

AMATH 482
Computational Methods for Data Analysis
Zach Shaw
Winter 2019
Homework 4

Due: March 8, 2019.

March 7, 2019

Abstract

In this homework, I analyzed the deconstruction of facial images using SVD and PCA. We had a cropped data set vs. and uncropped data set. I performed PCA on both to determine how it behaves on an ideal vs. a noisy data set. The next part of the homework asked me to train an algorithm to classify music. I investigated 3 different cases. After performing FFT on the samples, I achieved extremely high accuracy using 3 different classifiers.

1 Introduction and Overview

This homework is an introduction to machine learning. First, I am going to examine two sets of images. These images are provided from Yale, and the data sets contain "filtered" and "messy" data. The "filtered" data comes in the form of cropped pictures. Cropping them reduces the impact of the background on the SVD. The "messy" data comes in the form of uncropped pictures. These pictures still contain the background, and aren't as clean as the cropped. The goal is to examine the impact of the "filtered" data and the impact of the "noisy" data on the SVD and the accuracy in reconstruction.

Next, I am going to train 3 separate models to classify test data from a training set. The data for this set is going to be 5 second audio clips from songs. I will be testing three cases. These cases are: different musical genres and try to identify artist, Seattle based grunge bands and try to identify artist, and try to identify genre. After the data is created, I am going to train models using the Naive Bayes, Linear Discriminant Analysis (LDA), and K-nearest neighbors (KNN) algorithms. Then to cross validate, I will run the model 50 times to approximate the average accuracy of each method.

2 Theoretical Background

The details of SVD were discussed in a previous project. The short version is that it takes a matrix A and decomposes it into the matrices U, Σ, V . Σ is a diagonal matrix with the singular values on the main diagonal. These values indicate the principle components of our system. Using SVD, we can create a lower dimension representation of our system. This lower dimensional representation is useful for analyzing.

For the images in this project, I am going to examine the difference in the number of modes needed to recreate the original images from the cropped and uncropped pictures. Because the cropped image data set is larger than the uncropped image set, I am going to compare the percent of nodes required for both cropped and uncropped.

Moving on to the music, I am going to be doing something similar. to the faces. I am going to preform SVD on the data. But the data is going to the Fourier Transform of the music¹. This gives me access to the frequencies occurring in these 5 second clips. This is important because the combination of frequencies are unique across genres and artists.²After preforming this SVD, I can then take the components of V that correspond to each sample, and use that to train the algorithms mentioned in the introduction. One thing to note is that all of these clustering algorithms are "supervised" algorithms. This is an advantage over "unsupervised" algorithms. Supervised algorithms implies that your training data has labels. Whereas unsupervised algorithms do not have labels on the training data.

2.1 Naive Bayes

The Naive Bayes classification algorithm assumes that you have n classes. Then, you can set up the following:

$$\frac{P(1|x)}{P(0|x)} = \frac{P(x|1)P(1)}{P(x|0)P(0)} \quad (1)$$

The RHS is very easy to calculate, because we know $P(1), P(0)$ and we can calculate $P(x|1), P(x|2)$. Matlab has a built in function: `fitcnb(...)`. This function takes training data and labels and creates the Naive Bayes model.

2.2 Linear Discriminant Analysis

LDA and Naive Bayes share a lot of similarities. They are both built upon equation (1). However Naive Bayes assumes that the data is statistically independent. LDA does not assume this, but can only be applied to linear data sets, hence the name. LDA simply computes (1) and assigns the new data point to whichever category maximizes (1). LDA is the background of the `classify(...)` function within Matlab.

¹It was suggested to preform a spectrogram of the data and SVD that. However because the length of the audio clip is 5 seconds, it is essentially a spectrogram with very large Δt . This kept the data sizes much more manageable than preforming the spectrogram

²Fourier Transform just takes the data and expands it into sines and cosines. This gives quick access to the frequency makeup of the data.

2.3 KNN

This is quite a simple algorithm. Because it is a supervised algorithm, it knows the group of all of the nearest neighbors. When it gets new data, it simply looks for the closest neighbor. Once it identifies the closest neighbor, it checks the group of that neighbor and classifies the new point as the same group as the nearest neighbor. Matlab's implementation of this algorithm is `fitcknn(...)`.

3 Algorithm Implementation and Development

3.1 Part 1: Images

To develop this algorithm, I first needed to load the data. To do this, I made use of the `dir` function in Matlab, this function takes a string as input, and returns an object containing all of the items in that folder. Using lines 12-33, this loaded the first 13 files. The pictures provided skipped 14, so I had to split the loop to go from 1 to 13, and 15 to 39. While loading the data, I converted each image to grayscale. This makes it easier to manipulate the data.

The uncropped pictures were stuck in a .tar file, so I had to use the `untar` function in Matlab to extract. Then I used the same code for the cropped images to load the uncropped images. After loading the images and converting to grayscale, I reshaped into columns and stored them into my data matrix. The next step is to 0 the mean of the pictures. I accomplished this by subtracting out the "average" of the data set (Lines 86-97). Then I simply run SVD on the data/ $\sqrt{n-1}$ as covered in the course notes. I then plotted the singular values, and ran the code to determine the appropriate value of r . I did this for both the cropped and uncropped. Then I used line 138 to calculate the percentage of singular nodes required to accurately recreate the images.

3.2 Part 2: Music Clustering

I first had to load all of the musical data. In order to do this, I used the `dir` function like in Part 1 to get all of the files in each folder. I had to delete useless files like album art. Once each song was loaded, I had to create 5 second snippets. That is achieved with lines 204-209. Each column represented a 5 second snippet. I repeated this process for each genre/artist I was testing. After that, I FFT'd the data to get the frequencies. The FFT'd data needed to be transposed in order to compute SVD. Lines 239 to 241 ensure that the ordering doesn't matter when constructing the data for SVD. I then plot the singular values to see how I can reduce the dimension of this system. I then plot the first 2 V projections for all 3 or 4 data sets. These are the 2 dominant directions, and they depict almost completely unique clusters. This is good because it helps the clustering algorithms described earlier attain better accuracy. Next, I construct a random training and testing set from my SVD'd data. The purpose of the randomness is to assure that order doesn't matter. Then, I simply trained the algorithms, and predicted with my test data. After obtaining the prediction, I compared with the expected solution. I ran this 50 times to ensure that order truly didn't matter, and that my results were accurate.

This technique is called cross-validation. I did the exact same thing for the rest of the cases. I just changed the data loaded in.

4 Computational Results

4.1 Part 1: Images

4.1.1 Cropped

In order to accurately recreate the data, r had to encompass 82.85% of all of the singular modes.

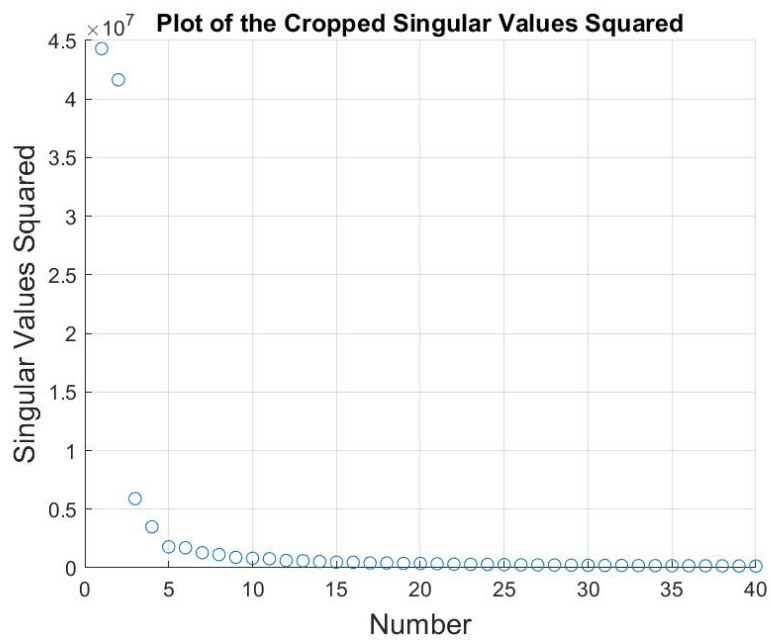


Figure 1: This shows the dominant modes for the cropped images.

Reconstructed Cropped Images $r=2015$

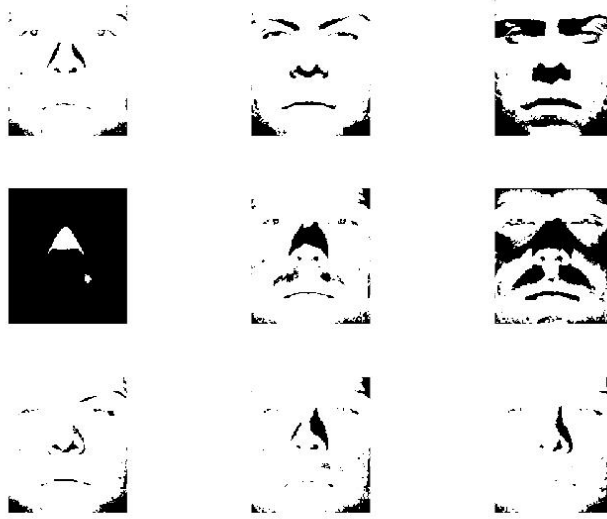


Figure 2: This shows the first 9 images reconstructed with $r = 118$.

Cropped Images Original

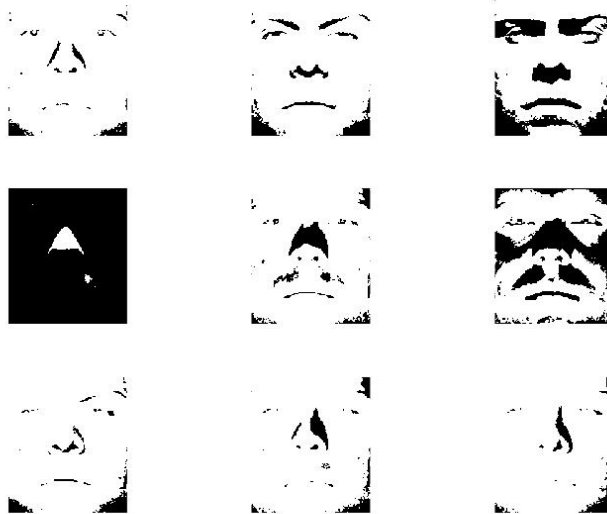


Figure 3: Compare figure 2 to figure 3, this verifies that this dimension reduction works.

4.1.2 Uncropped

In order to accurately recreate the data, r had to encompass 71.52% of all of the singular modes.

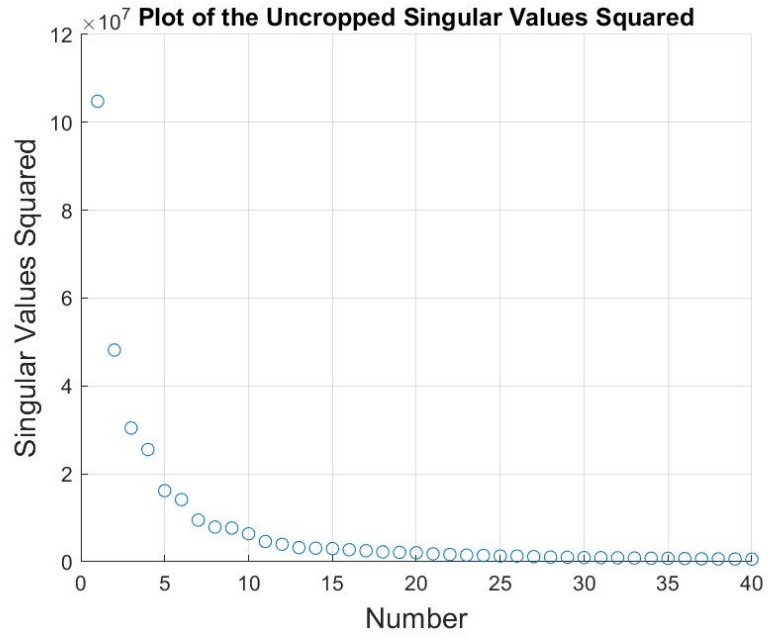


Figure 4: This shows the dominant modes for the uncropped images from Yale.



Figure 5: This shows the first 9 images reconstructed with $r = 118$.

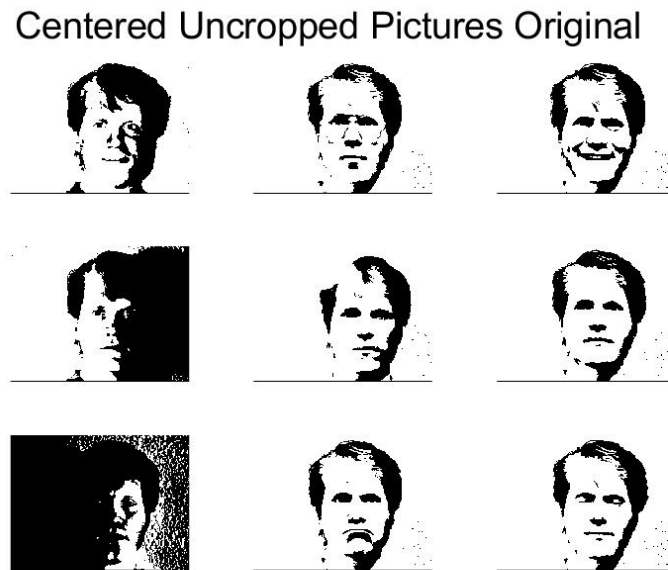


Figure 6: Compare figure 5 to figure 6 to verify that this dimension reduction works.

4.2 Part 2: Music Clustering

4.2.1 Case 1: Identify Artist of Separate Genres

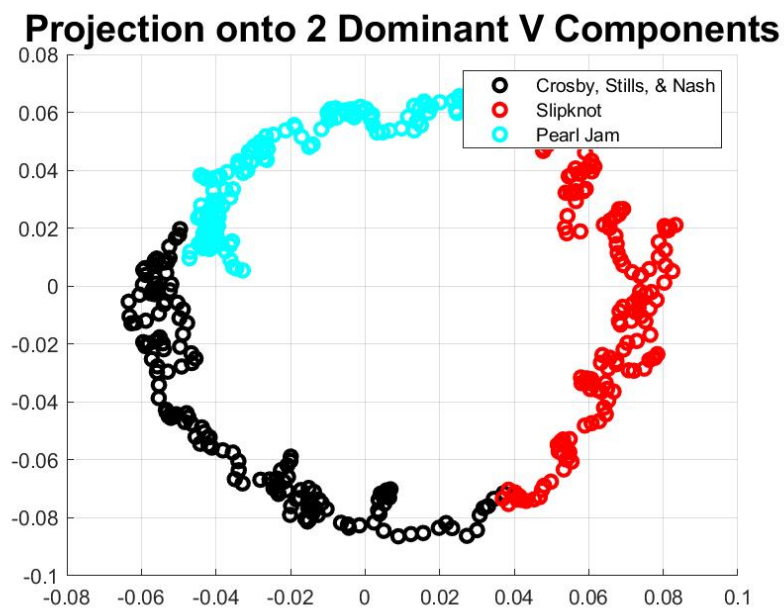


Figure: This projection shows that a clustering algorithm should be quite accurate in classifying artists of different genres

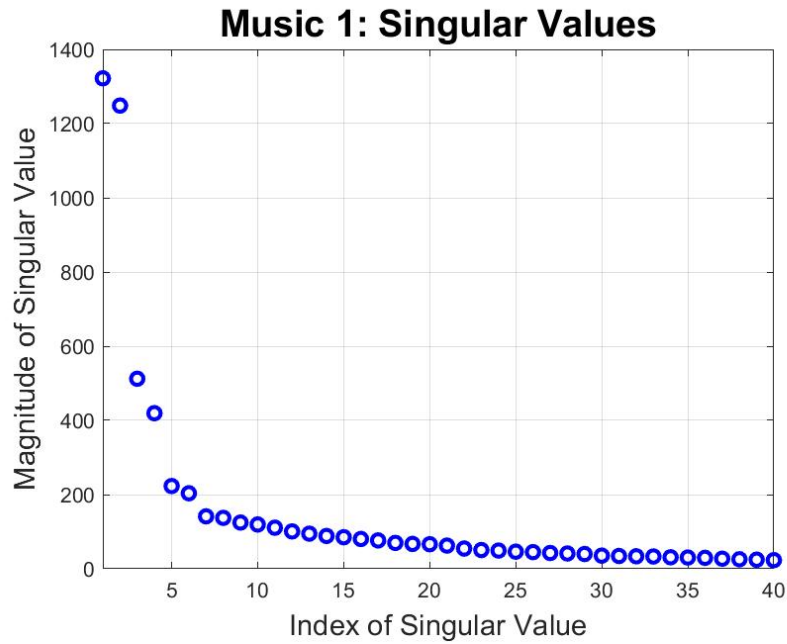


Figure: This shows the principle components for Case 1. We can reduce this matrix down to a 29 dimensional system.

4.2.2 Case 2: Identify Seattle Based Grunge Artists

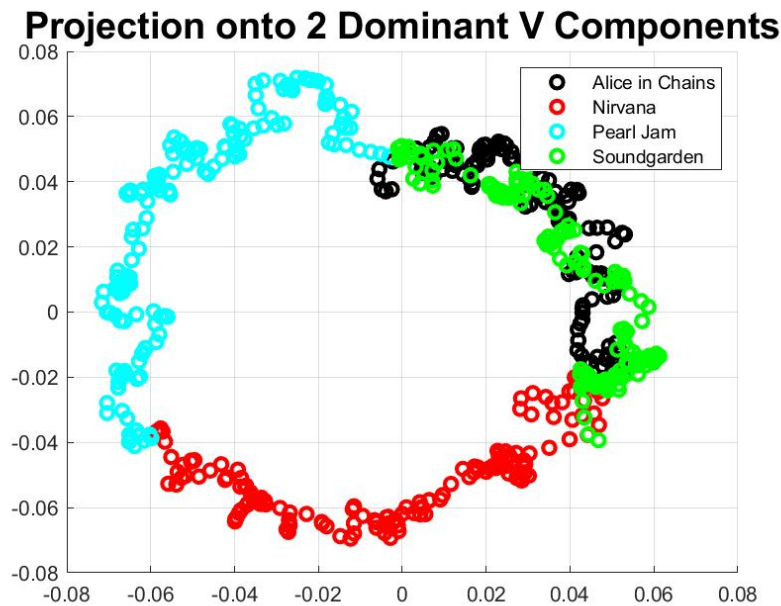


Figure: This projection shows that a clustering algorithm should be quite accurate in classifying Seattle based grunge bands. There may be an issue determining the difference between Alice in Chains and Soundgarden.

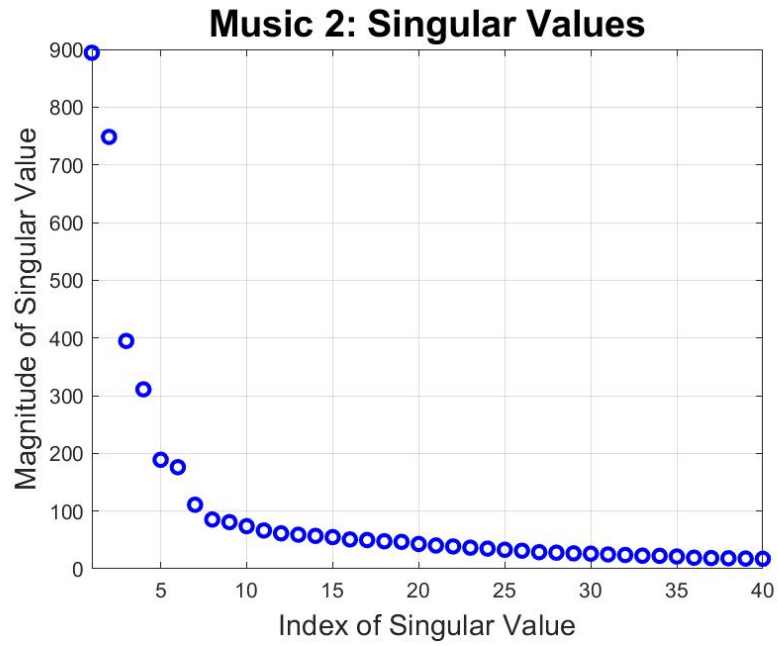


Figure: This shows the principle components for Case 1. We can reduce this matrix down to a 26 dimensional system.

4.2.3 Case 3: Identify Genre

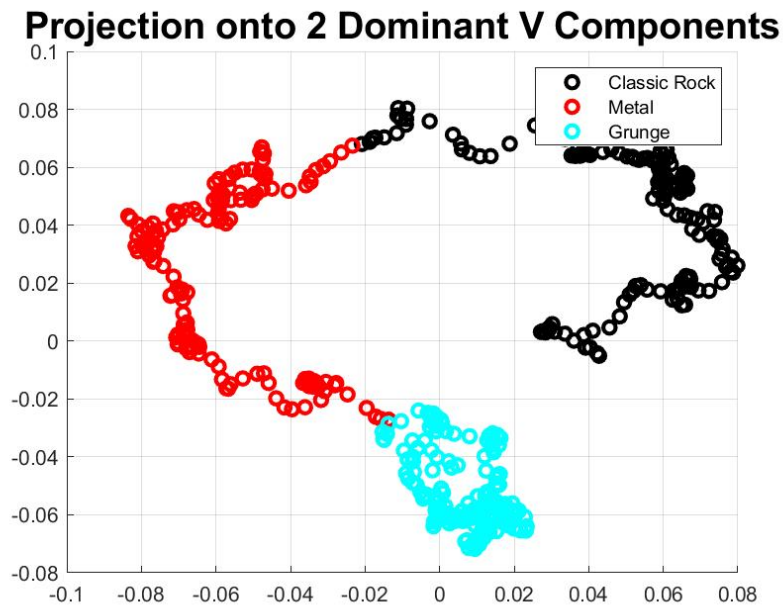


Figure: This projection shows that a clustering algorithm should be quite accurate in classifying different genres.

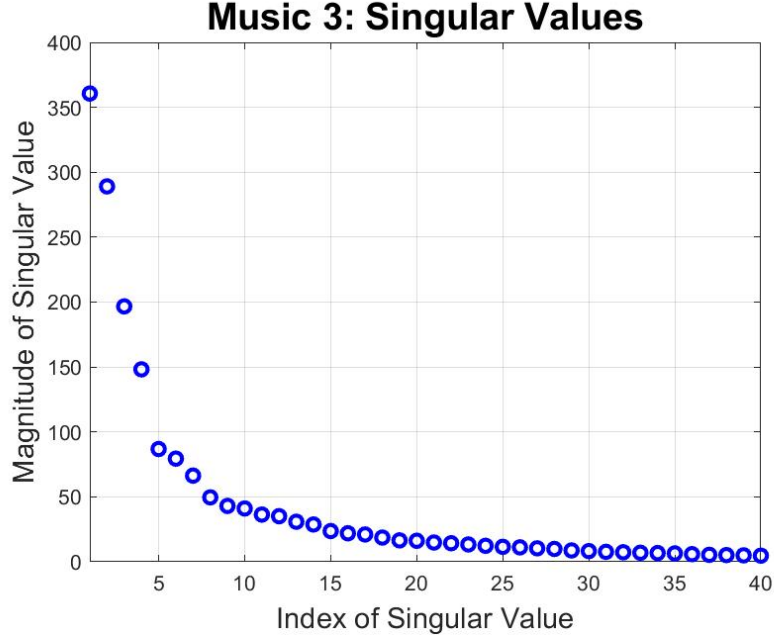


Figure: This shows the principle components for Case 1. We can reduce this matrix down to a 26 dimensional system.

5 Summary and Conclusions

5.1 Part 1: Images

From the figures above and the computational results, I can conclude that surprisingly, uncropped could be reduced more than the cropped photos. The way I rationalize this is due to the fact that the background across more of the photos is constant. This increases the covariance, and thus more of the energy is stored in earlier nodes. The interpretation of Σ is that it contains the value of the singular modes along its main diagonal. These values are the square root of the eigenvalues. Majorly non zero singular values indicate the the corresponding mode is important. U indicates the directions that actually matter, and V indicates how the image data projects onto the principle directions of U .

5.2 Part 2: Music Clustering

All three of the algorithms I implemented were extremely accurate. Using only 150 samples per artist or genre, I was able to achieve at least 98.57% accuracy. One of the important aspects of this homework I realized was randomly sampling all of the five second audio clips I made. In order to attain this randomness, I had to use lines like 239-242. This way I take a random sample of the clips I created. For Case 1, NB achieved an average accuracy of 98.87%, LDA achieved 99.33%, and KNN achieved 99.17%. For Case 2, NB achieved 99.05%, LDA achieved 99.43%, and KNN achieved 99.53%. And finally for Case 3, achieved 98.57%, LDA achieved 99.6%, and KNN achieved 99.67%. I cross validated these by taking 50 trials and randomizing the test data and the training data.

A MATLAB Functions and Implementation

Functions I used to implement the algorithm and why I used them:

zeros(...): This creates a matrix of zeros. Useful for pre-allocating the size that I want for my data.

['...','num2str(...)]: This was useful for string concatenation. This allowed me to load the data efficiently.

dir(...): This loads all of the files within a folder.

rgb2gray(...): This converts the image from color to grayscale.

uint8(...): This converts whatever is within the parentheses into image data. Used to display images after mathematical manipulation.

[U S V]=svd(...): Performs the svd and returns the matrices U, S, and V. This was used to perform the PCA.

x,ylabel(...): The x or y label function sets the label of the corresponding axis to the specified label.

title(...): The title command sets the title of the active figure.

print(...): This print function saves the active figure to the name specified by the 1st input. The file type is determined by the 2nd input.

untar(...): This was used to extract the uncropped face data.

diags(...): This grabs the diagonal elements of the matrix. I used this to grab the value of the singular values and plot them. **double(...)**: This was used to convert the uint8 variables loaded by the load function to double variables. Casting to double means I can perform mathematical operations on them.

size(...): This was used to get the size of the 4th component of the data (number of frames). This is what determines how long the for loops in my code run.

reshape(...): This was used to take the 2-D data and convert it down to 1-D vector. It was important for tracking the paint can.

floor(...): Because I was tracking the can with pixel data, it doesn't make sense for a pixel location to provide a non-integer value, so this required that the location be an integer value.

imshow(...): This is used to display the each picture. This helped with visualization.

imread(...): This was used to read each of the pictures and store them in Matlab.

plot(...): Plot the provided data

scatter(...): Create a scatter plot from the given data.

fitcknn(...): Fit a k-nearest neighbor model to the data. Provide the training labels.

fitcnb(...): Fit a Naive Bayes model to the data. Provide the training labels.

classify(...): Perform LDA on the training data and the training labels. Provide test data to generate a prediction.

B MATLAB Code

```
1 %% Load Cropped Images
2 clear all, close all; clc
3 %Create variables
4 m=192;
5 n=168;
6 A=[];
7 avg=zeros(m*n,1);
8 count=0;
9 picPerPerson=64;
10
11 %Loop through first 13 files
12 for z=1:13
13     s=['yaleB',num2str(z,'%02d')];
14     list=dir(s);
15     list=list(3:end);
16     %load pictures
17     for j=1:64
18         f=list(j);
19         u=imread([s,'\ ',f.name]);
20         imshow(u)
21         %Convert to grayscale if not already
22         if(size(u,3)==1)
23             M=double(u);
24         else
25             M=double(rgb2gray(u));
26         end
27         pause(0.1);
28         R = reshape(M,size(M,1)*size(M,2),1);
29         A = [A,R];
```

```

30     avg = avg + R;
31     count = count + 1;
32     end
33 end
34 %% Load the second half of images
35 %Start at 15 because files skipped 14. do the same as above
36 for z=15:39
37     s=['yaleB',num2str(z,'%02d')];
38     list=dir(s);
39     list=list(3:end);
40     for j=1:64
41         f=list(j);
42         u=imread([s,'\ ',f.name]);
43         imshow(u)
44         if(size(u,3)==1)
45             M=double(u);
46         else
47             M=double(rgb2gray(u));
48         end
49         pause(0.1);
50         R = reshape(M,size(M,1)*size(M,2),1);
51         A = [A,R];
52         avg = avg + R;
53         count = count + 1;
54     end
55 end
56
57 %% Load Uncropped
58 s='yalefaces';
59 list=dir(s);
60 list=list(3:end);
61 %Create uncropped variables
62 m_uc=243;
63 n_uc=320;
64 A_uc=[];
65 avg_uc=zeros(m_uc*n_uc,1);
66 count_uc=0;
67 %loop through all the files and store them
68 for j=1:165
69     f=list(j);
70     u=imread([s,'\ ',f.name]);
71     imshow(u)
72     if(size(u,3)==1)
73         M=double(u);
74     else
75         M=double(rgb2gray(u));

```

```

76         end
77         pause(0.1);
78         R_uc = reshape(M,size(M,1)*size(M,2),1);
79         A_uc = [A_uc,R_uc];
80         avg_uc = avg_uc + R_uc;
81         count_uc = count_uc + 1;
82     end
83
84
85 %% Calculate Average Face
86 avg=avg/count; %sum divided by the number
87 avgTS = uint8(reshape(avg,m,n));
88 figure(1), imshow(avgTS); %show the average face
89 %% Zero about the "Origin"
90 N=size(A,2);
91 %Subtract the average from each data point to give a mean of
    0
92 for j=1:N
93     A(:,j) = A(:,j) - avg;
94     R = reshape(A(:,j),m,n);
95     imshow(R);
96     pause(.1);
97 end
98
99 %% Compute the SVD, Visualize, and Recreate
100 clear VR V U S UR SR
101 %SVD
102 [U S V]=svd(A/sqrt(count-1),0);
103 %Examine the important nodes
104 figure(1)
105 scatter(1:2432,diag(S).^2)
106 xlim([0 40])
107 grid on
108 ylabel('Singular Values Squared','FontSize',14)
109 xlabel('Number','FontSize',14)
110 title('Plot of the Cropped Singular Values Squared','FontSize
    ',12)
111 print('C_SV','-djpeg')
112 r=2015;
113 %Dimension Reduction
114 UR(:,1:r)=U(:,1:r);
115 SR(1:r,1:r)=S(1:r,1:r);
116 V=V';
117 VR(1:r,:)=V(1:r,:);
118 %Recreate the original data
119 L=UR*SR*VR;

```

```

120 %%
121 figure(2)
122 subplot(2,1,1)
123 %Compare the original data
124 figure(2)
125 sgtitle('Reconstructed Cropped Images r=2015','FontSize',18)
126 for z=1:9
127     subplot(3,3,z)
128     imshow(uint8(25000*reshape(L(:,z),m,n))))
129 end
130 print('C_RC','-djpeg');
131 figure(3)
132 sgtitle('Cropped Images Original','FontSize',18)
133 for z=1:9
134     subplot(3,3,z)
135     imshow(uint8(25000*reshape(A(:,z),m,n))))
136 end
137 print('C_D','-djpeg');
138 recon_perc=(r/(length(S)))*100
139 %% Calculate Average Uncropped Face
140 %Do the same things for the cropped faces
141 avg_uc=avg_uc/count_uc;
142 aveUC=uint8(reshape(avg_uc,m_uc,n_uc));
143 figure(1),imshow(aveUC)
144 %% Center the Uncropped faces
145 N_uc=size(A_uc,2);
146 for j=1:N_uc
147     A_uc(:,j)=A_uc(:,j)-avg_uc;
148     R_uc=reshape(A_uc(:,j),m_uc,n_uc);
149     imshow(R_uc);
150     pause(.1);
151 end
152 %% Compute the Uncropped SVD, Visualize Modes, and
    Reconstruct
153 clear u_uc s_uc v_uc U_uc V_uc S_uc
154 [u_uc s_uc v_uc]=svd(A_uc/sqrt(N_uc-1),0);
155 scatter(1:165,diag(s_uc).^2)
156 xlim([0 40])
157 grid on
158 ylabel('Singular Values Squared','FontSize',14)
159 xlabel('Number','FontSize',14)
160 title('Plot of the Uncropped Singular Values Squared','
    FontSize',12)
161 print('UC_SV','-djpeg')
162 r_uc=118;
163 U_uc(:,1:r_uc)=u_uc(:,1:r_uc);

```

```

164 S_uc(1:r_uc,1:r_uc)=s_uc(1:r_uc,1:r_uc);
165 v_uc=v_uc';
166 V_uc(1:r_uc,:)=v_uc(1:r_uc,:);
167 L_uc=U_uc*S_uc*V_uc;
168 figure(2)
169 sgtitle('Uncropped Pictures Recreated r=118','FontSize',18)
170 for z=1:9
171     subplot(3,3,z)
172     imshow(uint8(25000*reshape(L_uc(:,z),m_uc,n_uc))))
173 end
174 print('UC_RC','-djpeg');
175 figure(3)
176 sgtitle('Centered Uncropped Pictures Original','FontSize',18)
177 for z=1:9
178     subplot(3,3,z)
179     imshow(uint8(25000*reshape(A_uc(:,z),m_uc,n_uc))))
180 end
181 print('UC_D','-djpeg');
182 ruc_perc=r_uc/length(diag(s_uc))*100
183 %% Music Classification Part 1: Band Classification
184 %% Load the Data
185 clear all;close all; clc
186 %Load Songs in directory
187 s='Classic Rock\Crosby, Stills & Nash\The Greatest Hits';
188 g='Metal\Antennas to Hell The Best of Slipknot\';
189 l='Seattle_Grunge\Pearl Jam\Rearviewmirror (Pearl Jam
    Greatest Hits 1991-2003) Disc 1\';
190 %Create directory
191 list1=dir(s);
192 list2=dir(g);
193 list3=dir(l);
194 %Remove useless files
195 list1=list1(3:end-5);
196 list2=list2(3:end);
197 list3=list3(3:end-8);
198 P=[];
199 m=[];
200 %Load first 2 songs of Classic Rock
201 for z=1:2
202     [x Fs]=audioread([s,'\ ',list1(z).name]);
203     Xmono=(x(:,1)+x(:,2))/2;
204     for j=1:floor(length(Xmono)/Fs)-5
205         start=j*Fs;
206         End=(j+5)*Fs;
207         m=Xmono(start:End);
208         P=[P,m];

```



```

209     end
210 end
211 %Load first two songs of Metal
212 Metal=[];
213 for z=1:2
214     [x Fs]=audioread([g,list2(z).name]);
215     Xmono=(x(:,1)+x(:,2))/2;
216     for j=1:floor(length(Xmono)/Fs)-5
217         start=j*Fs;
218         End=(j+5)*Fs;
219         m=Xmono(start:End);
220         Metal=[Metal,m];
221     end
222 end
223 %Load first two songs of Grunge
224 Grunge=[];
225 for z=1:2
226     [x Fs]=audioread([l,'\ ',list3(z).name]);
227     Xmono=(x(:,1)+x(:,2))/2;
228     for j=1:floor(length(Xmono)/Fs)-5
229         start=j*Fs;
230         End=(j+5)*Fs;
231         m=Xmono(start:End);
232         Grunge=[Grunge,m];
233     end
234 end
235 %% FFT Data
236 clc, clear song_fft song_Spec
237 %Set a number of 5 second snippets to test
238 samp=150;
239 g1=randperm(samp);
240 g2=randperm(samp);
241 g3=randperm(samp);
242 X=[P(:,g1(1:samp)) Metal(:,1:g2(1:samp)) Grunge(:,1:g3(1:samp))];
243
244 n = length(X);
245 t = (1:n)/Fs;
246 song_fft=[];
247 %FFT the data
248 for i = 1:3*samp
249     song_fft=[song_fft;real(fftshift(fft(X(i,:))))];
250 end
251 %% Compute SVD
252 clear U S V KNN_AA KNN_OA NB_OA NB_AA DA_AA DA_OA
253 %FFT needs to be transposed

```

```

254 song_fft=song_fft';
255 [U S V]=svd(real(song_fft)/sqrt(3*samp-1),0);
256 figure(1)
257 plot(diag(S).^2,'bo','Linewidth',2)
258 xlim([1 40]);
259 grid on
260 xlabel('Index of Singular Value','FontSize',14)
261 ylabel('Magnitude of Singular Value','FontSize',14)
262 title('Music 1: Singular Values','FontSize',18)
263 print('MC1_Sing','-djpeg')
264 figure(2)
265 hold on
266 plot(V(1:samp,1),V(1:samp,2),'ko','Linewidth',2)
267 plot(V(samp+1:2*samp,1),V(samp+1:2*samp,2),'ro','Linewidth'
    ,2)
268 plot(V(2*samp+1:3*samp,1),V(2*samp+1:3*samp,2),'co','
    Linewidth',2)
269 hold off
270 grid on
271 title('Projection onto 2 Dominant V Components','FontSize'
    ,18)
272 legend('Crosby, Stills, & Nash','Slipknot','Pearl Jam')
273 print('MC1_Proj','-djpeg')
274 train=samp-20;
275 test=samp-train;
276 %% Train and Predict Model
277 clear q1 q2 q3 xClassic xMetal xGrunge xtrain xtest ctrain nb
    pre solution, close all
278 for z=1:50
279     %Create random permutations
280     q1=randperm(samp);
281     q2=randperm(samp);
282     q3=randperm(samp);
283     %Create the actual solution (for comparing to prediction)
284     solution=[ones(test,1);2*ones(test,1);3*ones(test,1)];
285
286     %Create the training data
287     xClassic=V(1:samp,1:29);
288     xMetal=V(samp+1:2*samp,1:29);
289     xGrunge=V(2*samp+1:3*samp,1:29 );
290
291     %Put everything together into train and test
292     xtrain=[xClassic(q1(1:train),:); xMetal(q2(1:train),:);
        xGrunge(q3(1:train),:)];
293     xtest=[xClassic(q1(train+1:end),:); xMetal(q2(train+1:end
        ),:);xGrunge(q3(train+1:end),:)];

```

```

294     ntest=size(xtest,1);
295     % Naive Bayes
296     ctrain=[ones(train,1); 2*ones(train,1);3*ones(train,1)];
297     nb=fitcnb(xtrain,ctrain);
298     % N-B Prediction
299     pre=nb.predict(xtest);
300     bar(pre)
301     NB_acc=pre==solution;
302     NB_OA(z)=(sum(NB_acc)/ntest)*100;
303     % Discriminant Analysis
304     SVM=classify(xtest,xtrain,ctrain);
305     % DA Prediction
306     bar(SVM)
307     DA_acc=SVM==solution;
308     DA_OA(z)=(sum(DA_acc)/ntest)*100;
309     % K Nearest Neighbor
310     idx=fitcknn(xtrain,ctrain);
311     % KNN Predict
312     pre_knn=idx.predict(xtest);
313     bar(pre_knn)
314     KNN_acc=pre_knn==solution;
315     KNN_OA(z)=(sum(KNN_acc)/ntest)*100;
316 end
317 nave=size(NB_OA,2);
318 NB_AA=sum(NB_OA)/nave
319 DA_AA=sum(DA_OA)/nave
320 KNN_AA=sum(KNN_OA)/nave
321 %% Music Part 2: The Case for Seattle
322 %% Load the Data
323 clear all;close all; clc
324 s='Seattle_Grunge\Alice in Chains\Alice in Chains1\';
325 g='Seattle_Grunge\Nirvana\Nevermind\';
326 l='Seattle_Grunge\Pearl Jam\Rearviewmirror (Pearl Jam
    Greatest Hits 1991-2003) Disc 1\';
327 d='Seattle_Grunge\Soundgarden\A-Sides\';
328 list1=dir(s);
329 list2=dir(g);
330 list3=dir(l);
331 list4=dir(d);
332 list1=list1(3:end);
333 list2=list2(3:end-5);
334 list3=list3(3:end-8);
335 list4=list4(3:end-6);
336 m=[];
337 %Load first 2 songs of Alice in Chains
338 AIC=[];

```

```

339 for z=1:2
340     [x Fs]=audioread([s,list1(z).name]);
341     Xmono=(x(:,1)+x(:,2))/2;
342     for j=1:floor(length(Xmono)/Fs)-5
343         start=j*Fs;
344         End=(j+5)*Fs;
345         m=Xmono(start:End);
346         AIC=[AIC,m];
347     end
348 end
349 %Load first two songs of Nirvana
350 Nirv=[];
351 for z=1:2
352     [x Fs]=audioread([g,list2(z).name]);
353     Xmono=(x(:,1)+x(:,2))/2;
354     for j=1:floor(length(Xmono)/Fs)-5
355         start=j*Fs;
356         End=(j+5)*Fs;
357         m=Xmono(start:End);
358         Nirv=[Nirv,m];
359     end
360 end
361 %Load first two songs of Pearl Jam
362 PJ=[];
363     for z=1:2
364         [x Fs]=audioread([l,'\',list3(z).name]);
365         Xmono=(x(:,1)+x(:,2))/2;
366         for j=1:floor(length(Xmono)/Fs)-5
367             start=j*Fs;
368             End=(j+5)*Fs;
369             m=Xmono(start:End);
370             PJ=[PJ,m];
371         end
372     end
373 %Load first two songs of Soundgarden
374 SG=[];
375 for z=1:2
376     [x Fs]=audioread([d,list4(z).name]);
377     Xmono=(x(:,1)+x(:,2))/2;
378     for j=1:floor(length(Xmono)/Fs)-5
379         start=j*Fs;
380         End=(j+5)*Fs;
381         m=Xmono(start:End);
382         SG=[SG,m];
383     end
384 end

```

```

385 %% FFT Data
386 clc, clear song_fft song_Spec
387 %Set a number of 5 second snippets to test
388 samp=150;
389 g1=randperm(samp);
390 g2=randperm(samp);
391 g3=randperm(samp);
392 g4=randperm(samp);
393 X=[AIC(:,g1(1:samp)) Nirv(:,1:g2(1:samp)) PJ(:,1:g3(1:samp))
    SG(:,g4(1:samp))];
394 n = length(X);
395 t = (1:n)/Fs;
396 song_fft=[];
397 %FFT the data
398 for i = 1:4*samp
399     song_fft=[song_fft;real(fftshift(fft(X(i,:))))];
400 end
401 %% Compute SVD
402 clear U S V KNN_AA KNN_OA NB_OA NB_AA DA_AA DA_OA
403 %FFT needs to be transposed
404 song_fft=song_fft';
405 [U S V]=svd(real(song_fft)/sqrt(4*samp-1),0);
406 figure(1)
407 plot(diag(S).^2,'bo','Linewidth',2)
408 xlim([1 40]);
409 grid on
410 xlabel('Index of Singular Value','FontSize',14)
411 ylabel('Magnitude of Singular Value','FontSize',14)
412 title('Music 2: Singular Values','FontSize',18)
413 print('MC2_Sing','-djpeg')
414 figure(2)
415 hold on
416 plot(V(1:samp,1),V(1:samp,2),'ko','Linewidth',2)
417 plot(V(samp+1:2*samp,1),V(samp+1:2*samp,2),'ro','Linewidth',
    ,2)
418 plot(V(2*samp+1:3*samp,1),V(2*samp+1:3*samp,2),'co','
    Linewidth',2)
419 plot(V(3*samp+1:4*samp,1),V(3*samp+1:4*samp,2),'go','
    Linewidth',2)
420 hold off
421 grid on
422 title('Projection onto 2 Dominant V Components','FontSize'
    ,18)
423 legend('Alice in Chains','Nirvana','Pearl Jam','Soundgarden')
424 print('MC2_Proj','-djpeg')
425 train=samp-20;

```

```

426 test=samp-train;
427 %% Train and Predict Model
428 clear q1 q2 q3 xClassic xMetal xGrunge xtrain xtest ctrain nb
    pre solution, close all
429 for z=1:50
430     %Create random permutations
431     q1=randperm(samp);
432     q2=randperm(samp);
433     q3=randperm(samp);
434     q4=randperm(samp);
435     %Create the actual solution (for comparing to prediction)
436     solution=[ones(test,1);2*ones(test,1);3*ones(test,1);4*
        ones(test,1)];
437
438     %Create the training data
439     xAIC=V(1:samp,1:26);
440     xNirv=V(samp+1:2*samp,1:26);
441     xPJ=V(2*samp+1:3*samp,1:26);
442     xSG=V(3*samp+1:4*samp,1:26);
443
444     %Put everything together into train and test
445     xtrain=[xAIC(q1(1:train),:); xNirv(q2(1:train),:);xPJ(q3
        (1:train),:);xSG(q4(1:train),:)]];
446     xtest=[xAIC(q1(train+1:end),:); xNirv(q2(train+1:end),:);
        xPJ(q3(train+1:end),:);xSG(q4(train+1:end),:)]];
447     ntest=size(xtest,1);
448     % Naive Bayes
449     ctrain=[ones(train,1); 2*ones(train,1);3*ones(train,1);4*
        ones(train,1)];
450     nb=fitcnb(xtrain,ctrain);
451     % N-B Prediction
452     pre=nb.predict(xtest);
453     bar(pre)
454     NB_acc=pre==solution;
455     NB_OA(z)=(sum(NB_acc)/ntest)*100;
456     % Discriminant Analysis
457     SVM=classify(xtest,xtrain,ctrain);
458     % DA Prediction
459     bar(SVM)
460     DA_acc=SVM==solution;
461     DA_OA(z)=(sum(DA_acc)/ntest)*100;
462     % K Nearest Neighbor
463     idx=fitcknn(xtrain,ctrain);
464     % KNN Predict
465     pre_knn=idx.predict(xtest);
466     bar(pre_knn)

```

```

467     KNN_acc=pre_knn==solution;
468     KNN_OA(z)=(sum(KNN_acc)/ntest)*100;
469 end
470 nave=size(NB_OA,2);
471 NB_AA=sum(NB_OA)/nave
472 DA_AA=sum(DA_OA)/nave
473 KNN_AA=sum(KNN_OA)/nave
474
475 %% Music Classification Part 3: Genre Classification
476 %% Load the Data
477 clear all;close all; clc
478 %Load Songs in directory
479 CR1='Classic Rock\Crosby, Stills & Nash\The Greatest Hits\05
    Marrakesh Express.mp3';
480 CR2='Classic Rock\Steve Miller Band\Wide River\06 Horse and
    Rider.wma';
481 M1='Metal\Antennas to Hell The Best of Slipknot\16 Sulfur.
    wma';
482 M2='Metal\Somewhere Back in Time The Best of 1980-1989\04
    The Trooper.wma';
483 G1='Seattle_Grunge\Pearl Jam\Rearviewmirror (Pearl Jam
    Greatest Hits 1991-2003) Disc 1\04 Jeremy.mp3';
484 G2='Seattle_Grunge\Alice in Chains\Alice In Chains2\28 Would
    (disc 2).mp3';
485
486 CR=[];
487 m=[];
488 %Load first 2 songs of Classic Rock
489 [x Fs]=audioread(CR1);
490 Xmono=(x(:,1)+x(:,2))/2;
491 for j=1:floor(length(Xmono)/Fs)-5
492     start=j*Fs;
493     End=(j+5)*Fs;
494     m=Xmono(start:End);
495     CR=[CR,m];
496 end
497 [x Fs]=audioread(CR2);
498 Xmono=(x(:,1)+x(:,2))/2;
499 for j=1:floor(length(Xmono)/Fs)-5
500     start=j*Fs;
501     End=(j+5)*Fs;
502     m=Xmono(start:End);
503     CR=[CR,m];
504 end
505 %Load first two songs of Metal
506 Metal=[];

```

```

507 [x Fs]=audioread(M1);
508 Xmono=(x(:,1)+x(:,2))/2;
509 for j=1:floor(length(Xmono)/Fs)-5
510     start=j*Fs;
511     End=(j+5)*Fs;
512     m=Xmono(start:End);
513     Metal=[Metal,m];
514 end
515 [x Fs]=audioread(M2);
516 Xmono=(x(:,1)+x(:,2))/2;
517 for j=1:floor(length(Xmono)/Fs)-5
518     start=j*Fs;
519     End=(j+5)*Fs;
520     m=Xmono(start:End);
521     Metal=[Metal,m];
522 end
523 %Load first two songs of Grunge
524 Grunge=[];
525 [x Fs]=audioread(G1);
526 Xmono=(x(:,1)+x(:,2))/2;
527 for j=1:floor(length(Xmono)/Fs)-5
528     start=j*Fs;
529     End=(j+5)*Fs;
530     m=Xmono(start:End);
531     Grunge=[Grunge,m];
532 end
533 [x Fs]=audioread(G2);
534 Xmono=(x(:,1)+x(:,2))/2;
535 for j=1:floor(length(Xmono)/Fs)-5
536     start=j*Fs;
537     End=(j+5)*Fs;
538     m=Xmono(start:End);
539     Grunge=[Grunge,m];
540 end
541 %% FFT Data
542 clc, clear song_fft song_Spec
543 %Set a number of 5 second snippets to test
544 samp=150;
545 g1=randperm(samp);
546 g2=randperm(samp);
547 g3=randperm(samp);
548 X=[CR(:,g1(1:samp)) Metal(:,1:g2(1:samp)) Grunge(:,1:g3(1:
    samp))];
549 n = length(X);
550 t = (1:n)/Fs;
551 song_fft=[];

```



```

552 %FFT the data
553 for i = 1:3*samp
554     song_fft=[song_fft;real(fftshift(fft(X(i,:))))];
555 end
556 %% Compute SVD
557 clear U S V KNN_AA KNN_OA NB_OA NB_AA DA_AA DA_OA
558 %FFT needs to be transposed
559 song_fft=song_fft';
560 [U S V]=svd(real(song_fft)/sqrt(3*samp-1),0);
561 figure(1)
562 plot(diag(S).^2,'bo','Linewidth',2)
563 xlim([1 40]);
564 grid on
565 xlabel('Index of Singular Value','FontSize',14)
566 ylabel('Magnitude of Singular Value','FontSize',14)
567 title('Music 3: Singular Values','FontSize',18)
568 print('MC3_Sing','-djpeg')
569 figure(2)
570 hold on
571 plot(V(1:samp,1),V(1:samp,2),'ko','Linewidth',2)
572 plot(V(samp+1:2*samp,1),V(samp+1:2*samp,2),'ro','Linewidth',
573     ,2)
574 plot(V(2*samp+1:3*samp,1),V(2*samp+1:3*samp,2),'co','
575     Linewidth',2)
576 hold off
577 grid on
578 title('Projection onto 2 Dominant V Components','FontSize'
579     ,18)
580 legend('Classic Rock','Metal','Grunge')
581 print('MC3_Proj','-djpeg')
582 train=samp-20;
583 test=samp-train;
584 %% Train and Predict Model
585 clear q1 q2 q3 xClassic xMetal xGrunge xtrain xtest ctrain nb
586 pre solution, close all
587 for z=1:50
588     %Create random permutations
589     q1=randperm(samp);
590     q2=randperm(samp);
591     q3=randperm(samp);
592     %Create the actual solution (for comparing to prediction)
593     solution=[ones(test,1);2*ones(test,1);3*ones(test,1)];
594
595     %Create the training data
596     xClassic=V(1:samp,1:26);
597     xMetal=V(samp+1:2*samp,1:26);

```

```

594     xGrunge=V(2*samp+1:3*samp,1:26);
595
596     %Put everything together into train and test
597     xtrain=[xClassic(q1(1:train),:); xMetal(q2(1:train),:);
598           xGrunge(q3(1:train),:)];
599     xtest=[xClassic(q1(train+1:end),:); xMetal(q2(train+1:end
600           ),:);xGrunge(q3(train+1:end),:)];
601     ntest=size(xtest,1);
602     % Naive Bayes
603     ctrain=[ones(train,1); 2*ones(train,1);3*ones(train,1)];
604     nb=fitcnb(xtrain,ctrain);
605     % N-B Prediction
606     pre=nb.predict(xtest);
607     bar(pre)
608     NB_acc=pre==solution;
609     NB_OA(z)=(sum(NB_acc)/ntest)*100;
610     % Discriminant Analysis
611     SVM=classify(xtest,xtrain,ctrain);
612     % DA Prediction
613     bar(SVM)
614     DA_acc=SVM==solution;
615     DA_OA(z)=(sum(DA_acc)/ntest)*100;
616     % K Nearest Neighbor
617     idx=fitcknn(xtrain,ctrain);
618     % KNN Predict
619     pre_knn=idx.predict(xtest);
620     bar(pre_knn)
621     KNN_acc=pre_knn==solution;
622     KNN_OA(z)=(sum(KNN_acc)/ntest)*100;
623 end
624 nave=size(NB_OA,2);
625 NB_AA=sum(NB_OA)/nave
626 DA_AA=sum(DA_OA)/nave
627 KNN_AA=sum(KNN_OA)/nave

```