CS170A — Mathematical Modeling & Methods for Computer Science
HW#2 *Eigenfaces*
Due: 3:59pm Monday February 24, 2014

D. Stott Parker, Xingyu Ma, Costas Sideris

Using CourseWeb, please turn in your Matlab code and resulting output (enough to show that the program is working) as a PDF document.

1. **Eigenfaces** Chapter 11 of the course notes is on Eigenfaces. For this assignment you can use the files in the directory `old_faces`, which includes some Matlab scripts and a database of 177 face images, each a grayscale `.bmp` bitmap file of size 64×64 pixels. The face images have been pre-processed so that the background and hair are removed and the faces have similar lighting conditions.

   The notes explain how to reshape each face image into a $1 \times 64^2 = 1 \times 4096$ row vector, and collect them into a matrix. The principal components of the matrix then define the main dimensions of variance in the faces. The program `eigenfaces.m` shows how to do this. These principal components are called *eigenfaces*.

   The goal of this problem is to apply the same ideas to a new set of 200 faces in the directory `new_faces`. The subdirectory `new_faces/faces` has 200 faces that have been normalized, cropped, and equalized. The subdirectory `new_faces/smiling_faces` has 200 images of the same people, but they are smiling. Each of these images is a grayscale `.jpg` file with size 193×162.

Figure 1: The 64×64 face bitmap image `old_faces/face000.bmp`, along with 193×162 jpeg images `new_faces/faces/23a.jpg` and `new_faces/smiling_faces/23b.jpg`. PLEASE USE THE MATLAB `imread()` FUNCTION TO READ IN IMAGES.

   Using Matlab, perform the following steps:

   (a) Modify the program `eigenfaces.m` (available in this directory) to use the `new_faces` images instead of the `old_faces` images. Also, modify it to use the Matlab function `imresize` to downsample each of the new faces by a factor of 3, so it is 64×54 (instead of 193×162). Then: PAD both sides of the image with `zeros(64,5)` so the result is a 64×64 image.

   (b) Create a function that takes as input a string array of filenames of face images, an integer k, and an integer sample size s — and yields the average face and the first k singular values and eigenfaces as output values for a sample of size s.

   Apply your function to both the `new_faces/faces` and the `new_faces/smiling_faces` directories, and plot the absolute value of the difference between your average face and (your downsampled version of) the average face provided in the directory. (You can use the `imagesc` function to display images with automatic rescaling of numeric values.)

   (c) If your mean normal face is $\overline{\mathbf{f}}_0$, and your mean smiling face is $\overline{\mathbf{f}}_1$, render (using `imagesc`) the difference $\overline{\mathbf{f}}_0 - \overline{\mathbf{f}}_1$ (the average difference between normal faces and smiling faces).

   (d) Using your downsampled images, perform PCA on each set of faces (normal and smiling).

   For each image (normal or smiling), construct its $64^2 \times 1$ vector $\mathbf{f}$. Then, subtract the average face ($\overline{\mathbf{f}}_0$ or $\overline{\mathbf{f}}_1$) and project the remainder on the first $k = 60$ eigenfaces. For example, with a smiling face, the projection of $\mathbf{f}$ on the j-th smiling eigenface $\mathbf{e}_j$ is

$$c_j \;=\; \langle\, (\mathbf{f} \,-\, \mathbf{f}_1),\, \mathbf{e}_j \,\rangle \qquad (j = 1, \ldots, k).$$

For each set of faces (normal or smiling), make one large scree plot for the set, showing all sequences of the first k coefficients for each image overplotted (e.g. with `hold on`). ????

(e) Let us say the *unusualness* of a face is the L_2 norm of its eigen-coefficient vector.
Determine, for each set, the most unusual face. Normal: Face 124
Smiling: 69

(f) Develop three different face classifiers using the eigenfaces you've computed; each should be a function that, given a face image $\mathbf{f}$ as input, yields the output value 1 if $\mathbf{f}$ is smiling, and 0 otherwise.
Specifically, implement the following 3 classifiers that take an input image $\mathbf{f}$: N: 1 S: 0

i. *Classifier X*: yield 1 if the normal face unusualness of $\mathbf{f}$ is less than smiling face unusualness of $\mathbf{f}$, else 0.

ii. *Classifier Y*: yield 1 if $\| \mathbf{f} - \mathbf{f}_0 \|^2 \ \geq \ \| \mathbf{f} - \mathbf{f}_1 \|^2$, else 0. N: 1 S: 1

iii. *Classifier Z*: if C_0 is the covariance matrix for normal faces, and C_1 is the covariance matrix for smiling faces, yield 1 if $\| \mathbf{f} - \mathbf{f}_0 \|_{C_0}^2 \ \geq \ \| \mathbf{f} - \mathbf{f}_1 \|_{C_1}^2$, where $\| \mathbf{x} \|_C^2 \ = \ \mathbf{x}' \, C^{-1} \, \mathbf{x}$, else 0.

Using each of these three classifiers, determine the classification it yields for the two most unusual images you found in the previous question.

(g) Write a function `[Sublist1 Sublist2] = randsplit(List)` that takes an array `List` of length `n` and splits it randomly into two sublists of size `floor(n/2)` and `ceil(n/2)`. (Hint: `randperm`)
Use `randsplit` to split each of the 200-face sets into a training subset and testing subset of equal size.
For both sets of faces (100 normal faces and 100 smiling faces) in the training set, compute:
both average faces $\mathbf{f}_0$, $\mathbf{f}_1$, both face covariance matrices C_0, C_1, and both sets of eigenfaces.

(h) For each of the three Classifiers (X, Y, Z) above: error = 0.75, 0.25 x, y

i. classify each of the 200 faces $\mathbf{f}$ in the testing set, and count classification errors.

ii. compute the *error rate* (percentage of errors in test face classifications) for the Classifier.

Then rank the three classifiers by their error rate.

2. **Face Compression**
In the previous problem you produced a scree plot showing the first 60 eigenface coefficients for each face, and determined the most unusual face. Text
For each 64×64 image X from your (downsampled) smiling faces, compute the following sorted sequences:

(a) sorted absolute values of eigenface coefficients for X

(b) sorted absolute values of coefficient values in the two-sided FFT of X (in Matlab: `fft2(X)`) DCT 0.0657

(c) sorted absolute values of coefficient values in the two-sided DCT of X (in Matlab: `dct2(X)`) FFT 0.0448
PCA: 0.0968

(d) sorted singular values from the SVD of X.

We get an *image compression* scheme if we keep only the first k coefficients, and discard the rest. W

Suppose we keep only the first 60 coefficients. If we compute *compression error* as (the L_2 norm of absolute values of coefficients after the first 60) divided by (the L_2 norm of absolute values of all coefficients), we can compare compression with each of the 4 transforms above.

For the smiling test set (the last 100 smiling images), compute the average compression error for the 4 transforms. Rank the 4 transforms above by their compression error.

3. **Floating Point Horror**
Using the Matlab command `format long` to make the following floating point results visible.
For each of the following matrices `X`:

- print the results of `svd(X)`, `log10(svd(X))`, and `range(log10(svd(X)))`, `cond(X)`.
- print the Frobenius norm of the difference between the product `X * inv(X)` and the identity matrix.
- Write a program that computes the *exact* inverse $\mathtt{X}^{-1}$ of `X`. (With the `hilb(n)` matrix, this is computed by `invhilb(n)` for example, but you may need to do a little research on the internet to get the form of the exact inverse.) Then print the Frobenius norm of the difference between the product $\mathtt{X} * \mathtt{X}^{-1}$ and the identity matrix.

(a) The Hilbert matrix `hilb(12)`.

(b) The Pascal matrix `pascal(12)`.

(c) The Fourier matrix `fft(eye(12))`.

(d) The Vandermonde matrix `vander(1:12)`.