstructedimages=uc*sigmac(:,1:modestotal(4))*vcprime(1:modestotal(4),:);
```

```
subplot(2,3,5)
face1=reshape(reconstructedimages(:,10),mc,nc); pcolor(flipud(face1)), shading interp,
colormap(gray)
title('Face with 90% energy')
subplot(2,3,6)
face1=reshape(croppedfaces(:,10),mc,nc); pcolor(flipud(face1)), shading interp,
colormap(gray)
title('Original face')
%%
%orignal image compared to reconstructed images
figure
subplot(3,2,1)
face1=reshape(croppedfaces(:,100),mc,nc); pcolor(flipud(face1)), shading interp,
colormap(gray)
title('original face')
subplot(3,2,2)
face1=reshape(reconstructedimages(:,100),mc,nc); pcolor(flipud(face1)), shading interp,
colormap(gray)
title('reconstructed face')
subplot(3,2,3)
face1=reshape(croppedfaces(:,200),mc,nc); pcolor(flipud(face1)), shading interp,
colormap(gray)
subplot(3,2,4)
face1=reshape(reconstructedimages(:,200),mc,nc); pcolor(flipud(face1)), shading interp,
colormap(gray)
subplot(3,2,5)
face1=reshape(croppedfaces(:,300),mc,nc); pcolor(flipud(face1)), shading interp,
colormap(gray)
subplot(3,2,6)
face1=reshape(reconstructedimages(:,300),mc,nc); pcolor(flipud(face1)), shading interp,
colormap(gray)
sgtitle('Cropped Faces with rank=179')

%% UnCropped Section
%taking SVD of uncropped images
[uu, sigmau, vu]=svd(uncropped, 'econ');
%%
%plot of the modes from most imporant to least important
figure
plot(diag(sigmau)/max(diag(sigmau)), 'ro', 'Linewidth',[2])
xlabel('mode')
ylabel('normalized value')
title('Uncropped Diagonal Values for SVD')
%This tells us how much each mode is contributing in terms of percentage
percentuncropped=(diag(sigmau)/max(diag(sigmau)))/sum(diag(sigmau)/max(diag(sigmau)));

%%
%The first 6 eigenfaces
figure
subplot(2,3,1)
face1=reshape(uu(:,1),mu,nu); pcolor(flipud(face1)), shading interp, colormap(gray)
subplot(2,3,2)
face1=reshape(uu(:,2),mu,nu); pcolor(flipud(face1)), shading interp, colormap(gray)
subplot(2,3,3)
```

```
face1=reshape(uu(:,3),mu,nu); pcolor(flipud(face1)), shading interp, colormap(gray)
subplot(2,3,4)
face1=reshape(uu(:,4),mu,nu); pcolor(flipud(face1)), shading interp, colormap(gray)
subplot(2,3,5)
face1=reshape(uu(:,5),mu,nu); pcolor(flipud(face1)), shading interp, colormap(gray)
subplot(2,3,6)
face1=reshape(uu(:,6),mu,nu); pcolor(flipud(face1)), shading interp, colormap(gray)
sgtitle('6 First Modes for Eigenfaces for Uncropped Faces')
%%
%this allows for me to find the number of modes needed to meet my energy
%threshold value of chosing.
sumpercu=0;
thresholdu=.9;
ru=1;
while sumpercu<thresholdu
    sumpercu=sumpercu+percentuncropped(ru);
    ru=ru+1;
end
%%reconstructed images only
vuprime=vu';
modes=ru;
reconstructedimagesu=uu*sigmau(:,1:modes)*vuprime(1:modes,:);
%%
%orignal compared to reconstructed
figure
subplot(3,2,1)
face1=reshape(uncropped(:,10),mu,nu); pcolor(flipud(face1)), shading interp, colormap(gray)
title('original face')
subplot(3,2,2)
face1=reshape(reconstructedimagesu(:,10),mu,nu); pcolor(flipud(face1)), shading interp,
colormap(gray)
title('reconstructed face')
subplot(3,2,3)
face1=reshape(uncropped(:,20),mu,nu); pcolor(flipud(face1)), shading interp, colormap(gray)
subplot(3,2,4)
face1=reshape(reconstructedimagesu(:,20),mu,nu); pcolor(flipud(face1)), shading interp,
colormap(gray)
subplot(3,2,5)
face1=reshape(uncropped(:,30),mu,nu); pcolor(flipud(face1)), shading interp, colormap(gray)
subplot(3,2,6)
face1=reshape(reconstructedimagesu(:,30),mu,nu); pcolor(flipud(face1)), shading interp,
colormap(gray)
sgtitle('Uncropped Faces with rank=94')
```

Part 2 Test Case 1

```
%for this assignment we were asked to create a classifier to correctly
%classify music files to their artist and genre from 5 second clips.
%Here I read in the music, create the files from the spectrogram of the
%songs and then use various classifiers to test the different effects on
%the different classifiers used
%Here we are just reading in the music files.
```

```
[p1,Fs1]=audioread('pkbloom2.wav'); %read the audiorecording
p1=p1'; %take the transpose
p1=mean(p1);
tr=length(p1)/Fs1;
[p2,Fs2]=audioread('pkfeatherstone.wav');
p2=p2';
p2=mean(p2);
tr2=length(p2)/Fs2;
[p3, Fs3]=audioread('dploseyourselftodance.wav');
p3=p3';
p3=mean(p3);
tr3=length(p3)/Fs3;
[p4,Fs4]=audioread('dpinstantcrush.wav');
p4=p4';
p4=mean(p4);
tr4=length(p4)/Fs4;
[p5, Fs5]=audioread('Beethovenfurelise.wav');
p5=p5';
p5=mean(p5);
tr5=length(p5)/Fs5;
[p6,Fs6]=audioread('Beethoven5thSymphony.wav');
p6=p6';
p6=mean(p6);
tr6=length(p6)/Fs6;
%reading in the test music files
[test1,test1_fs]=audioread('pkwoodland.wav');
test1=test1';
test1=mean(test1);
test1_tr=length(test1)/test1_fs;
[test2,test2_fs]=audioread('dpgetlucky.wav');
test2=test2';
test2=mean(test2);
test2_tr=length(test2)/test2_fs;
[test3, test3_fs]=audioread('MoonlightSonata.wav');
test3=test3';
test3=mean(test3);
test3_tr=length(test3)/test3_fs;
%%
%created a function that takes in the song and splits it into 5 seconds
%clips. See splitsong function to get better understanding of what it does
song1=splitsong(tr,p1,Fs1);
song2=splitsong(tr2,p2,Fs2);
song3=splitsong(tr3,p3,Fs3);
song4=splitsong(tr4,p4,Fs4);
song5=splitsong(tr5,p5,Fs5);
song6=splitsong(tr6,p6,Fs6);
%splitting the songs for the test songs
testsong1=splitsong(test1_tr,test1,test1_fs);
testsong2=splitsong(test2_tr,test2,test2_fs);
testsong3=splitsong(test3_tr,test3,test3_fs);
%finding the spectograms of each song and coverting it to a vector. To find
%more information on this function look at spectransform function.
specsong1=spectransform(song1);
specsong2=spectransform(song2);
```

```
specsong3=spectransform(song3);
specsong4=spectransform(song4);
specsong5=spectransform(song5);
specsong6=spectransform(song6);
%finding spectograms for test songs
test1_spec=spectransform(testsong1);
test2_spec=spectransform(testsong2);
test3_spec=spectransform(testsong3);
%%
%combining the songs into one matrix
allsongs=[specsong1 specsong2 specsong3 specsong4 specsong5 specsong6];
testsongs=[test1_spec test2_spec test3_spec];
%%
%finding the mean across rows and then computing the SVD
allsongs=allsongs-mean(allsongs,2);
[u, sigma, v]=svd(allsongs, 'econ');
%%
%finding the signature of the songs from the training dataset
%and then finding the signature of the test songs on the same basis
%as the training dataset.
vprime_train=v';
vprime_test=inv(sigma)*u'*testsongs;
%%
%projection of test songs onto the basis for the training songs
%found this as well becuase I wanted to see if it had
%any significant impact on the accuracy on the different models
Y=u'*testsongs;
input=u'*allsongs;
Y=Y';
input=input';
%here we create an array for the actual classes for each 5 second clip and
%for loop that simply labels the song with the correct band
actual_values=string(zeros(1, size(testsongs,2)));
predicted_values=zeros(1, size(testsongs,2));
for j=1:length(actual_values)
   if j<size(testsong1,1)+1
       actual_values(j)='paperkites';
   elseif j<size(testsong1,1)+size(testsong2,1)+1
       actual_values(j)='daftpunk';
    else
        actual_values(j)='beethoven';
    end
end
%%
%this does the same thing as above. IT creates an array with the actual
%bands that play each five second clip.
band1=size(song1,1)+size(song2,1);
band2=size(song3,1)+size(song4,1);
band3=size(song5,1)+size(song6,1);
actvals_train=string(zeros(1,size(allsongs,2)));
for j=1:length(actvals_train)
    if j<band1+1
        actvals_train(j)='paperkites';
    elseif j<band1+band2+1
```

```
        actvals_train(j)='daftpunk';
    else
        actvals_train(j)='beethoven';
    end
end
%%
%KNN: Uses k nearest neighbor to classify the data. First transformed the
%data into rows(where each row is the informaiton on a single 5 second
%clip). Then, ran the model over the first 200 modes and created an array
%that holds the number of correclty label data from the test set based off
%the modes. Then create a plot. A form of validation on the data.
vprime_train_normal=vprime_train';
vprime_test_normal=vprime_test';
actvalstrain_prime=actvals_train';
corrarray=[];
for modes=1:200
    model=fitcknn(vprime_train_normal(:,1:modes), actvals_train','NumNeighbors',1);
    label=predict(model,vprime_test_normal(:,1:modes));
    correctknn=0;
    for j=1:length(label)
        if label(j)==actual_values(j)
            correctknn=correctknn+1;
        end
    end
    corrarray=[corrarray correctknn];
end
knnperc=corrarray./length(label);
figure
plot(knnperc,'Linewidth',[2])
title('KNN:Accuracy on Test data based off of different modes')
xlabel('Modes: How many prinicipal components used')
ylabel('Percentage of accuracy')
%%
%tried the same thing as above out, but used u'data as the data used
%instead of just the v from the SVD
corrarray2=[];
for modes=1:200
    model=fitcknn(input(:,1:modes), actvals_train','NumNeighbors',1);
    label=predict(model,Y(:,1:modes));
    correctknn=0;
    for j=1:length(label)
        if label(j)==actual_values(j)
            correctknn=correctknn+1;
        end
    end
    corrarray2=[corrarray2 correctknn];
end
figure
plot(corrarray2,'Linewidth',[2])
%% SMV
%SVM: Uses support vector machines to classify the data.Ran the model over
%the first 200 modes and created an array
%that holds the number of correclty label data from the test set based off
%the modes. Then create a plot. A form of validation on the data.
```

```
corrarraysvm=[];
for modes=1:200
    modelsvm=fitcecoc(vprime_train_normal(:,1:modes), actvals_train');
    labelsvm=predict(modelsvm,vprime_test_normal(:,1:modes));
    correctsvm=0;
    for j=1:length(labelsvm)
        if labelsvm(j)==actual_values(j)
            correctsvm=correctsvm+1;
        end
    end
    corrarraysvm=[corrarraysvm correctsvm];
end
%percentage of accuracy based off the modes
svmperc=corrarraysvm./length(labelsvm);
figure
plot(svmperc,'Linewidth',[2])
title('SVM:Accuracy on Test data based off of different modes')
xlabel('Modes: How many prinicipal components used')
ylabel('Percentage of accuracy')
%% Naive Bayes
%NB: Uses Naive Bayes to classify the data.Ran the model over
%the first 200 modes and created an array
%that holds the number of correclty label data from the test set based off
%the modes. Then create a plot. A form of validation on the data.
corrarraynb=[];
for modes=1:200
    modelnb=fitcnb(vprime_train_normal(:,1:modes), actvals_train');
    labelnb=predict(modelnb,vprime_test_normal(:,1:modes));
    correctnb=0;
    for j=1:length(labelnb)
        if labelnb(j)==actual_values(j)
            correctnb=correctnb+1;
        end
    end
    corrarraynb=[corrarraynb correctnb];
end
%percentage of accuracy based off the modes
tperc=corrarraynb./length(labelnb);
figure
plot(tperc,'Linewidth',[2])
title('NB:Accuracy on Test data based off of different modes')
xlabel('Modes: How many prinicipal components used')
ylabel('Percentage of accuracy')
%%
%plot of the actual data normalized
figure
plot(allsongs(:,200), 'Linewidth',[2])
xlabel('time')
ylabel('y-value')
title('The different points extracted at each frame')
%%
%this looks at the diagonal values. This tells
%me what is going on in what direction. If it is big
%a lot is happening and if it is small then not
```

```
%much is happening.
figure
plot(diag(sigma)/max(diag(sigma)), 'ro', 'Linewidth',[2])
xlabel('mode')
ylabel('normalized value')
title('Diagonal Values for SVD for Test Case 1')
%This tells us how much each mode is contributing in terms of percentage
percentofenergy=(diag(sigma)/max(diag(sigma)))/sum(diag(sigma)/max(diag(sigma)));
%% Plot all plots togehter of the results
figure
hold on
plot(knnperc,'c', 'Linewidth',[2])
plot(svmperc,'y', 'Linewidth',[2])
plot(tperc,'m','Linewidth',[2])
hold off
legend('KNN','SVM','NB')
title('Three different accuracy scores on test set')
xlabel('Modes: How many prinicipal components used')
ylabel('Percentage of accuracy')
```

Part 2 Test Case 2

```
%for this assignment we were asked to create a classifier to correctly
%classify music files to their artist and genre from 5 second clips.
%Here I read in the music, create the files from the spectrogram of the
%songs and then use various classifiers to test the different effects on
%the different classifiers used
%Here we are just reading in the music files.
[p1,Fs1]=audioread('schubert.wav'); %read the audiorecording
p1=p1'; %take the transpose
p1=mean(p1);
tr=length(p1)/Fs1;
[p2,Fs2]=audioread('liszt.wav');
p2=p2';
p2=mean(p2);
tr2=length(p2)/Fs2;
[p3, Fs3]=audioread('chopin.wav');
p3=p3';
p3=mean(p3);
tr3=length(p3)/Fs3;
%%
%created a function that takes in the song and splits it into 5 seconds
%clips. See splitsong function to get better understanding of what it does
%Instead of using tr I hard code a value because the actual files is hours
%long and takes wayyyyyyy tooooo long to run so I truncate the files to
%only give me a smaller chunk of time.
song1=splitsong(400,p1,Fs1);
song2=splitsong(400,p2,Fs2);
song3=splitsong(400,p3,Fs3);
%%
%finding the spectograms of each song and coverting it to a vector. To find
```

```
%more information on this function look at spectransform function.
specsong1=spectransform(song1);
specsong2=spectransform(song2);
specsong3=spectransform(song3);
%this does the same thing as above. IT creates an array with the actual
%bands that play each five second clip.
band1=size(song1,1);
band2=size(song2,1);
band3=size(song3,1);
actvals_train=string(zeros(1,band1+band2+band3));
for j=1:length(actvals_train)
    if j<band1+1
        actvals_train(j)='schubert';
    elseif j<band1+band2+1
        actvals_train(j)='liszt';
    else
        actvals_train(j)='chopin';
    end
end
%%
%combining the songs into one matrix
allsongs=[specsong1 specsong2 specsong3];
%%
%In this section I simply split my data into training and test data
everything=[allsongs; actvals_train];
ind=randperm(size(everything,2));
everythingmixed=everything(:,ind);
%%
split=fix(.7*size(everything,2));
allsongs_almost=everything(:,1:split);
testsongs_almost=everything(:,split+1:end);
actual_values=testsongs_almost(end,:);
actvals_train=allsongs_almost(end,:);
allsongs=double(allsongs_almost(1:end-1,:));
testsongs=double(testsongs_almost(1:end-1,:));
%%
%finding the mean across rows and then computing the SVD
allsongs=allsongs-mean(allsongs,2);
[u, sigma, v]=svd(allsongs, 'econ');
%%
%here we create an array for the actual classes for each 5 second clip and
%for loop that simply labels the song with the correct band
% actual_values=string(zeros(1, size(testsongs,2)));
% predicted_values=zeros(1, size(testsongs,2));
% for j=1:length(actual_values)
%    if j<size(testsong1,1)+1
%        actual_values(j)='schubert';
%    elseif j<size(testsong1,1)+size(testsong2,1)+1
%        actual_values(j)='liszt';
%    else
%        actual_values(j)='chopin';
%    end
% end
%%
```

```
%finding the signature of the songs from the training dataset
%and then finding the signature of the test songs on the same basis
%as the training dataset.
vprime_train=v';
vprime_test=inv(sigma)*u'*testsongs;
%%
%projection of test songs onto the basis for the training songs
%found this as well becuase I wanted to see if it had
%any significant impact on the accuracy on the different models
Y=u'*testsongs;
input=u'*allsongs;
Y=Y';
input=input';
%%
%KNN: Uses k nearest neighbor to classify the data. First transformed the
%data into rows(where each row is the informaiton on a single 5 second
%clip). Then, ran the model over the first 200 modes and created an array
%that holds the number of correclty label data from the test set based off
%the modes. Then create a plot. A form of validation on the data.
vprime_train_normal=vprime_train';
vprime_test_normal=vprime_test';
actvalstrain_prime=actvals_train';
corrarray=[];
for modes=1:150
    model=fitcknn(vprime_train_normal(:,1:modes), actvals_train','NumNeighbors',1);
    label=predict(model,vprime_test_normal(:,1:modes));
    correctknn=0;
    for j=1:length(label)
        if label(j)==actual_values(j)
            correctknn=correctknn+1;
        end
    end
    corrarray=[corrarray correctknn];
end
knnperc=corrarray./length(label);
figure
plot(knnperc,'Linewidth',[2])
title('KNN:Accuracy on Test data based off of different modes')
xlabel('Modes: How many prinicipal components used')
ylabel('Percentage of accuracy')
%%
%tried the same thing as above out, but used u'data as the data used
%instead of just the v from the SVD
corrarray2=[];
for modes=1:150
    model=fitcknn(input(:,1:modes), actvals_train','NumNeighbors',1);
    label=predict(model,Y(:,1:modes));
    correctknn=0;
    for j=1:length(label)
        if label(j)==actual_values(j)
            correctknn=correctknn+1;
        end
    end
    corrarray2=[corrarray2 correctknn];
```

```
end
figure
plot(corrarray2,'Linewidth',[2])
%% SMV
%SVM: Uses support vector machines to classify the data.Ran the model over
%the first 200 modes and created an array
%that holds the number of correclty label data from the test set based off
%the modes. Then create a plot. A form of validation on the data.
corrarraysvm=[];
for modes=1:150
    modelsvm=fitcecoc(vprime_train_normal(:,1:modes), actvals_train');
    labelsvm=predict(modelsvm,vprime_test_normal(:,1:modes));
    correctsvm=0;
    for j=1:length(labelsvm)
        if labelsvm(j)==actual_values(j)
            correctsvm=correctsvm+1;
        end
    end
    corrarraysvm=[corrarraysvm correctsvm];
end
%percentage of accuracy based off the modes
svmperc=corrarraysvm./length(labelsvm);
figure
plot(svmperc,'Linewidth',[2])
title('SVM:Accuracy on Test data based off of different modes')
xlabel('Modes: How many prinicipal components used')
ylabel('Percentage of accuracy')
%% Naive Bayes
%NB: Uses Naive Bayes to classify the data.Ran the model over
%the first 200 modes and created an array
%that holds the number of correclty label data from the test set based off
%the modes. Then create a plot. A form of validation on the data.
corrarraynb=[];
for modes=1:150
    modelnb=fitcnb(vprime_train_normal(:,1:modes), actvals_train');
    labelnb=predict(modelnb,vprime_test_normal(:,1:modes));
    correctnb=0;
    for j=1:length(labelnb)
        if labelnb(j)==actual_values(j)
            correctnb=correctnb+1;
        end
    end
    corrarraynb=[corrarraynb correctnb];
end
%percentage of accuracy based off the modes
tperc=corrarraynb./length(labelnb);
figure
plot(tperc,'Linewidth',[2])
title('NB:Accuracy on Test data based off of different modes')
xlabel('Modes: How many prinicipal components used')
ylabel('Percentage of accuracy')
%%
%plot of the actual data normalized
figure
```

```
plot(allsongs(:,200), 'Linewidth',[2])
xlabel('time')
ylabel('y-value')
title('The different points extracted at each frame')
%%
%this looks at the diagonal values. This tells
%me what is going on in what direction. If it is big
%a lot is happening and if it is small then not
%much is happening.
figure
plot(diag(sigma)/max(diag(sigma)), 'ro', 'Linewidth',[2])
xlabel('mode')
ylabel('normalized value')
title('Diagonal Values for SVD Case 2')
%This tells us how much each mode is contributing in terms of percentage
percentofenergy=(diag(sigma)/max(diag(sigma)))/sum(diag(sigma)/max(diag(sigma)));
%% Plot all plots togehter of the results
figure
hold on
plot(knnperc,'c', 'Linewidth',[2])
plot(svmperc,'y', 'Linewidth',[2])
plot(tperc,'m','Linewidth',[2])
hold off
legend('KNN','SVM','NB')
title('Three different accuracy scores on test set')
xlabel('Modes: How many prinicipal components used')
ylabel('Percentage of accuracy')

```

Part 2 Test Case 3

```
%for this assignment we were asked to create a classifier to correctly
%classify music files to their artist and genre from 5 second clips.
%Here I read in the music, create the files from the spectrogram of the
%songs and then use various classifiers to test the different effects on
%the different classifiers used
%Here we are just reading in the music files.
[p1,Fs1]=audioread('rockclassic.wav'); %read the audiorecording
p1=p1'; %take the transpose
p1=mean(p1);
tr=length(p1)/Fs1;
[p2,Fs2]=audioread('country.wav');
p2=p2';
p2=mean(p2);
tr2=length(p2)/Fs2;
[p3, Fs3]=audioread('swing.wav');
p3=p3';
p3=mean(p3);
tr3=length(p3)/Fs3;
%%
%created a function that takes in the song and splits it into 5 seconds
%clips. See splitsong function to get better understanding of what it does
%Instead of using tr I hard code a value because the actual files is hours
```

```
%long and takes wayyyyyyy tooooo long to run so I truncate the files to
%only give me a smaller chunk of time.
song1=splitsong(400,p1,Fs1);
song2=splitsong(400,p2,Fs2);
song3=splitsong(400,p3,Fs3);
%%
%finding the spectograms of each song and coverting it to a vector. To find
%more information on this function look at spectransform function.
specsong1=spectransform(song1);
specsong2=spectransform(song2);
specsong3=spectransform(song3);
%this does the same thing as above. IT creates an array with the actual
%bands that play each five second clip.
band1=size(song1,1);
band2=size(song2,1);
band3=size(song3,1);
actvals_train=string(zeros(1,band1+band2+band3));
for j=1:length(actvals_train)
    if j<band1+1
        actvals_train(j)='rock';
    elseif j<band1+band2+1
        actvals_train(j)='liszt';
    else
        actvals_train(j)='chopin';
    end
end
%%
%combining the songs into one matrix
allsongs=[specsong1 specsong2 specsong3];
%%
%In this section I simply split my data into training and test data
everything=[allsongs; actvals_train];
ind=randperm(size(everything,2));
everythingmixed=everything(:,ind);
%%
split=fix(.8*size(everything,2));
allsongs_almost=everything(:,1:split);
testsongs_almost=everything(:,split+1:end);
actual_values=testsongs_almost(end,:);
actvals_train=allsongs_almost(end,:);
allsongs=double(allsongs_almost(1:end-1,:));
testsongs=double(testsongs_almost(1:end-1,:));
%%
%finding the mean across rows and then computing the SVD
allsongs=allsongs-mean(allsongs,2);
[u, sigma, v]=svd(allsongs, 'econ');
%%
%here we create an array for the actual classes for each 5 second clip and
%for loop that simply labels the song with the correct band
% actual_values=string(zeros(1, size(testsongs,2)));
% predicted_values=zeros(1, size(testsongs,2));
% for j=1:length(actual_values)
%    if j<size(testsong1,1)+1
%        actual_values(j)='schubert';
```

```
%    elseif j<size(testsong1,1)+size(testsong2,1)+1
%        actual_values(j)='liszt';
%    else
%        actual_values(j)='chopin';
%    end
% end
%%
%finding the signature of the songs from the training dataset
%and then finding the signature of the test songs on the same basis
%as the training dataset.
vprime_train=v';
vprime_test=inv(sigma)*u'*testsongs;
%%
%projection of test songs onto the basis for the training songs
%found this as well becuase I wanted to see if it had
%any significant impact on the accuracy on the different models
Y=u'*testsongs;
input=u'*allsongs;
Y=Y';
input=input';
%%
%KNN: Uses k nearest neighbor to classify the data. First transformed the
%data into rows(where each row is the informaiton on a single 5 second
%clip). Then, ran the model over the first 200 modes and created an array
%that holds the number of correclty label data from the test set based off
%the modes. Then create a plot. A form of validation on the data.
vprime_train_normal=vprime_train';
vprime_test_normal=vprime_test';
actvalstrain_prime=actvals_train';
corrarray=[];
for modes=1:150
    model=fitcknn(vprime_train_normal(:,1:modes), actvals_train','NumNeighbors',1);
    label=predict(model,vprime_test_normal(:,1:modes));
    correctknn=0;
    for j=1:length(label)
        if label(j)==actual_values(j)
            correctknn=correctknn+1;
        end
    end
    corrarray=[corrarray correctknn];
end
knnperc=corrarray./length(label);
figure
plot(knnperc,'Linewidth',[2])
title('KNN:Accuracy on Test data based off of different modes')
xlabel('Modes: How many prinicipal components used')
ylabel('Percentage of accuracy')
%%
%tried the same thing as above out, but used u'data as the data used
%instead of just the v from the SVD
corrarray2=[];
for modes=1:150
    model=fitcknn(input(:,1:modes), actvals_train','NumNeighbors',1);
    label=predict(model,Y(:,1:modes));
```

```
    correctknn=0;
    for j=1:length(label)
        if label(j)==actual_values(j)
            correctknn=correctknn+1;
        end
    end
    corrarray2=[corrarray2 correctknn];
end
figure
plot(corrarray2,'Linewidth',[2])
%% SMV
%SVM: Uses support vector machines to classify the data.Ran the model over
%the first 200 modes and created an array
%that holds the number of correclty label data from the test set based off
%the modes. Then create a plot. A form of validation on the data.
corrarraysvm=[];
for modes=1:150
    modelsvm=fitcecoc(vprime_train_normal(:,1:modes), actvals_train');
    labelsvm=predict(modelsvm,vprime_test_normal(:,1:modes));
    correctsvm=0;
    for j=1:length(labelsvm)
        if labelsvm(j)==actual_values(j)
            correctsvm=correctsvm+1;
        end
    end
    corrarraysvm=[corrarraysvm correctsvm];
end
%percentage of accuracy based off the modes
svmperc=corrarraysvm./length(labelsvm);
figure
plot(svmperc,'Linewidth',[2])
title('SVM:Accuracy on Test data based off of different modes')
xlabel('Modes: How many prinicipal components used')
ylabel('Percentage of accuracy')
%% Naive Bayes
%NB: Uses Naive Bayes to classify the data.Ran the model over
%the first 200 modes and created an array
%that holds the number of correclty label data from the test set based off
%the modes. Then create a plot. A form of validation on the data.
corrarraynb=[];
for modes=1:150
    modelnb=fitcnb(vprime_train_normal(:,1:modes), actvals_train');
    labelnb=predict(modelnb,vprime_test_normal(:,1:modes));
    correctnb=0;
    for j=1:length(labelnb)
        if labelnb(j)==actual_values(j)
            correctnb=correctnb+1;
        end
    end
    corrarraynb=[corrarraynb correctnb];
end
%percentage of accuracy based off the modes
tperc=corrarraynb./length(labelnb);
figure
```

```
plot(tperc,'Linewidth',[2])
title('NB:Accuracy on Test data based off of different modes')
xlabel('Modes: How many prinicipal components used')
ylabel('Percentage of accuracy')
%%
%plot of the actual data normalized
figure
plot(allsongs(:,150), 'Linewidth',[2])
xlabel('time')
ylabel('y-value')
title('The different points extracted at each frame')
%%
%this looks at the diagonal values. This tells
%me what is going on in what direction. If it is big
%a lot is happening and if it is small then not
%much is happening.
figure
plot(diag(sigma)/max(diag(sigma)), 'ro', 'Linewidth',[2])
xlabel('mode')
ylabel('normalized value')
title('Diagonal Values for SVD for test case 3')
%This tells us how much each mode is contributing in terms of percentage
percentofenergy=(diag(sigma)/max(diag(sigma)))/sum(diag(sigma)/max(diag(sigma)));
%% Plot all plots togehter of the results
figure
hold on
plot(knnperc,'c', 'Linewidth',[2])
plot(svmperc,'y', 'Linewidth',[2])
plot(tperc,'m','Linewidth',[2])
hold off
legend('KNN','SVM','NB')
title('Three different accuracy scores on test set')
xlabel('Modes: How many prinicipal components used')
ylabel('Percentage of accuracy')

```

Spectrogram Function

```
%this function takes a song and saves the absolute value spectogram
%information in a vector and appends it to an array called songlist
function songlist=spectransform(song)
songlist=[];
for j=1:size(song,1)
    s=abs(spectrogram(song(j,:)));
    s=reshape(s,[size(s,1)*size(s,2),1]);
    songlist=[songlist s];
end
```

SplitSong Function

```
%this function takes in the time of the song,tr, the song amplititude
%file,py, and the sample rate, Fsp and chops the song into 5 second clips
```

```
function songs=splitsong(tr,py,Fsp)
songs=[]; %file to hold the 5 second clips
%this tells me how many 5 second clips I can get from the song
totalsongclips=fix(tr/5);
%initial start and end for the first song
starttemp=1*Fsp;
endtemp=6*Fsp;
%loop that take a song and splits it into 5 second clips and then holds it
%in the array, songs.
for j=1:totalsongclips-1 %I take one less just to avoid complications with
    %how long we have at the end of the song
  segtemp=py(:,starttemp:endtemp);
  songs=[songs; segtemp];
  starttemp=starttemp+5*Fsp;
  endtemp=endtemp+5*Fsp;
end
```