

Problem Set 2

Face Detection and Recognition Using PCA and PPCA

Prof. Jim Rehg
CS 7635 Computational Perception
College of Computing
Georgia Institute of Technology

February 5, 2002

Your results for this homework set will consist of some written responses, some printouts of variables and plots, and some Matlab functions. The deliverables are listed explicitly for each subproblem. Following our naming convention for functions and variables will make it easier to grade your work and assign partial credit. Your solutions must be handed in at the beginning of class on the due date. Feel free to do your write-up in long hand, as long as it is organized and legible. You should email a zip file containing your matlab functions to Hao Wang (wanghao@cc.gatech.edu) by 5pm on the due date.

Note that you will be asked to compare the performance of several different approaches, and that some of the code you will be developing will be used by multiple problems. It would be a good idea to read the problem set completely before writing any code. It will also be helpful if you save and organize the results from each of the subproblems, so you won't have to redo any work.

You may want to look at the Matlab tutorial pages which are accessible from the course web page.

1. Preliminaries [5 points]

Download and extract the file *empca.zip* from the class web site. The file contains matlab code for computing PCA using the EM algorithm.¹ The file *empca.m* contains the main function and describes how to call it. The file *truepca.m* contains a straight-forward implementation of PCA based on the matlab *eig* function.

empca.zip also contains a test matrix, *test.mat*, which contains 10 test images of 100 by 75 pixels consisting of different patterns of vertical stripes. Each column of *test* contains a single image, stored in column major order. In other words, if *I* is the *i*th image, then $\text{test}(:,i) = \text{reshape}(I, 100 * 75, 1)$. To display the *i*th image, use $\text{imshow}(\text{reshape}(\text{test}(:,i), 100, 75))$. Use *empca* to find the first 5 eigenvectors and eigenvalues of *test*. Verify that you get the eigenvalues $1.0e+007 * (4.6500, 1.1917, 0.5521, 0.3294, 0.2276)$.

Download and extract the file *faceata.zip* from the class web-site. It contains face and nonface images stored as .mat files. The face data has been taken from the CMU PIE database.² The file *face.mat* contains five matrices: *neutral*, *smile*, *talk*, *light1*, and *light2*. Note that you can use *who* after loading .mat files to see what variables are defined. Each matrix is 7500 by 20, and contains one 100 by 75 image for each of 20 subjects. The subject order is the same for all matrices. So a single matrix containing all of the images for the *i*th subject can be assembled as $\text{subject}\{i\} = [\text{neutral}(:,i), \text{smile}(:,i), \text{talk}(:,i), \text{light1}(:,i), \text{light2}(:,i)]'$. The file *nonface.mat* contains a set of 200 nonface images. The zip file also contains four images, *face1.gif*, *face2.gif*, *nonface1.gif*, and *nonface2.gif*, which will be used in question 2(e) for face detection.

The face images have *already* been preprocessed: Using hand-labeled feature locations, the images have been scaled and rotated so the centers of the eyes and nose are aligned across all of the images. The images have been cropped to include only the interior of the face. The lower left and right corners of the images have been masked out to eliminate background pixels. You must perform the remaining preprocessing step of *intensity normalization*.

Write a matlab function *imagenorm* which takes an arbitrary gray scale image as input and normalizes it so the pixels have a mean value of zero and a variance of 1. The functions *mean* and *std* will be useful here. Process all of the face and nonface images with *imagenorm*. Apply *imagenorm* to the test matrix and then

¹The code was written by Dr. Sam Roweis, a professor at the Univ. of Toronto. It can also be found on his web site, <http://www.cs.toronto.edu/~roweis>.

²See <http://hid.ri.cmu.edu/Hid> for more information on the database, including a technical report.

compute the first five eigenvalues.

Deliverables (Preliminaries):

- Code for *imagenorm*
- Print the first two eigenvalues of the intensity-normalized test matrix:
- How does intensity normalization affect the eigenvalues and eigenvectors of *test*?

2. Face Detection [50 points]

In this problem we will compare the performance of classifiers based on Nearest-Neighbor (NN) and PCA for the task of face detection. We will use a few samples from their ROC curves to compare the overall performance of the classifiers. We will also evaluate the performance in practice on a few test images. We will measure the speed of the different classifiers (i.e. the average time required to classify an input). The *cpitime* function will be useful in this task.

For performance evaluation and testing, we will divide the face and nonface data into training and testing sets. Let the training set consist of the first ten people (i.e. columns 1-10 of the face matrices) and the first half of the nonface images (i.e. columns 1-100 of *nonface*). Let the remaining images be the testing set. It goes without saying that you will *never* train on your testing images!

(a) NN detector In a nearest-neighbor classifier, the training data itself is the “model.” Given an input image y (also called a *probe*), the NN classifier will assign to y the label associated with the closest image in the training set. So if y happens to be closest to another face it will be assigned $L = 1$ (face), otherwise it will be assigned $L = 0$ (nonface). We will use the Euclidean distance D to measure how close together two images are. For two image vectors (i.e. matrix columns) y_1, y_2 , we have $D = \|y_1 - y_2\|^2$. Write a function *NNclassify* that outputs the label L for a probe y given training faces *ftrain* and nonfaces *nftrain*. So an example of the calling convention would be:

```
ftrain = [neutral(:,1:10),smile(:,1:10),talk(:,1:10),
          light1(:,1:10),light2(:,1:10)];
nftrain = nonface(:,1:100);
L = NNclassify(y, ftrain, nftrain);
```

Write a function *NNtest* which outputs the percentage of correct detections, R_C , and percentage of false positives, R_F , over the face and nonface testing data.³ A false positive occurs when a probe in the nonface class is assigned to the face class by the NN rule.

Evaluate the performance of the NN classifier in the following way: First, check your code by testing it on your training data. Second, evaluate the performance on all of the testing data. Since there is no threshold or tuning parameter for this classifier, its ROC curve consists of the single point given by (R_F, R_C) over all of the test data. Third, evaluate the performance separately for each of the three categories of face test images listed below.

Deliverables (NN detector):

- Code for *NNclassify* and *NNtest*
- Fill in the table:

	Neutral + Talk	Smile	Light1 + Light2	Total
R_C				

- Comment on the performance of the NN rule. By examining some of the examples where the detector failed, can you explain the detection performance above?

- What is the average *cputime* required by *NNclassify*?

³These are also known as the *detection rate* and *false alarm rate*, respectively.

y as face or nonface. Intuitively, we expect $d_n(y) > d_f(y)$ to suggest that y is a face. The LR for PCA is denoted $\Delta_d(y)$. The LR test is defined as:

$$\Delta_d(y) = \frac{d_n(y)}{d_f(y)} \underset{L=0}{\overset{L=1}{>}} \eta,$$

where η is a threshold parameter. For example, when $\Delta_d(y) > \eta$, the classifier labels y as a face.

Write a function *PCAclassify* which implements Δ_d and outputs a classification of an input image as a function of η . Write a function *PCAtest*, analogous to *NNtest*, which computes R_C and R_F on the testing set as a function of η .

Through experimentation, find four values for η that lead to fairly evenly-spaced (R_C, R_F) pairs (i.e. sample the interesting part of the ROC curve as evenly as you can). Let *RCdata* and *RFdata* be vectors containing the R_C and R_F values for the thresholds *Tdata*. Plot the ROC curve defined by these points (e.g. use *plot*). Make sure the axes have reasonable scales and tick marks for the range 0-1.

Deliverables (PCA classifier):

- Code for *PCAclassify* and *PCAtest*
- Printouts of *RCdata*, *RFdata*, *Tdata* and the ROC curve plot. Mark the location of the NN classifier on this ROC curve.
- Print $d_n(\text{neutral}(:,1))$ and $d_f(\text{nonface}(:,1))$, the distances from nonface/face space for the first face and nonface training images:
- What is the average *cputime* required by *PCAclassify*?

(d) Face Detector Write a function *FaceDetect* which scans *PCAclassify* across an input image and detects all of the up-right, frontal faces that it contains. You do not need to search over different orientations in the image. You will, however, need to consider different spatial scales (sizes) in doing detection. Thus the problem reduces to evaluating *PCAclassify* at all of the possible (x, y) positions that an appropriately-sized set of eigenimages can take in the input image. You can easily scale the eigenimages to the correct size using *imresize*. An ideal program would search over a range sizes and report all of the faces that it found. However, you

may prefer to manually tune the detector size to the faces in the particular test images. Your program should accept a threshold and output a list of all of the locations at which faces were detected.

Write a function *FaceDisplay* which takes the input image and the list of face locations produced by *FaceDetect* and draws boxes around all the detected faces. The following command draws a thick white box at the location (x, y) :

```
plot([x x+75 x+75 x x],[y y y+100 y+100 y],  
      'w-', 'LineWidth', 3)
```

Use the ROC curve for PCAclassify to pick a good threshold setting. Then apply *FaceDetector* to the test images *face1.gif*, *face2.gif*, *nonface1.gif*, and *nonface2.gif*. Use *FaceDisplay* on the output. Was the result what you expected? It may be interesting to experimentally determine the threshold setting at which the true faces in *face1* and *face2* are successfully detected. Why is such a threshold setting guaranteed to exist? How many false positives per image are produced at that setting?

Deliverables (Face Detector):

- Code for *FaceDetect* and *FaceDisplay*
- Printout of the four test images with white boxes overlaid on all of the detections. What threshold setting did you use?
- Calculate the value of R_F that would be required in order for your face detector to make one false detection (on average) for an image the size of *face1*. Locate the corresponding point (approximately) on the combined ROC curves. What are the corresponding detection rates?
- How would you explain the performance differences, in terms of accuracy and run-time, between the NN and PCA detectors? What could you do to improve the detection performance?

3. Face Recognition [45 points]

The face recognition task which we will address in this problem is known as M -way classification. We are given a database of M subjects and it is known that each probe image *will* correspond to one of the subjects. Thus the problem reduces to determining which one of M known classes the test image belongs to.

Our main goal will be to study the role of *intrapersonal* and *extrapersonal* factors in determining recognizer performance. The intrapersonal factors are the variations between images of the *same person* due to changes in lighting, pose, and facial expression. In contrast, the extrapersonal variations between images of *different people* include differences in identity. If the intrapersonal variations are completely controlled (e.g. by fixing the head pose, lighting, and facial expression for all subjects), then all of the intensity differences between face images will be due to differences in identity. In this case, recognition is as easy as possible. In practice, however, both testing and training images will contain a mixture of intrapersonal and extrapersonal factors.

(a) Nearest-Neighbor PCA A standard approach to PCA-based face recognition is to use a NN classifier within the subspace defined by PCA. Use the PCA subspace defined by U_f , which you calculated in problem 2(c). You will perform recognition experiments using the remaining 10 subjects (numbered 11-20) in the face dataset (i.e. $M = 10$).

Write a function $NNtestM$ which takes multi-class training and test images and outputs N_C , the number of faces in the testing set (across all of the subjects) that were correctly classified. Use the Euclidean distance measure as in $NNtest$. Your program should support arbitrary numbers of training and testing images per subject. By calling $NNtestM$ on specific categories of training and test data such as smile, light1, etc. we can study the performance as a function of the types of variation in the images. We can implement PCA-NN with $NNtestM$ by projecting the training and testing data before calling the function.

Deliverables:

- Code for $NNtestM$
- Perform four experiments and fill out the following four tables. The experiments differ in the choice of distance function (full-NN vs. PCA-NN) and in the training data (neutral vs. neutral+talk+light2). For each experiment,

you are to record a separate N_C for each of the categories of testing images (talk+smile+light1+light2 vs. smile+light1).

1. Classifier: *full-NN*
Training: *Neutral*

	Smile	Talk	Light1	Light2	Total
N_C :					

2. Classifier: *PCA-NN*
Training: *Neutral*

	Smile	Talk	Light1	Light2	Total
N_C					

3. Classifier: *full-NN*
Training: *Neutral + Talk + Light2*

	Smile	Light1	Total
N_C			

4. Classifier: *PCA-NN*
Training: *Neutral + Talk + Light2*

	Smile	Light1	Total
N_C			

- Explain the performance you have observed above:

(b) PPCA Recognition At the heart of the previous approach to recognition is the problem of scoring intensity differences between pairs of face images. For example, in the full-NN method, the comparison between a probe image y_p and a training image y_t is scored as $\|y_p - y_t\|^2$. The intensity differences $\Delta y = y_p - y_t$ are a complex function of the variation in lighting, expression, and identity between the two images. If the intrapersonal and extrapersonal causes could be separated somehow, recognition performance should improve.⁴

Using the images of the first 10 subjects in the face dataset (the training gallery) you will construct Y_I and Y_E , the intrapersonal and extrapersonal training sets. Y_I is constructed by computing intensity differences between all pairs of images of the *same subject* in the training gallery. Y_I is 7500 by 100 and its columns consist of 'talk(:,i) - neutral(:,i)', 'smile(:,i) - light1(:,i)', etc. where i ranges from 1 to 10. Y_E consists of intensity differences between all pairs of images of *different subjects*. Its columns consist of 'talk(:,j) - neutral(:,i)' and so forth where $i \neq j$ and i, j range from 1 to 10. How many columns does Y_E have? Note that both Y_E and Y_I contain variations due to lighting and expression, but only Y_E contains variations due to identity.

You will construct separate Probabilistic PCA models for the intrapersonal and extrapersonal classes of image differences. The generative model for an image difference Δy is given by $\Delta y = Cx + v + \mu$ and $\Delta y \sim N(\mu, CC^T + \sigma^2 I)$. Find subspace dimensions m_I and m_E that capture 95% of the variance in the two classes. Let σ_I^2 and σ_E^2 denote the "left-over" variance in the PPCA models for the given choices of m_I and m_E . From a linear algebra prospective, develop a formula for σ^2 that does not require the computation of all the eigenvalues. How would this estimate compare to an estimate based on a full-rank sample covariance matrix, if sufficient data were available? We will use an alternative estimate proposed by Dr. Baback Moghaddam: Assume that $\lambda_i = \lambda_{p-1}, p \leq i \leq n$. Derive a second formula for σ^2 under this assumption. You will compute σ_I^2 and σ_E^2 using this second formula.

Construct separate PPCA models for Y_I and Y_E after intensity normalization. Let $p_I(\Delta y)$ and $p_E(\Delta y)$ be the probabilities of a vector of intensity differences due to intrapersonal and extrapersonal variations, respectively. This results in a

⁴This approach is based on the paper "Beyond Eigenfaces: Probabilistic Matching for Face Recognition", by Baback Moghaddam, Wasiuddin Wahid, and Alex Pentland. Proc. 3rd Intl. Conf. on Automatic Face and Gesture Recognition, Nara, Japan, April 1998.

new “distance function” for image matching:

$$d_r(y_p, y_t) = \frac{p_I(y_p - y_t)}{p_I(y_p - y_t) + p_E(y_p - y_t)},$$

where we have assumed that the prior probabilities of the intrapersonal and extrapersonal labels are equal. This new distance measure is a direct replacement for the Euclidean distance in *NNtestM*.

Repeat the experiments from part (3b) using the the new distance measure.

Deliverables:

- Print m_I , m_E , σ_I^2 , and σ_E^2 :
- Responses to questions about σ^2 and various formulae:

- Perform two experiments and fill out the following two tables. The experiments differ in the choice of training data (neutral vs. neutral+talk+light2).

1. Training: *Neutral*

	Smile	Talk	Light1	Light2	Total
N_C :					

2. Training: *Neutral + Talk + Light2*

	Smile	Light1	Total
N_C			

- How does this approach compare to PCA-NN?

4. Extra Credit: Interpretation of Factors [+20 points]

The vector of hidden factors x in the PPCA model can be interpreted as “explaining” the common elements in the input data. Since the variability in the Y_I dataset is due to specific changes in illumination and expression, we might hope to find factors for “expression” and “lighting” in the PPCA model of this data. Form the “smile” and “light1” data matrices Y_s and Y_l by extracting the columns from Y_I that contain differences with the neutral face. The columns of Y_s have the form ‘smile(:,i) - neutral(:,i)’ for i from 1 to 10. The columns of Y_l have the form ‘light1(:,i) - neutral(:,i).’

For any given column of Y_s and Y_l you can compute $E\{x|y\}$, the mean of the conditional distribution of the factors. Compute $\hat{x}_s = E\{x|Y_s\}$ and $\hat{x}_l = E\{x|Y_l\}$. This can be done by appropriate modification of the formula for $E\{x|y\}$. It is tempting to interpret these means as the average values of “smile-ness” and “light1-ness” in the dataset. Examine this hypothesis by synthesizing some sample images using the equations:

$$\begin{aligned}y_i^s &= y_i^n + CE\{x|Y_s\} + v + \mu \\y_i^l &= y_i^n + CE\{x|Y_l\} + v + \mu,\end{aligned}$$

where y_i^n is the neutral face image for the i th subject. Compare y_i^s and y_i^l to their corresponding images in the training gallery. Synthesize with $v = 0$ and with v drawn at random from the appropriate distribution. Test this idea further by synthesizing some examples using one or more neutral images in the testing set.

A further step would be to synthesize by drawing random samples from the conditional factor densities, $p(x|Y_s)$ and $p(x|Y_l)$. It would also be interesting to synthesize “blends” of the two factors.

Comment on the utility of PPCA as an image synthesis tool and on the interpretability of PPCA factors.