STONY BROOK UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CSE 527: INTRODUCTION TO COMPUTER VISION

Handed out: February 17, 2009 **Due on:** March 12, 2009

Problem Set #2

Submit the homework by email to cse527@cs.sunysb.edu

Problem 1: ***PCA algebraic derivation***
Derive the third PCA basis vector by writing down the objective function and maximizing it.

Problem 2: ***Image pre-processing***
Download the file `truepca.m` which contains a straight-forward implementation of PCA based on the matlab `eig` function. While this algorithm works well with small data sets it has memory problems with larger ones. Recall the relationship between the singular value decomposition and the eigenvalue decomposition and modify the algorithm appropriately. Employ the economy size `svd` in matlab, `svd(…,0)`.

Download and extract the file *facedata.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. 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
`subject{i}= [neutral(:,i),smile(:,i),talk(:,i),light1(:,i),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 3(d) 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 compute the first five eigenvalues.

Deliverables (Image pre-processing):
- 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*?
- Explain why this intensity normalization necessary

Problem 3: ***Face Detection***
In this problem we will compare the performance of classifiers based on Nearest- Neighbor (NN) and PCA for the task of face detection.

A Receiver Operating Characteristic curve (or ROC curve) is a plot of the true positive rate against the false positive rate for the different possible cutpoints of a diagnostic test. 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 `cputime` function will 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!

However, as a simple sanity check, in order to prove to yourself that your implementation is what you actually intended to implement, test your procedure on your training images. You should have 100% accuracy, otherwise you have a bug. Of course, 100% accuracy in your sanity test is not proof of a correct implementation of an algorithm. This can only be achieved by testing your algorithm on "corner cases".

(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 it 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. vectorized images) $\mathbf{y}_1$, $\mathbf{y}_2$, we have $D=\| \mathbf{y}_1 - \mathbf{y}_2 \|^2$.

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. These are knowns as the detection rate and false alarm rate respectively. 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`?

(b) PCA Preliminaries: We will compute *separate* subspace models for faces and nonfaces. We will be performing PCA on $\mathbf{S}_f$ and $\mathbf{S}_n$, the sample covariance matrices for the face and nonface training images, respectively.
A basic question is: How many eigenvectors should be used? Let

$$K_{var}(m) = \frac{\sum_{i=1}^{m} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \qquad \text{where } m \leq n ,$$

denote the fraction of the total variance which is captured in the first *m* eigenvalues of a sample covariance matrix **S** .
Find , the *minimum* number of eigenvalues for the face and nonface
PCA models, such that $K_{var}(m_f)>0.95$ and $K_{var}(m_n)>0.95$.
.

Deliverables:

- Responses to questions about K_{var} :
- Print m_f and m_n :
- Print the first two eigenvalues for the face and nonface PCA models

(c) PCA Detection: Let $\mathbf{U}_f$ and $\mathbf{U}_n$ denote the PCA projection matrices for the face and nonface subspace models defined above. These projection matrices contain the top m_f and m_n eigenvectors, respectively. Similarly, we will define the "distance from subspace" measures d_f and d_n, where $d_f(\mathbf{y}) = \left\| \mathbf{y} - \mathbf{U}_f \mathbf{U}_f^T \mathbf{y} \right\|^2$ and d_n is similarly defined. We will use a likelihood ratio (LR) test to classify a probe $\mathbf{y}$ as face or nonface. Intuitively, we expect $d_n(\mathbf{y}) > d_f(\mathbf{y})$ to suggest that $\mathbf{y}$ is a face. The LR for PCA is denoted $\Delta_d(\mathbf{y})$. The LR test is defined as:

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

where η is a threshold parameter. For example, when $\Delta_d(\mathbf{y}) > \eta$, the classifier labels $\mathbf{y}$ 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`, analagous 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`
- Explain why are we using a subset of all possible basis vectors to model the data and how is the recognition rate impacted by this decision
- Prove that if we use all the basis vectors, a new image is equivalent to representing it as a linear combination of the training images.
- Printouts of *RCdata, RFdata, Tdata* and the ROC curve plot. Mark the location of the NN classifier on this ROC curve.
- Print d_n (neutral(:,1)) and d_f(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. To speed up the search you will want to implement a multi-resolution strategy. Search a low-res version of the input image and search the image at a higher resolution where a face is most likely to be present given the suggested output of the low-res search.

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 `FaceDetect` 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?

Problem 4: ***Face Recognition***

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.

Download from the course web page freiburg_train.mat and freiburg_test.mat The freiburg_train.mat contains a facial image data tensor of 75 people, 6 viewpoints (viewpoint angles $\theta = \pm 35, \pm 20, \pm 5; \phi = 0$), 6 illuminations (illumination angles $\theta = 90 \pm 35, \pm 20, \pm 5; \phi = 45$). The test data set, freiburg_test.mat contains a facial image data tensor of 75 people, 9 viewpoints (viewpoint angles $\theta = \pm 30, , \pm 25, \pm 15, \pm 10, 0; \phi = 0$), 9 illuminations (illumination angles $\theta = 90 \pm 30, , \pm 25, \pm 15, \pm 10, 0; \phi = 0$).

(a) EigenFaces (Nearest-Neighbor PCA):
A standard approach to PCA-based face recognition is to use a NN classifier within the subspace defined by PCA. 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.

Reorganize the data tensor into a data matrix using the `matricize` function from Tensor Decomposition Toolbox. (Download the Tensor Decomposition Toolbox and explore the code by downloading the small dataset and instruction set from: http://web.media.mit.edu/~maov/classes/pra/index.html)

```
D=matricize(reshape(dtensor_train,75,6,6, 80*107),4);
```

Compute the basis vectors (eigenfaces), U. Display the top 10 eigenfaces:

```
eigimages=[];
for  i=1:10,
    im=reshape(U(:,i),80,107);
    mn=min(min(im));  mx=max(max(im));
    eigimages=[eigimages  (im-mn)/mx-mn];
end;
imshow(eigimages,[]);
```

Deliverables:

- Code for `NNtestM`
- Code for `PCA-NN`
- Display the top 10 eigenfaces.
- Explain why are we vectorizing each image $\mathbf{I}_n \rightarrow \mathbf{i}_n$ rather than leaving it as a square. Hint: Consider what type of covariances are comput when our training data is composed of $\mathbf{D}=[\mathbf{I}_1 \ldots \mathbf{I}_n \ldots \mathbf{I}_N]$ versus $\mathbf{D}=[\mathbf{i}_1 \ldots \mathbf{i}_n \ldots \mathbf{i}_N]$.
- Perform three experiments. The experiments differ in the choice of distance function
    - full-NN

- PCA-NN (225 basis vectors)
- PCA-NN (optimal basis)

(b) TensorFaces:

Using the `m_mode_svd` compute the Tensorfaces basis vectors, $\mathcal{T} = \mathcal{D} \times_1 \mathbf{U}_{\text{people}}^{\text{T}} \times_2 \mathbf{U}_{\text{views}}^{\text{T}} \times_3 \mathbf{U}_{\text{illums}}^{T}$. Display the main 3 faces of tensorfaces, ie. the illumination-viewpoint face of the Tensorfaces cube, the people-viewpoint face and the illumination-people face.

For example:

```
% The illumination-viewpoint face
basisimages=[];
for p=1;
    for il=1:5,
        rowim=[];
        for v=1:5
            im=reshape(T(p, vp, il,:),80,107);
            mn=min(min(im)); mx=max(max(im));
            rowim=[rowim (im-mn)/(mx-mn)];
        end
    basisimages=[basisimages; rowim];
end;
```

Implement a multilinear projection function, m_mode_proj, that takes an image, the learned TensorFaces basis vectors and projects the image into the person, viewpoint and illumination subspaces ($\mathbf{U}_{\text{people}}, \mathbf{U}_{\text{views}}, \mathbf{U}_{\text{illums}}$). The function should return the person coefficient vector representation, viewpoint representation, illumination coefficient vector representation, etc.

Use NNtestM to write a TensorFaces-NN function that classifies the person representation. Employ a dimensionaly reduced Tensorfaces (ie., 75 x 3 x 1 x 8560) to classify each image.

Deliverables:

- Display the main 3 faces of TensorFaces cube.
- Code for `m_mode_proj`
- Code for `TensorFaces-NN`
- Classify using Tensorfaces-NN
    - TensorFaces-NN (75 x 3 x 1 x 8560 basis vectors)
    - TensorFaces-NN (optimal number of basis vectors)
    - Generate a ROC curve – this will be time consuming