AMATH 582 Homework 4: ‘Eigenfaces’ & Music Identification

Sara L. Daley

3/6/2020

Abstract

The goal of this paper is to perform Spectral Value Decomposition analysis on two different types of data. The first type being facial images (either cropped or uncropped), and the second being music. After performing analysis on the music, we use Matlab’s Classification Learner app to classify the music for three different tests.

1 Introduction and Overview

This homework set contained two sections. The first section was to take two face databases (cropped and uncropped) from Yale, and perform Spectral Value Decomposition (SVD) analysis. We were asked to decompose the signal, and find the number of modes necessary for good facial image reconstructions. We then were to compare these results between the cropped and uncropped facial images.

The second section of this homework was to classify music of our choosing. We were asked to consider three tests. Test 1 was to classify between three different bands, from different genres. Test 2 was to classify three different bands from the same genre. Test 3 was a binary classification of genre. For Test 1 we chose AC/DC, Chuck Berry, and Taylor Swift. For Test 2 we chose AC/DC, Aerosmith, and Tom Petty. For Test 3 we chose to create a Classic Rock classifier.

2 Theoretical Background

In the previous homework set, *Principal Component Analysis*, we went through the SVD more in depth so we only provide a quick refresher. Singular value decomposition (SVD) is a factorization of a matrix into a number of constitutive components all of which have a specific meaning in applications[1]. The full SVD decomposition takes the form[1]

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^*. \tag{1}$$

This is saying that we can decompose a matrix that we don't know anything about ($\mathbf{A}$ here) and find out information from its displacement ($\mathbf{U}$), singular values ($\boldsymbol{\Sigma}$), and direction ($\mathbf{V}$). Note that if `rank(A)` is r, then there are r nonzero singular values ($\boldsymbol{\Sigma}$).

In essence, the decomposition of a matrix reduces its dimensions and this is one area of what machine learning uses to accomplish it's purpose. There are two forms of machine learning, supervised and unsupervised. We focus on supervised learning, as we provide labels to the model. There are two to three phases of learning:

1. Test
2. Train
3. Validation

The first two phases are vital, and the validation phase is required if the model will export into a production environment; it is also recommended for large data sets. When working with supervised learning models, we considered

- K Nearest Neighbor (KNN): algorithm assumes similar labels exist close (or within K neighbor close)[4]
- Support Vector Machines (SVM): look for which data points are on/nearest the boundary line
- Classification and Regression Trees (CART): series of features used to classify
- Naive Bayes: uses Bayes Theorem to predict[4]

We also worked within Matlab's Classification Learner app. With this app we feed it a table of our data in one column and our labels in another column. We chose to let Matlab perform a K-fold cross validation with $K = 5$, and did not use a validation set. Cross validation is important because we want a fair representation of the randomly selected test/train split of data. We can think of train-test-validation to follow the patterns of school. We train during the quarter with homework, we test at specific intervals, and our validation is the final exam. When doing random splits of the data for training and test, we must do so numerous times to accurately represent the variance of data. For example, if you were in class and had homework that covered more World War II than any other topic, were tested on WWII and other topics, and your final exam was on nothing but WWII- you'd expect to do well. The steps with K-Fold Cross-validation are:

1. Shuffle the dataset randomly
2. Split the dataset into K groups
3. For each group,
   (a) Take $K - 1$ groups as the training set (Example: groups 1-4)
   (b) Take 1 group as the test set (Example: group 5)
4. Run model training on the training set, test on the test set
5. Train again with different K train/test (Example: train with groups 2-5, test with group 1)
6. Final model score is an average of K-Fold scores

3 Algorithm Implementation and Development

Please refer to Algorithm 1 for 'Eigenfaces' and Algorithm 2 for Music Classification

Algorithm 1: SVD Analysis on Facial Images

```
import images into a cell
for $j = 1 : length(imagecell)$ do
  photos = imread
end for
for $j = 1 : length(photos)$ do
  reshape each image to be a column vector
end for
set a lambda value
AR = fastpcp
svd(AR,'Econ')
calculate $\mathbf{\Sigma}$ and rank
reconstruct by $\mathbf{U} * \mathbf{S} * \mathbf{V'}$
```

Algorithm 2: Music Classification

```
import songs of artist into a cell
for $j = 1 : length(artistcell)$ do
  [artist, sample rate] = audioread
end for
while $k \leq Numofsongsfortheartist$ do
  pick a time start and time end
  while time end $\leq$ length of song - 5 do
    artist sectioned song = artist(timestart:timeend)
    enumerate needed indexes
  end while
  enumerate needed indexes
end while
set a lambda value
AR = inexact_alm_rpca
svd(AR,'Econ')
calculate rank(A) so know how many features to feed to model
Pull out v(:,1:rank) features
randperm the length of artist row 1
xtrain = vfeatures(randperm,:)
Create tables to pass to Classification Learner
```

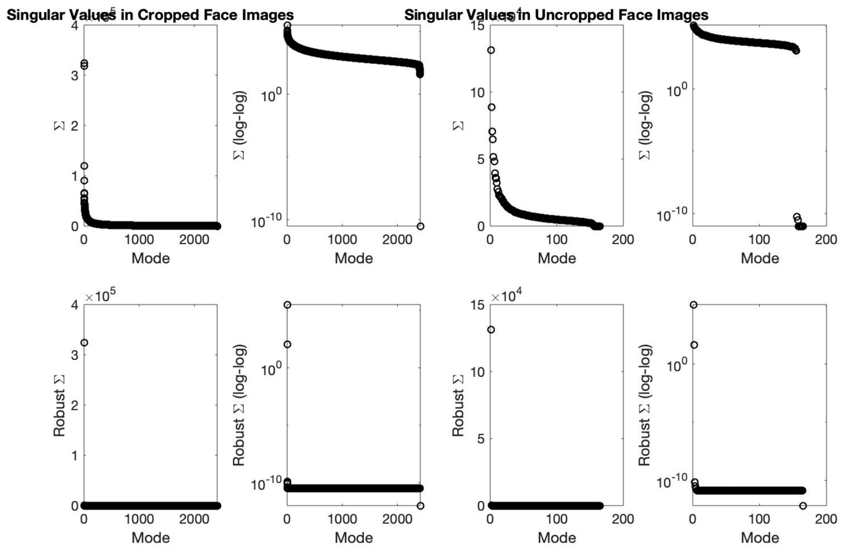


Figure 1: Singular Values for both the Cropped and Uncropped facial images. Top row represents singular values from Economy SVD, while bottom row represents singular values using fastpcp filtering.

4 Computational Results

4.1 Eigenfaces

For the Yale faces section, we had two face databases, cropped and uncropped. In running the SVD, we took two different SVD methods. The first was running the Economy SVD, the other was running with the algorithm fastpcp[3]. We chose to run the fastpcp (fast principal component pursuit) algorithm rather than Inexact ALM as it runs an order of magnitude faster. To reconstruct the faces, we multiplied the components together as in Equation 1. In this case, **U** represents an m x n matrix consisting of n faces, m image pixels (the eigenface basis). **Σ** represents singular values (information about the rank of the data set), and **V** gives us the coefficients needed to reconstruct each face using the eigenface basis. The singular values for the image reconstructions vary between the cropped and uncropped data sets.

Figure 1 shows that between the Economy SVD and the fastpcp algorithm, the number of modes needed to contain most of the variance differs

	Economy Rank	`fastpcp` Rank
Cropped (2414 modes)	2413	2
Uncropped (165 modes)	155	2

Table 1: Rank of both Cropped and Uncropped images

drastically. This implies that it will take many more modes to reconstruct the uncropped faces than it will the cropped faces. We look at rank with Table 4.1, we see that the difference between rank and modes is not substantial for the raw images and Economy SVD, but is with the fastpcp algorithm.

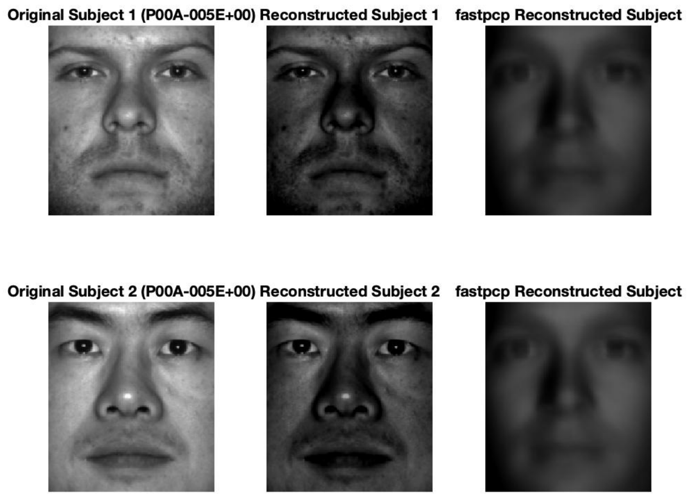

Original Subject 1 (P00A-005E+00)
Reconstructed Subject 1
fastpcp Reconstructed Subject
Original Subject 2 (P00A-005E+00)
Reconstructed Subject 2
fastpcp Reconstructed Subject

Figure 2: SVD Analysis of the cropped Yale faces. The first column represents the original subjects. The second column represents the reconstructed faces using Economy SVD. The third column represents the reconstructed faces using fastpcp filtering.

When you see that there is rank 2 for fastpcp, we begin to wonder if this was indeed the best algorithm for this problem. Nevertheless, Figure 2 shows the analysis for the first two subjects in the cropped faces data set. While the Economy SVD shows a good reconstruction of the face, the fastpcp only shows an ‘average’ face. This is because there are only two eigenfaces! When

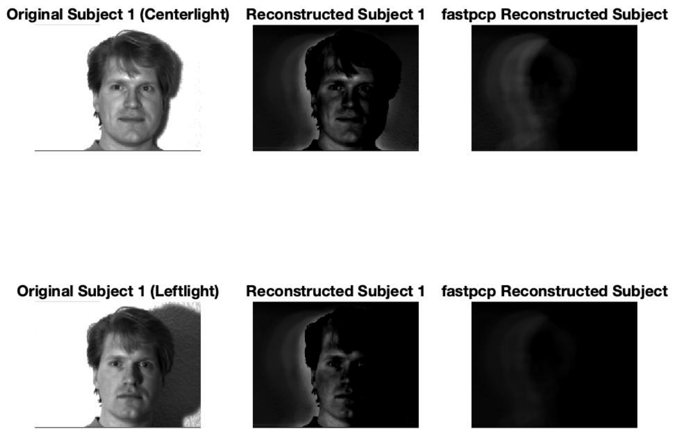

Original Subject 1 (Centerlight)
Reconstructed Subject 1
fastpcp Reconstructed Subject
Original Subject 1 (Leftlight)
Reconstructed Subject 1
fastpcp Reconstructed Subject

Figure 3: SVD Analysis of the Yale faces. The first column represents the original subject at two different lights. The second column represents the reconstructed face using Economy SVD. The third column represents the reconstructed face using fastpcp filtering.

we have a rank of 2, we have a value for our **A** matrix (Equation 1) that contains two columns (recall that **U** represents a matrix of n faces). Figure 3 shows that even the Economy SVD had trouble reconstructing the face when there are shadows and a background to filter out.

4.2 Music

There were three classification tests covered. The bands/artists considered throughout the tests were AC/DC, Aerosmith, Chuck Berry, Taylor Swift, and Tom Petty. Figure 4 shows a five second clip of a song considered in the data set. For each artist, there was anywhere from five to eight full songs sectioned apart in five second intervals to make the data set. To make the train/test set, we shuffled the data and placed all of the artists considered in a table. We then fed that table to Matlab's Classification Learner app. This app has the ability to do K-Fold cross-validation, so we did not worry about splitting up our data into 80% train and 20% test sets. We should

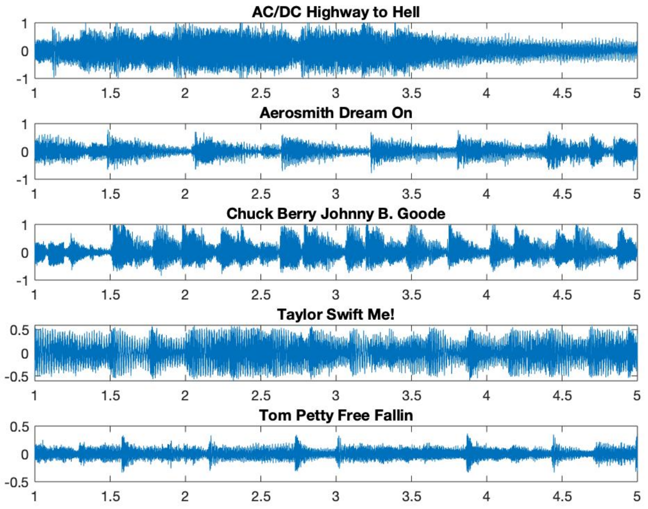


Figure 4: Five second clips of each of the Artists considered for Tests 1-3.

note that we did not create a validation set, so the accuracy of the models will be inflated. To find out how many features to give to the model to use, we found the lowest rank of each artists in the test case.

In Test 1 we were asked to classify between three different artists, so we considered AC/DC, Chuck Berry, and Taylor Swift. To see how the model did, we will look at the confusion matrix. The confusion matrix rows show the true class, columns show the predicted class. The diagonal shows where the true and predicted classes match. The bars to the right are summaries of how well the model did per class. Figure 5 shows that we misclassified 3% (11 times) of AC/DC as Chuck Berry, and 1% (3 times) of AC/DC as Taylor Swift. Overall we have an accuracy score of 97% for AC/DC, 100% for Chuck Berry, and 100% for Taylor Swift. The model that performed the best for this test was 'Gaussian Naive Bayes' which uses a Gaussian distribution for the predictors.

In Test 2 we were asked to classify three different artists, but of all the same genre. Thus, we considered AC/DC, Aerosmith, and Tom Petty. As it's a harder problem, we are not surprised to see the scores decrease as seen in Figure 6. We misclassified AC/DC as Aerosmith (4%/19 times) and Tom Petty (1%/3 times). We also misclassified Tom Petty as Aerosmith

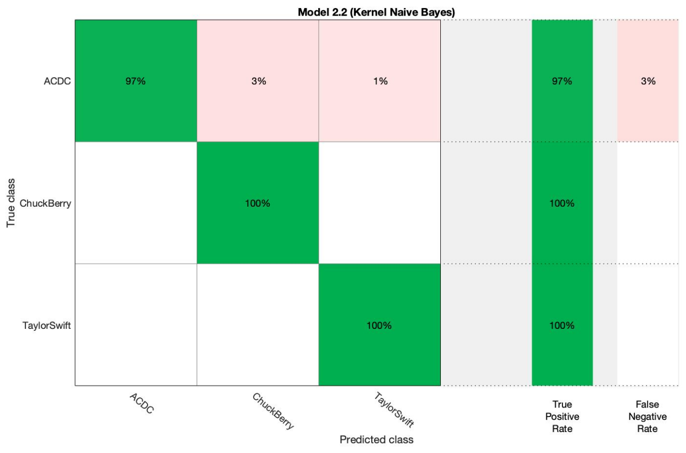


Figure 5: Confusion matrix of Test 1: Classifying three distinctly different artists

(1%/2 times). Overall we have an accuracy score of 95% for AC/DC, 100% for Aerosmith, and 99% for Tom Petty. The model that performed the best for this test was 'Fine Gaussian SVM'. This SVM makes finely detailed distinctions between the classes.

In Test 3 we were asked to make a genre classifier. We decided to make a classic rock classifier. The data set contained classic rock: AC/DC, Aerosmith, Tom Petty and not classic rock: Chuck Berry, and Taylor Swift. Figure 7 shows that we misclassified classic rock as not being so 3% (27 times) of the time. We did not misclassify music that isn't classic rock as being classic rock. Overall we have an accuracy score of 97% for classic rock, and 100% for not classic rock. The model that performed best for this test was 'Fine KNN'. It is KNN, with only considering one nearest neighbor.

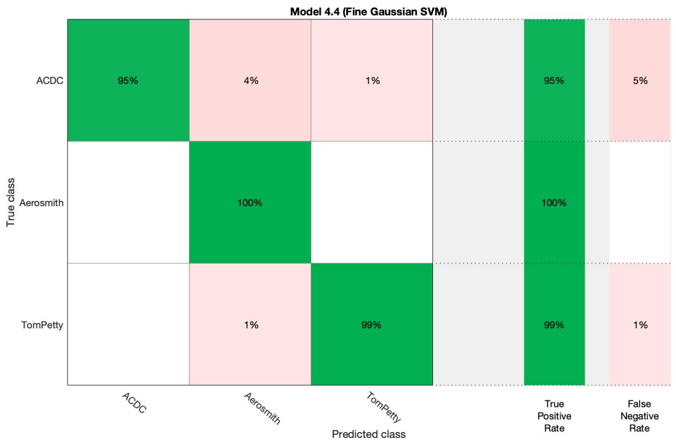


Figure 6: Confusion matrix of Test 2: Classifying three artists of the same genre.

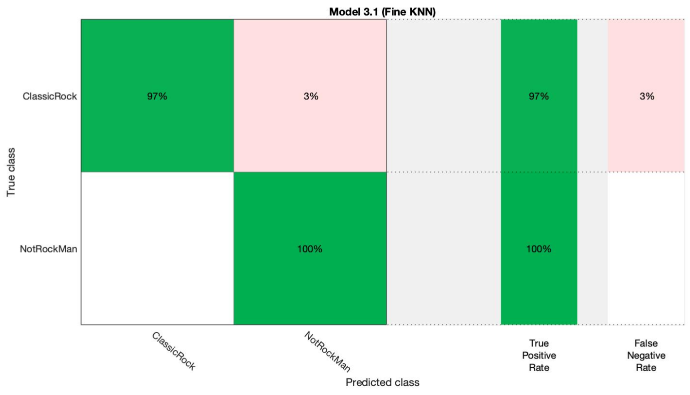


Figure 7: Confusion matrix of Test 3: Classic Rock classifier.

5 Summary and Conclusions

5.1 Eigenfaces

Although fast principal component pursuit may be helpful in certain situations, it was not a good choice in facial reconstruction. We need more modes to contain the unique features that make up a face. The author should've known better; it was advertised to be a "low rank approximation" and as such it is good, but not for the scope of this problem. It should be noted that the fastpcp algorithm was chosen because the inexact_alm_rpca ran quite a while on this data set without a return. Thus, we chose this method due to its computational performance[3]. In doing so we got caught up in convenience. It was as if we bought a sports car for it's beauty and agility, but the first time we went to Costco we realized it was not practical for someone shopping for a family. More analysis should be completed to make an informed decision on the best method to use to filter; whether it's inexact_alm_rpca or another algorithm.

5.2 Music

The accuracy scores for each of the tests for music classification appear to be very good, but we have to take these scores with a grain of salt. There is one large red flag created when making up this data set: we sectioned off five second clips of songs, instead of taking one five second clip from one song. Music contains chorus', amongst other times of repetition, so it is highly likely that the machine was being tested on something *extremely* close to what it was trained on. We also did not create a validation set, which would also teach the model some humility much like a final exam does to a student. The second item that needs addressing is the size of each of the artist data sets. The author likes AC/DC, so she put more of their songs into the data than any other so that artist was over represented. We also note that on average Chuck Berry's songs were shorter in length which led to his music being slightly under represented. The last item to address would be why AC/DC, even with more data, was misclassified most often. This is because when we created the table of features, each artist array needed to be the same length. With AC/DC having a rank of 400, and Chuck Berry only having a rank of 32, we could not pull in enough features to assist in full classification. One thing that should be explored is changing the filter width (lambda) for AC/DC when creating the SVD to see if we could capture more variance with less modes.

References

[1] J. Nathan Kutz. *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*, Oxford University Press, 2013.

[2] `https://www.mathworks.com/help/matlab/index.html`

[3] `https://github.com/andrewssobral/lrslibrary` `https://www.mathworks.com/matlabcentral/fileexchange/48404-lrslibrary`

[4] `https://towardsdatascience.com`

Appendices

Please refer to my Github repo for all code and related documents.

A MATLAB Functions

- `randperm(N)`

  this returns a vector containing a random permutation of 1:N [2]

- `table`

  places heterogeneous data into one container. It has a ton of features so refer to Matlab help [2]

- APP: Classification Learner

  Extremely easy to use, and very fun. This introductory video was very helpful: Classify Data Using the Classification Learner App

B MATLAB Codes

Included here is my code for this homework set, though I do call the `fastpcp` & `inexact_alm_rpca` functions available through the LRS Library[3].

Table of Contents

```
close all; clc; clear
```

HW4, Part1: Load Yale Faces, start with the cropped images

```
% grab all the directories of the images
im_crop = dir('yalefaces_cropped/yaleB*/*.pgm');

% put each directory of images into one big cell of images
for k = 1:length(im_crop)
    imname = im_crop(k).name;
    imfolder = im_crop(k).folder;
    readthis = strcat(imfolder,'/',imname);
    yale_crop{k} = imread(readthis);
end

% flatten each image cell to be a vector
for j = 1:length(yale_crop)
    yale_crop_reshape(:,j) = yale_crop{j}(:);
end

%change from image space to double
yale_crop_reshape = double(yale_crop_reshape);
```

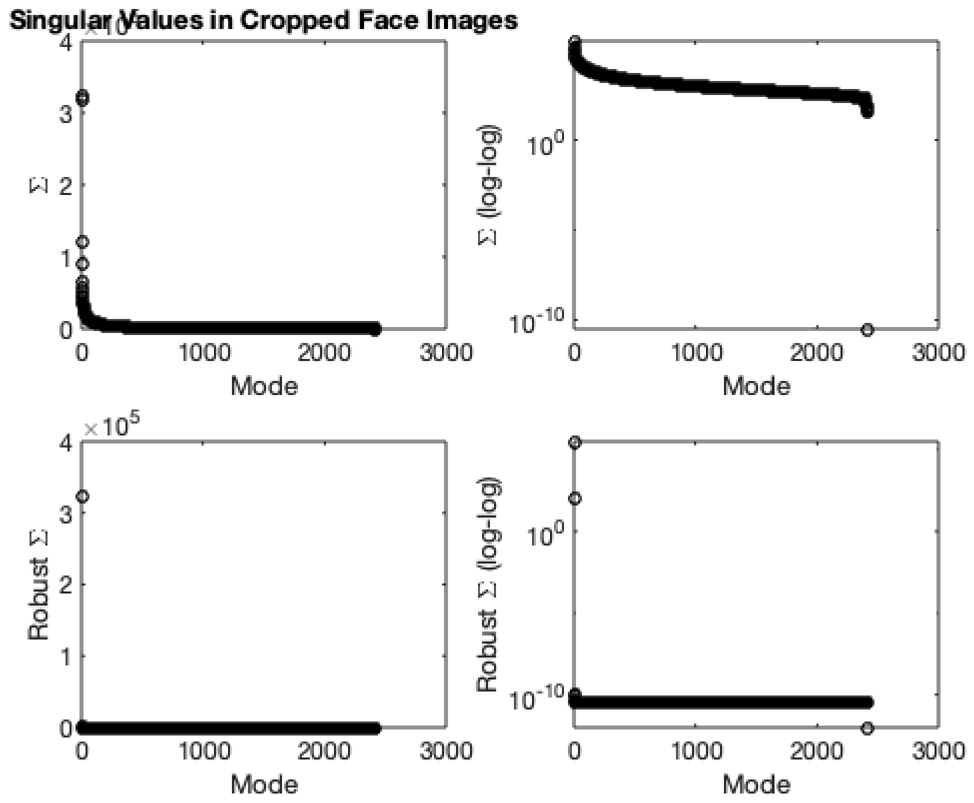
Perform SVD on cropped images

```
[Cm,Cn] = size(yale_crop_reshape);
Cmn = mean(yale_crop_reshape,2);
yale_crop_reshape = yale_crop_reshape - repmat(Cmn,1,Cn);

lambda = 0.02;
[C_AR,Shrink] = fastpcp(yale_crop_reshape.',lambda);
%[Cropped1R,Cropped12R] =
 inexact_alm_rpca(yale_crop_reshape.',lambda);
[C_U, C_S, C_V] = svd(yale_crop_reshape, 'Econ');
C_sigma = diag(C_S);
[C_Ur, C_Sr, C_Vr] = svd(C_AR.');
C_sigmaR = diag(C_Sr);
```

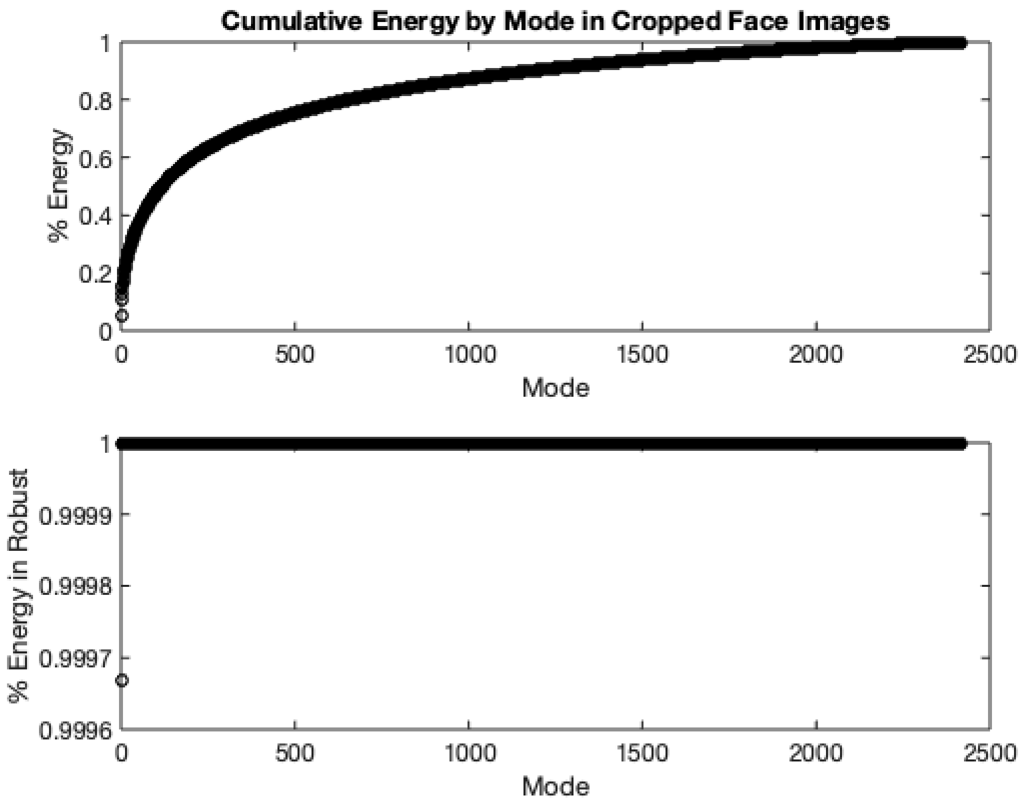
```
for k = 1:length(C_sigma)
    C_energy(k) = sum(C_sigma(1:k))/sum(C_sigma);
end

for k = 1:length(C_sigmaR)
    C_energyR(k) = sum(C_sigmaR(1:k))/sum(C_sigmaR);
end

Crank_s = rank(C_S);
Crank_sr = rank(C_Sr);
A_Crecon = C_U(:,1:Crank_s) * C_S(1:Crank_s,1:Crank_s) *
 C_V(1:Crank_s,1:Crank_s)';
A_CreconR = C_Ur(:,1:Crank_sr) * C_Sr(1:Crank_sr,1:Crank_sr) *
 C_Vr(1:Crank_sr,1:Crank_sr)';
```

Plots for Cropped Images

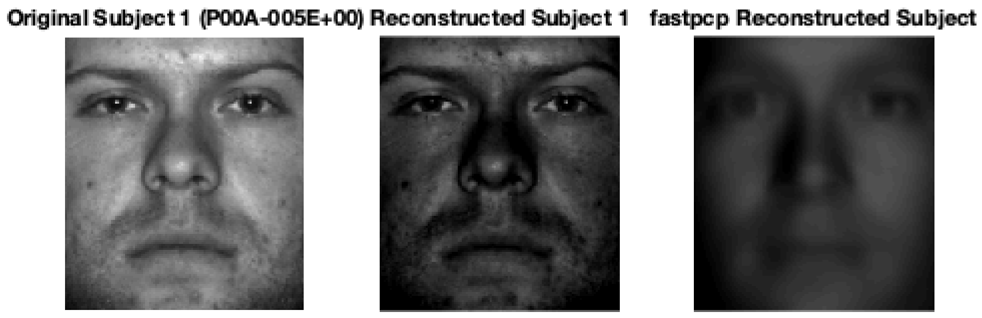
```
figure(1)
subplot(2,2,1), plot(C_sigma, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize',[12]); xlabel('Mode')
ylabel('\Sigma'); title('Singular Values in Cropped Face Images')

subplot(2,2,2), semilogy(C_sigma,'ko','Linewidth',[1.2])
set(gca,'Fontsize',[12])
ylabel('\Sigma (log-log)'); xlabel('Mode')

subplot(2,2,3), plot(C_sigmaR, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize',[12])
ylabel('Robust \Sigma'); xlabel('Mode')

subplot(2,2,4), semilogy(C_sigmaR,'ko','Linewidth',[1.2])
set(gca,'Fontsize',[12])
ylabel('Robust \Sigma (log-log)'); xlabel('Mode')

figure(2)
subplot(2,1,1), plot(C_energy, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize', [12]) % 2089(/2414) modes to 0.9850
ylabel('% Energy'); xlabel('Mode');
title('Cumulative Energy by Mode in Cropped Face Images')
subplot(2,1,2), plot(C_energyR, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize', [12]) % 1 mode to 0.9967
ylabel('% Energy in Robust'); xlabel('Mode')

% figure(3)
% j = 1;
% while j<=514
%     for h = 1:9
%         subplot(3,3,h)
%         test1 = reshape(A_Crecon(:,j),192,168);
%         imshow(uint8(test1));
%         title({'Reconstructed Subject ',h,'(P00A-005E-10)'})
%         j = j + 64;
%     end
% end

% figure(4)
% j = 1;
% while j<=514
%     for h = 1:9
%         subplot(3,3,h)
%         test1 = reshape(A_CreconR(:,j),192,168);
%         imshow(uint8(test1));
%         title({'Reconstructed Filtered Subject
',h,'(P00A-005E-10)'})
%         j = j + 64;
%     end
% end

figure(99)
subplot(2,3,1)
imshow(yale_crop{1});
title('Original Subject 1 (P00A-005E+00)')
subplot(2,3,2)
imshow(uint8(reshape(A_Crecon(:,1),192,168)));
title('Reconstructed Subject 1')
subplot(2,3,4)
imshow(yale_crop{65});
title('Original Subject 2 (P00A-005E+00)')
subplot(2,3,5)
imshow(uint8(reshape(A_Crecon(:,65),192,168)));
title('Reconstructed Subject 2')
subplot(2,3,3)
imshow(uint8(reshape(A_CreconR(:,1),192,168)));
title('fastpcp Reconstructed Subject')
subplot(2,3,6)
imshow(uint8(reshape(A_CreconR(:,2),192,168)));
title('fastpcp Reconstructed Subject')
```

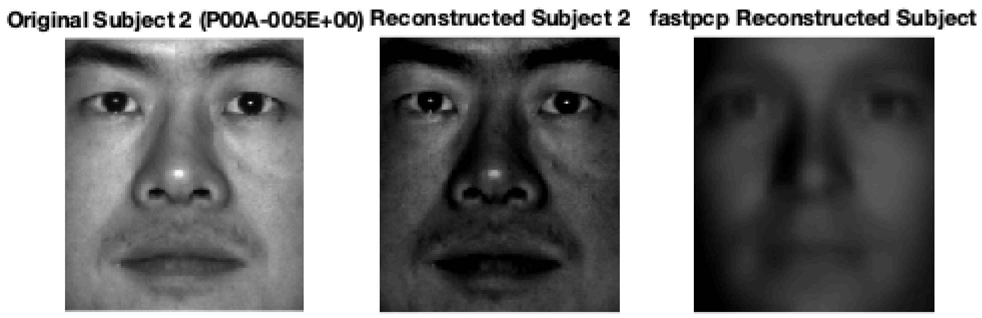


Singular Values in Cropped Face Images
$\times 10^5$
Σ
Mode
Σ (log-log)
Robust Σ
Robust Σ (log-log)



Cumulative Energy by Mode in Cropped Face Images
% Energy
Mode
% Energy in Robust


Original Subject 1 (P00A-005E+00)
Reconstructed Subject 1
fastpcp Reconstructed Subject


Original Subject 2 (P00A-005E+00)
Reconstructed Subject 2
fastpcp Reconstructed Subject

Load Yale Faces, now uncropped faces

```
% grab all the directories of the images
im_face = dir('yalefaces_uncropped/yalefaces/subject*');

% put each directory of images into one big cell of images
for k = 1:length(im_face)
    imname = im_face(k).name;
    imfolder = im_face(k).folder;
    readthis = strcat(imfolder,'/',imname);
    yale_face{k} = imread(readthis);
end

% flatten each image cell to be a vector
for j = 1:length(yale_face)
    yale_reshape(:,j) = yale_face{j}(:);
end

%change from image space to double
yale_reshape = double(yale_reshape);
```

Perform SVD on uncropped images

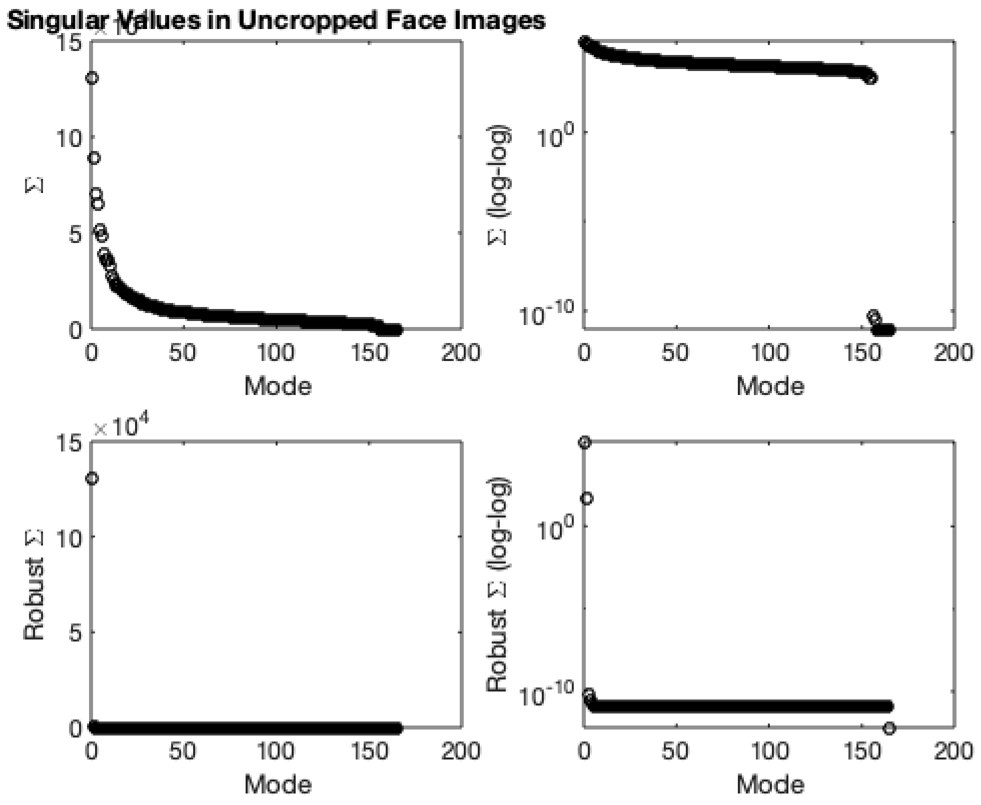
```
[Fm,Fn] = size(yale_reshape);
Fmn = mean(yale_reshape,2);
```

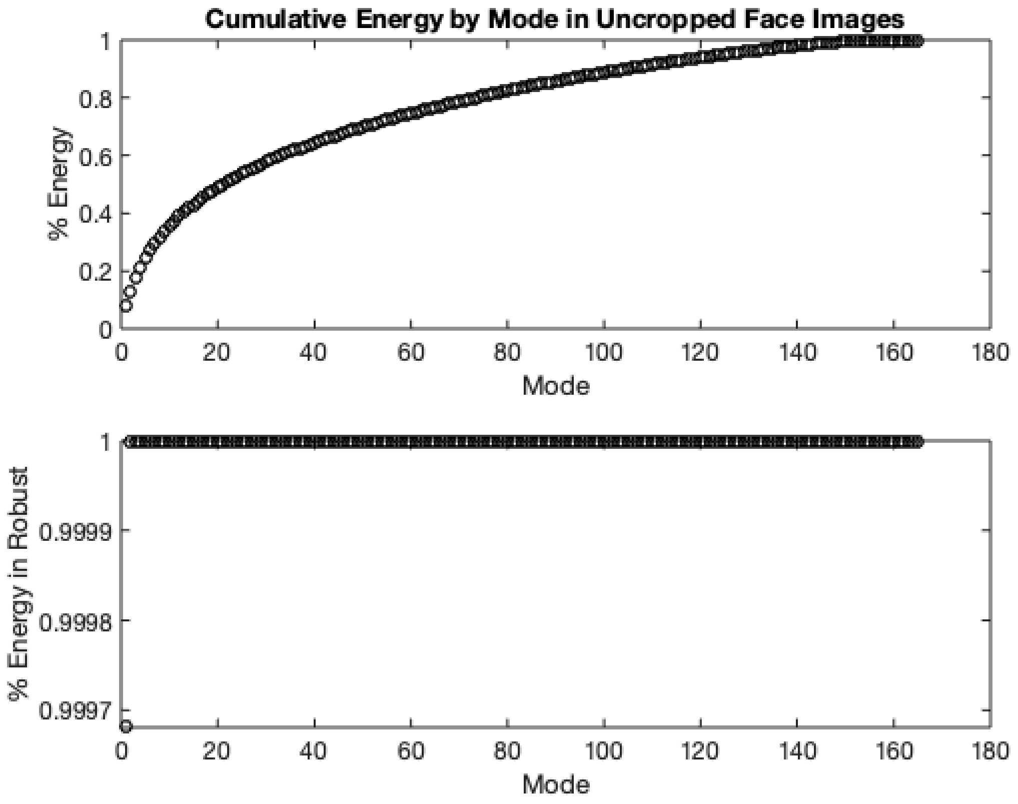
```
yale_reshape = yale_reshape - repmat(Fmn,1,Fn);

lambda = 0.02;
[F_AR,Shrink] = fastpcp(yale_reshape.',lambda);
%[Cropped1R,Cropped12R] = inexact_alm_rpca(yale_reshape.',lambda);
[F_U, F_S, F_V] = svd(yale_reshape, 'Econ');
F_sigma = diag(F_S);
[F_Ur, F_Sr, F_Vr] = svd(F_AR.','Econ'); %had to Econ- array size too
 big
F_sigmaR = diag(F_Sr);


for k = 1:length(F_sigma)
    F_energy(k) = sum(F_sigma(1:k))/sum(F_sigma);
end

for k = 1:length(F_sigmaR)
    F_energyR(k) = sum(F_sigmaR(1:k))/sum(F_sigmaR);
end

Frank_s = rank(F_S);
Frank_sr = rank(F_Sr);
A_Frecon = F_U(:,1:Frank_s) * F_S(1:Frank_s,1:Frank_s) *
 F_V(1:Frank_s,1:Frank_s)';
A_FreconR = F_Ur(:,1:Frank_sr) * F_Sr(1:Frank_sr,1:Frank_sr) *
 F_Vr(1:Frank_sr,1:Frank_sr)';
```

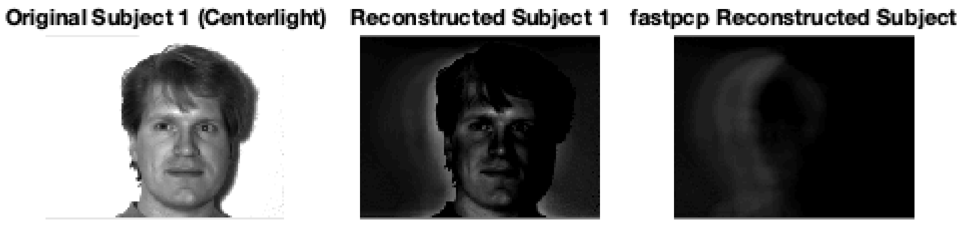
Plots for Uncropped Images

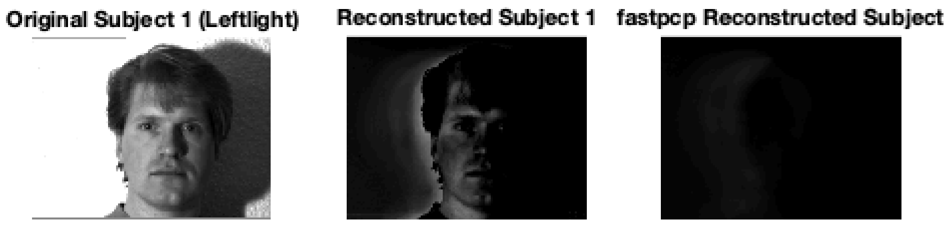
```
figure(5)
subplot(2,2,1), plot(F_sigma, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize',[12]); xlabel('Mode')
ylabel('\Sigma'); title('Singular Values in Uncropped Face Images')

subplot(2,2,2), semilogy(F_sigma,'ko','Linewidth',[1.2])
set(gca,'Fontsize',[12])
ylabel('\Sigma (log-log)'); xlabel('Mode')

subplot(2,2,3), plot(F_sigmaR, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize',[12])
ylabel('Robust \Sigma'); xlabel('Mode')

subplot(2,2,4), semilogy(F_sigmaR,'ko','Linewidth',[1.2])
set(gca,'Fontsize',[12])
ylabel('Robust \Sigma (log-log)'); xlabel('Mode')

figure(6)
subplot(2,1,1), plot(F_energy, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize', [12]) % 143(/165) modes to 0.9851
ylabel('% Energy'); xlabel('Mode');
title('Cumulative Energy by Mode in Uncropped Face Images')
subplot(2,1,2), plot(F_energyR, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize', [12]) % 1 mode to 0.9968
```

```
ylabel('% Energy in Robust'); xlabel('Mode')

% figure(7)
% j = 1;
% while j<=96
%     for h = 1:9
%         subplot(3,3,h)
%         test1 = reshape(A_Frecon(:,j),243,320);
%         imshow(uint8(test1));
%         title({'Reconstructed Subject (Centerlight) ',h})
%         j = j + 11;
%     end
% end
%
% figure(8)
% j = 1;
% while j<=96
%     for h = 1:9
%         subplot(3,3,h)
%         test1 = reshape(A_FreconR(:,j),243,320);
%         imshow(uint8(test1));
%         title({'Reconstructed Filtered Subject (Centerlight) ',h})
%         j = j + 11;
%     end
% end

figure(9)
subplot(2,3,1)
imshow(yale_face{1});
title('Original Subject 1 (Centerlight)')
subplot(2,3,2)
imshow(uint8(reshape(A_Frecon(:,1),243,320)));
title('Reconstructed Subject 1')
subplot(2,3,4)
imshow(yale_face{4});
title('Original Subject 1 (Leftlight)')
subplot(2,3,5)
imshow(uint8(reshape(A_Frecon(:,4),243,320)));
title('Reconstructed Subject 1')
subplot(2,3,3)
imshow(uint8(reshape(A_FreconR(:,1),243,320)));
title('fastpcp Reconstructed Subject')
subplot(2,3,6)
imshow(uint8(reshape(A_FreconR(:,2),243,320)));
title('fastpcp Reconstructed Subject')
```


Singular Values in Uncropped Face Images
Σ
Σ (log-log)
Robust Σ
Robust Σ (log-log)
Mode
×10⁴

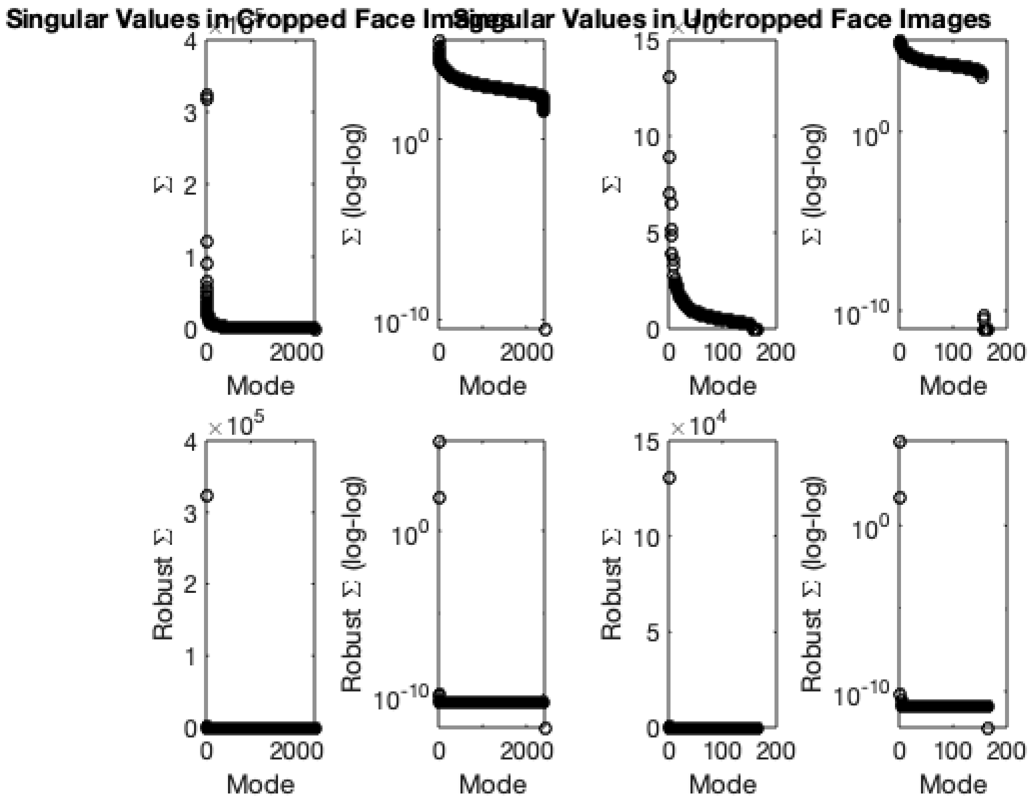

Cumulative Energy by Mode in Uncropped Face Images
% Energy
% Energy in Robust
Mode


Original Subject 1 (Centerlight)
Reconstructed Subject 1
fastpcp Reconstructed Subject



```
figure(100)

subplot(2,4,1), plot(C_sigma, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize',[12]); xlabel('Mode')
ylabel('\Sigma'); title('Singular Values in Cropped Face Images')

subplot(2,4,2), semilogy(C_sigma,'ko','Linewidth',[1.2])
set(gca,'Fontsize',[12])
ylabel('\Sigma (log-log)'); xlabel('Mode')

subplot(2,4,5), plot(C_sigmaR, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize',[12])
ylabel('Robust \Sigma'); xlabel('Mode')

subplot(2,4,6), semilogy(C_sigmaR,'ko','Linewidth',[1.2])
set(gca,'Fontsize',[12])
ylabel('Robust \Sigma (log-log)'); xlabel('Mode')

subplot(2,4,3), plot(F_sigma, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize',[12]); xlabel('Mode')
ylabel('\Sigma'); title('Singular Values in Uncropped Face Images')

subplot(2,4,4), semilogy(F_sigma,'ko','Linewidth',[1.2])
set(gca,'Fontsize',[12])
ylabel('\Sigma (log-log)'); xlabel('Mode')
```

```
subplot(2,4,7), plot(F_sigmaR, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize',[12])
ylabel('Robust \Sigma'); xlabel('Mode')

subplot(2,4,8), semilogy(F_sigmaR,'ko','Linewidth',[1.2])
set(gca,'Fontsize',[12])
ylabel('Robust \Sigma (log-log)'); xlabel('Mode')
```



Published with MATLAB® R2019b

Table of Contents

HW4, Part 2: Sounds of Music

```
close all; clc; clear;
```

Read in ACDC

grab all the songs

```
acdc = dir('Music_HW/ACDC/*.mp3');
% separate songs
for k = 1:length(acdc)
    songname = acdc(k).name;
    songfolder = acdc(k).folder;
    readthis = strcat(songfolder,'/',songname);
    [acdc_full{k}, acdc_Fs{k}] = audioread(readthis);
end
% sound(acdc_full{k}, Fs{k}); if you want to listen to the song

% section each song into 5 seconds
index = 1;
k = 1;
while k <= length(acdc_full)
    timestart = 5;
    timeend = 10;
    while timeend*acdc_Fs{k} < length(acdc_full{k})-5
        acdc_sect(index,:) = ...
            acdc_full{k}(timestart*acdc_Fs{k}:timeend*acdc_Fs{k});
        timestart = timeend;
        timeend = timeend + 5;
        index = index + 1;
    end
    k = k + 1;
end
acdc_sect = acdc_sect';
```

Read in Chuck Berry

grab all the songs

```
chb = dir('Music_HW/ChuckBerry/*.mp3');
% separate songs
for k = 1:length(chb)
    songname = chb(k).name;
    songfolder = chb(k).folder;
    readthis = strcat(songfolder,'/',songname);
    [chb_full{k}, chb_Fs{k}] = audioread(readthis);
end
% sound(chb_full{k}, Fs{k}); if you want to listen to the song

% section each song into 5 seconds
index = 1;
k = [1 4 5]; % we have different sampling rates
for kInd = 1:length(k)
    timestart = 2;
    timeend = 7;
    while timeend*chb_Fs{k(kInd)} < length(chb_full{k(kInd)})
        chb_sect(index,:) = ...
            chb_full{k(kInd)}
(timestart*chb_Fs{k(kInd)}:timeend*chb_Fs{k(kInd)});
        timestart = timeend;
        timeend = timeend + 5;
        index = index + 1;
    end

end
chb_sect = chb_sect';

index = 1;
k = [2 3];
for kInd = 1:length(k)
    timestart = 1;
    timeend = 6;
    while timeend*chb_Fs{k(kInd)} < length(chb_full{k(kInd)})
        chb_sect2(index,:) = ...
            chb_full{k(kInd)}
(timestart*chb_Fs{k(kInd)}:timeend*chb_Fs{k(kInd)});
        timestart = timeend;
        timeend = timeend + 5;
        index = index + 1;
    end

end
chb_sect2 = chb_sect2';
```

Read in Taylor Swift

grab all the songs

```
tay = dir('Music_HW/TSwift/*.mp3');
% separate songs
for k = 1:length(tay)
    songname = tay(k).name;
    songfolder = tay(k).folder;
    readthis = strcat(songfolder,'/',songname);
    [tay_full{k}, tay_Fs{k}] = audioread(readthis);
end
% sound(tay_full{k}, Fs{k}); if you want to listen to the song

% section each song into 5 seconds
index = 1;
k = 1;
while k <= length(tay_full)
    timestart = 5;
    timeend = 10;
    while timeend*tay_Fs{k} < length(tay_full{k})-5
        tay_sect(index,:) = ...
            tay_full{k}(timestart*tay_Fs{k}:timeend*tay_Fs{k});
        timestart = timeend;
        timeend = timeend + 5;
        index = index + 1;
    end
    k = k + 1;
end
tay_sect = tay_sect';
```

Read in Aerosmith

grab all the songs

```
aero = dir('Music_HW/Aerosmith/*.mp3');
% separate songs
for k = 1:length(aero)
    songname = aero(k).name;
    songfolder = aero(k).folder;
    readthis = strcat(songfolder,'/',songname);
    [aero_full{k}, aero_Fs{k}] = audioread(readthis);
end
% sound(aero_full{k}, Fs{k}); if you want to listen to the song

% section each song into 5 seconds
index = 1;
k = [1 2 3 5]; % we have different sampling rates
for kInd = 1:length(k)
    timestart = 5;
    timeend = 10;
    while timeend*aero_Fs{k(kInd)} < length(aero_full{k(kInd)})-5
        aero_sect(index,:) = ...
```

```
            aero_full{k(kInd)}
(timestart*aero_Fs{k(kInd)}:timeend*aero_Fs{k(kInd)});
        timestart = timeend;
        timeend = timeend + 5;
        index = index + 1;
    end

end
aero_sect = aero_sect';

index = 1;
k = [4];
for kInd = 1:length(k)
    timestart = 5;
    timeend = 10;
    while timeend*aero_Fs{k(kInd)} < length(aero_full{k(kInd)})-5
        aero_sect2(index,:) = ...
            aero_full{k(kInd)}
(timestart*aero_Fs{k(kInd)}:timeend*aero_Fs{k(kInd)});
        timestart = timeend;
        timeend = timeend + 5;
        index = index + 1;
    end

end
aero_sect2 = aero_sect2';
```

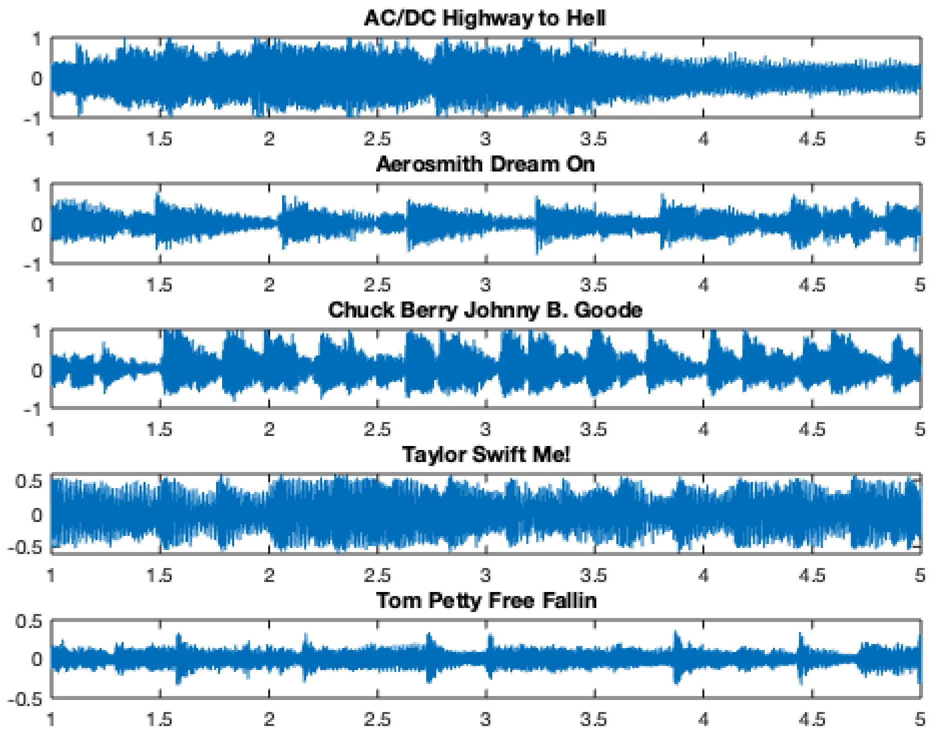
Read in Tom Petty

grab all the songs

```
tom = dir('Music_HW/TomPetty/*.m*');
% separate songs
for k = 1:length(tom)
    songname = tom(k).name;
    songfolder = tom(k).folder;
    readthis = strcat(songfolder,'/',songname);
    [tom_full{k}, tom_Fs{k}] = audioread(readthis);
end
% sound(tom_full{k}, Fs{k}); if you want to listen to the song

% section each song into 5 seconds
index = 1;
k = [2 3 4 5]; % we have different sampling rates
for kInd = 1:length(k)
    timestart = 5;
    timeend = 10;
    while timeend*tom_Fs{k(kInd)} < length(tom_full{k(kInd)})-5
        tom_sect(index,:) = ...
            tom_full{k(kInd)}
(timestart*tom_Fs{k(kInd)}:timeend*tom_Fs{k(kInd)});
        timestart = timeend;
        timeend = timeend + 5;
        index = index + 1;
    end

end
tom_sect = tom_sect';

index = 1;
k = [1];
for kInd = 1:length(k)
    timestart = 5;
    timeend = 10;
    while timeend*tom_Fs{k(kInd)} < length(tom_full{k(kInd)})-5
        tom_sect2(index,:) = ...
            tom_full{k(kInd)}
(timestart*tom_Fs{k(kInd)}:timeend*tom_Fs{k(kInd)});
        timestart = timeend;
        timeend = timeend + 5;
        index = index + 1;
    end

end
tom_sect2 = tom_sect2';
```

Plot Five Seconds of Each Artist

```
figure(1)
subplot(5,1,1)
t = linspace(1,5,length(acdc_sect(:,186)));
plot(t,acdc_sect(:,186)); title('AC/DC Highway to Hell')
subplot(5,1,2)
t = linspace(1,5,length(aero_sect(:,40)));
plot(t,aero_sect(:,40)); title('Aerosmith Dream On');
subplot(5,1,3)
t = linspace(1,5,length(chb_sect2(:,3)));
plot(t,chb_sect2(:,3)); title('Chuck Berry Johnny B. Goode');
subplot(5,1,4)
t = linspace(1,5,length(tay_sect(:,150)));
plot(t,tay_sect(:,150)); title('Taylor Swift Me!');
subplot(5,1,5)
t = linspace(1,5,length(tom_sect(:,30)));
plot(t,tom_sect(:,30)); title('Tom Petty Free Fallin');
```



SVDs of the Bands

```
% We take the SVD of the bands, this will give you the
% dominant spectrogram modes associated with a given band
lambda = 0.1;
[m,n] = size(acdc_sect);
mn = mean(acdc_sect,2);
acdc_sect = acdc_sect - repmat(mn,1,n);
[AR_acdc,~] = inexact_alm_rpca(acdc_sect.',lambda);
[acdc_u, acdc_s, acdc_v] = svd(AR_acdc.','Econ');

[m,n] = size(aero_sect);
mn = mean(aero_sect,2);
aero_sect = aero_sect - repmat(mn,1,n);
[AR_aero,~] = inexact_alm_rpca(aero_sect.',lambda);
[aero1_u, aero1_s, aero1_v] = svd(AR_aero,'Econ');

[m,n] = size(aero_sect2);
mn = mean(aero_sect2,2);
aero_sect2 = aero_sect2 - repmat(mn,1,n);
[AR_aero2,~] = inexact_alm_rpca(aero_sect2.',lambda);
[aero2_u, aero2_s, aero2_v] = svd(AR_aero2,'Econ');

[m,n] = size(chb_sect);
mn = mean(chb_sect,2);
```

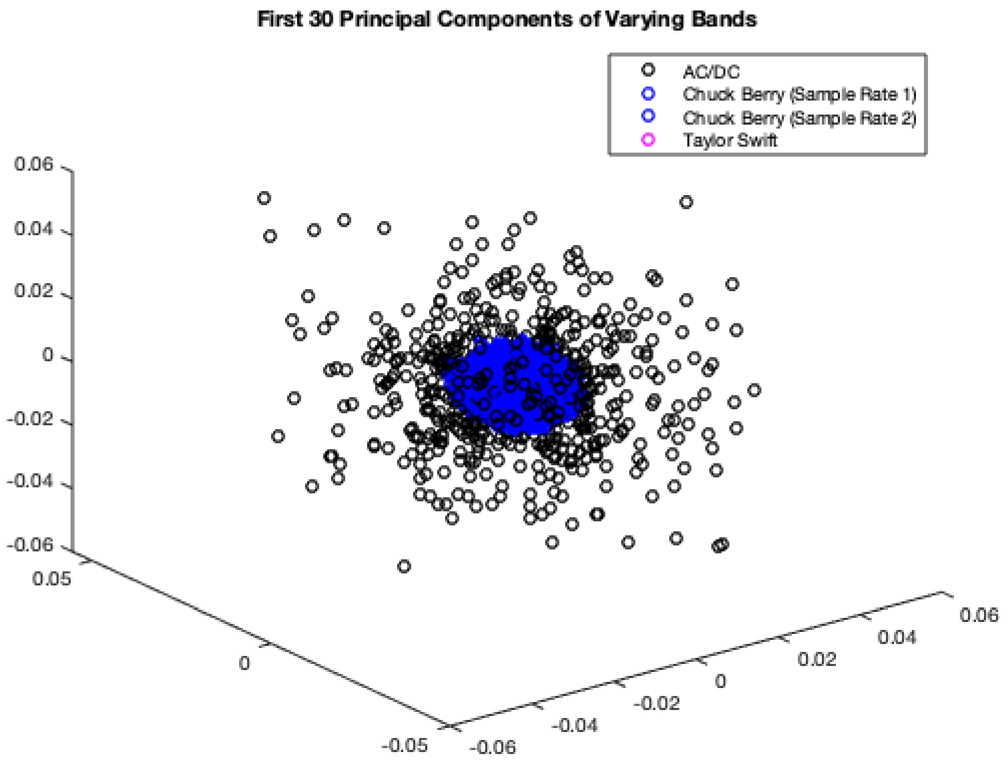
```
chb_sect = chb_sect - repmat(mn,1,n);
[AR_chb,~] = inexact_alm_rpca(chb_sect.',lambda);
[chb1_u, chb1_s, chb1_v] = svd(AR_chb,'Econ');

[m,n] = size(chb_sect2);
mn = mean(chb_sect2,2);
chb_sect2 = chb_sect2 - repmat(mn,1,n);
[AR_chb2,~] = inexact_alm_rpca(chb_sect2.',lambda);
[chb2_u, chb2_s, chb2_v] = svd(AR_chb2,'Econ');

[m,n] = size(tay_sect);
mn = mean(tay_sect,2);
tay_sect = tay_sect - repmat(mn,1,n);
[AR_tay,~] = inexact_alm_rpca(tay_sect.',lambda);
[tay_u, tay_s, tay_v] = svd(AR_tay,'Econ');

[m,n] = size(tom_sect);
mn = mean(tom_sect,2);
tom_sect = tom_sect - repmat(mn,1,n);
[AR_tom,~] = inexact_alm_rpca(tom_sect.',lambda);
[tom1_u, tom1_s, tom1_v] = svd(AR_tom,'Econ');

[m,n] = size(tom_sect2);
mn = mean(tom_sect2,2);
tom_sect2 = tom_sect2 - repmat(mn,1,n);
[AR_tom2,~] = inexact_alm_rpca(tom_sect2.',lambda);
[tom2_u, tom2_s, tom2_v] = svd(AR_tom2,'Econ');

sigma_acdc = diag(acdc_s);
for k = 1:length(sigma_acdc)
    energy_acdc(k) = sum(sigma_acdc(1:k))/sum(sigma_acdc);
end

sigma_aero1 = diag(aero1_s);
for k = 1:length(sigma_aero1)
    energy_aero1(k) = sum(sigma_aero1(1:k))/sum(sigma_aero1);
end
sigma_aero2 = diag(aero2_s);
for k = 1:length(sigma_aero2)
    energy_aero2(k) = sum(sigma_aero2(1:k))/sum(sigma_aero2);
end

sigma_chb1 = diag(chb1_s);
for k = 1:length(sigma_chb1)
    energy_chb1(k) = sum(sigma_chb1(1:k))/sum(sigma_chb1);
end
sigma_chb2 = diag(chb2_s);
for k = 1:length(sigma_chb2)
    energy_chb2(k) = sum(sigma_chb2(1:k))/sum(sigma_chb2);
end

sigma_tay = diag(tay_s);
for k = 1:length(sigma_tay)
    energy_tay(k) = sum(sigma_tay(1:k))/sum(sigma_tay);
```

```
end

sigma_tom1 = diag(tom1_s);
for k = 1:length(sigma_tom1)
    energy_tom1(k) = sum(sigma_tom1(1:k))/sum(sigma_tom1);
end
sigma_tom2 = diag(tom2_s);
for k = 1:length(sigma_tom2)
    energy_tom2(k) = sum(sigma_tom2(1:k))/sum(sigma_tom2);
end

rank_acdc = rank(acdc_s);
rank_aero1 = rank(aero1_s);
rank_aero2 = rank(aero2_s);
rank_chb1 = rank(chb1_s);
rank_chb2 = rank(chb2_s);
rank_tay = rank(tay_s);
rank_tom1 = rank(tom1_s);
rank_tom2 = rank(tom2_s);


mu =

    0.0074


mu_bar =

   7.3851e+04


rho =

    1.5000


mu =

    0.0071


mu_bar =

   7.0861e+04


rho =

    1.5000


mu =

    0.0176
```

```
mu_bar =

   1.7586e+05


rho =

    1.5000


mu =

    0.0118


mu_bar =

   1.1768e+05


rho =

    1.5000


mu =

    0.0097


mu_bar =

   9.6817e+04


rho =

    1.5000


mu =

    0.0025


mu_bar =

   2.4529e+04


rho =
    1.5000

#svd 10 r(A) 256 |E|_0 0 stopCriterion 0.0019939

mu =

    0.0254


mu_bar =

   2.5390e+05


rho =

    1.5000

#svd 10 r(A) 192 |E|_0 0 stopCriterion 3.4298e-15

mu =

    0.0159


mu_bar =

   1.5896e+05


rho =

    1.5000
```

Plot PCs 1-30 of AC/DC, Chuck Berry, Taylor Swift

```
figure(2)
k = 1:30;
for kInd = 1:length(k)
    plot3(acdc_v(1:20,k(kInd)),acdc_v(1:20,k(kInd)+1),...
        acdc_v(1:20,k(kInd)+2),'ko','Linewidth',[1.2]);
    hold on
    plot3(chb1_v(1:20,k(kInd)),chb1_v(1:20,k(kInd)+1),...
        chb1_v(1:20,k(kInd)+2),'bo','Linewidth',[1.2]);
    plot3(chb2_v(:,k(kInd)),chb2_v(:,k(kInd)+1),...
        chb2_v(:,k(kInd)+2),'bo','Linewidth',[1.2]);
    plot3(tay_v(1:60,k(kInd)),tay_v(1:60,k(kInd)+1),...
        tay_v(1:60,k(kInd)+2),'mo','Linewidth',[1.2]);
end
legend('AC/DC','Chuck Berry (Sample Rate 1)', ...
    'Chuck Berry (Sample Rate 2)', 'Taylor
 Swift', 'Location','northeast');
title('First 30 Principal Components of Varying Bands')
hold off
```



Build Training Sets for Test1

```
max_rank_test1 = min([rank_acdc,rank_chb1,rank_chb2,rank_tay]);
xacdc = acdc_v(:,1:max_rank_test1);
xchb1 = chb1_v(:,1:max_rank_test1);
xchb2 = chb2_v(:,1:max_rank_test1);
xtay = tay_v(:,1:max_rank_test1);

num_acdc_train = floor(length(acdc_sect(1,:))*1);
%num_acdc_test = length(acdc_sect(1,:)) - num_acdc_train;
num_chb_train1 = floor(length(chb_sect(1,:))*1);
%num_chb_test1 = length(chb_sect(1,:)) - num_chb_train1;
num_chb_train2 = floor(length(chb_sect2(1,:))*1);
%num_chb_test2 = length(chb_sect2(1,:)) - num_chb_train2;
num_tay_train = floor(length(tay_sect(1,:))*1);
%num_tay_test = length(tay_sect(1,:)) - num_tay_train;
%labels = [ones(num_acdc_train,1); 2*ones(num_chb_train1,1); ...
%         2*ones(num_chb_train2,1); -1*ones(num_tay_train,1)];

acdc_train = randperm(num_acdc_train);
%acdc_test = randperm(num_acdc_test);
```

```
chb_train1 = randperm(num_chb_train1);
%chb_test1 = randperm(num_chb_test1);
chb_train2 = randperm(num_chb_train2);
%chb_test2 = randperm(num_chb_test2);
tay_train = randperm(num_tay_train);
%tay_test = randperm(num_tay_test);

xtrain = [xacdc(acdc_train,:); xchb1(chb_train1,:); ...
    xchb2(chb_train2,:); xtay(tay_train,:)];
%xtest = [xacdc(acdc_test,:); xchb1(chb_test1,:); ...
%    xchb2(chb_test2,:); xtay(tay_test,:)];

%[pre,err] = classify(xtest,xtrain,labels,'mahalanobis');

%figure(3)
%bar(pre);
```

Test1 tables

```
T1T1 = table(xacdc(acdc_train,:), repmat({'ACDC'},1,num_acdc_train)');
T1T2 =
 table(xchb1(chb_train1,:),repmat({'ChuckBerry'},1,num_chb_train1)');
T1T3 =
 table(xchb2(chb_train2,:),repmat({'ChuckBerry'},1,num_chb_train2)');
T1T4 = table(xtay(tay_train,:),
 repmat({'TaylorSwift'},1,num_tay_train)');
Test1Table = [T1T1; T1T2; T1T3; T1T4];
```

Build Training Sets for Test2

```
max_rank_test2 =
 min([rank_acdc,rank_aero1,rank_aero2,rank_tom1,rank_tom2]);
xacdc = acdc_v(:,1:max_rank_test2);
xaero1 = aero1_v(:,1:max_rank_test2);
xaero2 = aero2_v(:,1:max_rank_test2);
xtom1 = tom1_v(:,1:max_rank_test2);
xtom2 = tom2_v(:,1:max_rank_test2);

num_acdc_train = length(acdc_sect(1,:));
num_aero_train1 = length(aero_sect(1,:));
num_aero_train2 = length(aero_sect2(1,:));
num_tom_train1 = length(tom_sect(1,:));
num_tom_train2 = length(tom_sect2(1,:));

acdc_train = randperm(num_acdc_train);
aero_train1 = randperm(num_aero_train1);
aero_train2 = randperm(num_aero_train2);
tom_train1 = randperm(num_tom_train1);
tom_train2 = randperm(num_tom_train2);

xtrain = [xacdc(acdc_train,:); xaero1(aero_train1,:); ...
    xaero2(aero_train2,:); xtom1(tom_train1,:); ...
    xtom2(tom_train2,:)];
```

Test2 tables

```
T2T1 = table(xacdc(acdc_train,:), repmat({'ACDC'},1,num_acdc_train)');
T2T2 =
 table(xaero1(aero_train1,:),repmat({'Aerosmith'},1,num_aero_train1)');
T2T3 =
 table(xaero2(aero_train2,:),repmat({'Aerosmith'},1,num_aero_train2)');
T2T4 = table(xtom1(tom_train1,:),
 repmat({'TomPetty'},1,num_tom_train1)');
T2T5 = table(xtom2(tom_train2,:),
 repmat({'TomPetty'},1,num_tom_train2)');
Test2Table = [T2T1; T2T2; T2T3; T2T4; T2T5];
```

Build Training Sets for Test3

```
max_rank_test3 = min([rank_acdc,rank_aero1,rank_aero2,...
    rank_chb1,rank_chb2,rank_tay,rank_tom1,rank_tom2]);
xacdc = acdc_v(:,1:max_rank_test3);
xaero1 = aero1_v(:,1:max_rank_test3);
xaero2 = aero2_v(:,1:max_rank_test3);
xchb1 = chb1_v(:,1:max_rank_test3);
xchb2 = chb2_v(:,1:max_rank_test3);
xtay = tay_v(:,1:max_rank_test3);
xtom1 = tom1_v(:,1:max_rank_test3);
xtom2 = tom2_v(:,1:max_rank_test3);
```

Test3 tables

Test3 uses training sets from Test1 and Test2

```
T3T1 = table(xacdc(acdc_train,:),
 repmat({'ClassicRock'},1,num_acdc_train)');
T3T2 =
 table(xaero1(aero_train1,:),repmat({'ClassicRock'},1,num_aero_train1)');
T3T3 =
 table(xaero2(aero_train2,:),repmat({'ClassicRock'},1,num_aero_train2)');
T3T4 = table(xtom1(tom_train1,:),
 repmat({'ClassicRock'},1,num_tom_train1)');
T3T5 = table(xtom2(tom_train2,:),
 repmat({'ClassicRock'},1,num_tom_train2)');
T3T6 =
 table(xchb1(chb_train1,:),repmat({'NotRockMan'},1,num_chb_train1)');
T3T7 =
 table(xchb2(chb_train2,:),repmat({'NotRockMan'},1,num_chb_train2)');
T3T8 = table(xtay(tay_train,:),
 repmat({'NotRockMan'},1,num_tay_train)');
Test3Table = [T3T1; T3T2; T3T3; T3T4; T3T5; T3T6; T3T7; T3T8];
```

Published with MATLAB® R2019b