

## Project III: Eigendigits

### Optical Character Recognition using Dimensionality Reduction and a Principal Component Analysis (PCA) classifier

Due April 29, 2008 AT THE BEGINNING OF CLASS

#### Purpose

This project has students build a recognition system that performs dimensionality reduction and recognition of hand-written arabic digits using Principal Component Analysis (PCA) classifier (also known as an eigenspace classifier).

#### Data

You are provided a set of training images of written numeric digits that make up the MNIST database compiled by LeCun *et. al.* for use in the following publication. The publication analyzes a wide variety of approaches for automatic digit recognition within written documents.

**LeCun**, Y., L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE, Volume 86, Issue 11, 2278-2324, November 1998.

The database consists of 28x28 pixel images of handwritten numeric digits that have been scaled to be approximately the same size and moved so that they lie approximately in the center of each image. There are two databases of images: (1) 60000 training images in file *train-images-idx3-ubyte* (size=47040016 bytes) and (2) 10000 test images in file *t10k-images-idx3-ubyte* (size=7840016 bytes). Accompanying these data files are the correct actual classifications indicating the true numeric digit in each image for the training images *train-labels-idx1-ubyte* (size=60008) and the test images *t10k-labels-idx1-ubyte* (size=10008).

You are provided a MATLAB function `function [label,imagedata,irows,icols]=readUByteImageAndLabel(index, image_filename, label_filename)` that will read the image and associated digit label value at the specified `index` from the data files where the `image_filename` is a string indicating the name of the data file holding the images and `label_filename` is a string indicating the name of the data file holding the label values. For example,

```
[label,imagedata,irows,icols]=readUByteImageAndLabel(1,'train-images-idx3-ubyte','train-labels-idx1-ubyte');
```

is MATLAB code that will load the first training image and its associated label into the variables `imagedata` and `label` respectively. The variables (`irows,icols`) denote the number of image rows and columns respectively for the loaded image.

A second MATLAB function `I = getImage(index,imagedata,rows,columns)` that accesses the loaded image data and returns a matrix `I` with dimension (`irows,icols`) that is the image at the indicated index in the `imagedata`. For example,

```
I = getImage(1,imagedata,irows,icols);
```

is MATLAB code that will return the first image in the variable `imagedata`.

#### Methodology

In this task, we will compute the mean image and the principal components for a set of  $M$  training images that contain  $D$  pixels, i.e.,  $D = rows * columns = 28^2$  for our MNIST images. For this project each test image is a training vector. We convert the training image with dimensions  $28 \times 28$  to a feature vector,  $\mathbf{\Gamma}_i$ , by stacking each image column vertically in a single vector such that  $\mathbf{\Gamma}_i$  has dimension  $28^2 \times 1$ . The  $M$  total training images consist of  $N_i$  examples of each of the 10 hand-written digits. For example, if  $N_i = 6$  then  $M = 60$ , or in general  $M = 10N_i$ .

## Computing the Principal Components of the Input Data

1. Read the set training feature vectors,  $\Gamma = \{\Gamma_1, \Gamma_2, \dots, \Gamma_M\}$ , from file and the associated label for each vector  $\mathbf{L} = \{l_1, l_2, \dots, l_M\}$  that indicate the correct labels for the handwritten digits in the corresponding images.
2. Compute  $\Psi$ , the average image for all digits, from the values of each training vector,  $\Gamma_i$ . Both  $\Psi$  and  $\Gamma_i$  will have dimensions  $D \times 1$ .

$$\Psi = \frac{1}{M} \sum_{n=1}^M \Gamma_n$$

3. Compute the zero-mean version of each of the training images  $\Phi_i = \Gamma_i - \Psi$ . Put all of the zero-centered training images into a  $D \times M$  matrix  $\Phi = \{\Phi_1, \Phi_2, \dots, \Phi_M\}$ .
4. Compute  $\mathbf{C}$ , the covariance matrix, using the zero-mean versions of the training images. *Note that  $\mathbf{C}$  has dimension  $M \times M$  and characterizes the variation of the digit images around the average image. Note that we assume here that the number of training images is much less than the number of pixels in an image, i.e.,  $M \ll D$ .*

$$\mathbf{C} = \frac{1}{M} \Phi' \Phi$$

5. Compute the principal components of the scatter matrix,  $\mathbf{C}$ , and convert the resulting  $M$  eigenvectors,  $\mathbf{v}_i$ , of dimension  $M \times 1$  into  $M$  eigenvectors,  $\mathbf{u}_i$ , of dimension  $D \times 1$ .

$$\mathbf{u}_i = \frac{1}{(M\lambda_i)^{1/2}} \Phi \mathbf{v}_i$$

This will generate  $M$  eigenvectors,  $\mathbf{u}_i$ , which each have unit length,  $\|\mathbf{u}_i\|^2 = \mathbf{u}_i' \mathbf{u}_i = 1$ , and dimension  $D \times 1$ . These eigenvectors span the eigenspace of the hand-written digits in the training images. *Beware numerical instabilities which may generate invalid negative eigenvalues for  $\mathbf{C}$ . Hence, by convention, if  $\lambda_i \leq 0$  then assign  $\mathbf{u}_i = 0$ . Which assigns the eigenvectors associated with such eigenvalues to the zero vector.*

## Dimensionality reduction and projecting the training data into the subspace (the eigenspace)

1. Compute the mean eigenvalue,  $\bar{\lambda} = \frac{1}{M} \sum_{i=1}^M \lambda_i$ , which we will use as a threshold for determining the subset of eigenvalues and associated eigenvectors which point in directions of significant image variation.
2. Form an eigenspace from the collection of eigenvalues and the associated eigenvectors which exceed the mean eigenvalue  $\bar{\lambda}$ . This will consist of  $K$  eigenvectors ( $K < M$ ) which we can arrange as a set of columns in a matrix  $\mathbf{U}$  with dimensions  $D \times K$ .

$$\mathbf{U} = [ \mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_K ]$$

3. Compute the eigenspace representation for each of the training images by projecting each of the training images onto the  $D \times K$  subspace. This is equivalent to projecting each of the  $D \times 1$  zero-centered image vectors onto each of the  $K$  eigenvectors spanning the eigenspace. Each projection will generate a scalar result called a weight. We denote the weight computed by projecting training vector  $\Gamma_m$  onto eigenvector  $\mathbf{u}_k$  as  $\omega_{m,k}$ .

$$\omega_{m,k} = \Phi_m' \mathbf{u}_k = (\Gamma_m - \Psi)' \mathbf{u}_k$$

The entire vector of  $K$  weights for the  $m^{th}$  training vector may be computed as a row vector  $\Omega_m = [\omega_{m,1}, \omega_{m,2}, \dots, \omega_{m,K}]$  in one matrix multiply using the following equation:

$$\Omega_m = \Phi_m' \mathbf{U} = (\Gamma_m - \Psi)' \mathbf{U}$$

which generates a  $1 \times K$  dimensional vector of weights for training vector  $m$ . Finally, the whole collection of weights for all training vectors may also be computed in a single matrix multiply through the following equation:

$$\Omega = \Phi' \mathbf{U}$$

Where the  $m^{th}$  row of the resulting  $M \times K$  matrix is the set weights  $\Omega_m$  for training vector  $m$ .

Note that our resulting representation of all the training data consists of three components: (i) the projected training image weights and their associated training labels,  $\{\mathbf{\Omega}, \mathbf{L}\}$ , (ii) the vectors that define our eigenspace  $\mathbf{U}$ , and (iii) the mean of the training images  $\mathbf{\Psi}$ .  $\mathbf{\Omega}$  is a matrix having  $M$  rows, one for each training image, and  $K$  columns, one for each dimension in the eigenspace. Since  $K \ll D$ , this representation is a dimensionality reduction. This dimensionality reduction incurs some loss of information. The amount of lost information is directly related to the amount of energy in the eigenvalues which are discarded in step (1) above where the dimensionality reduction occurs. We can look at the loss of information by reconstructing a training image from the eigenspace representation as follows:

$$\tilde{\mathbf{\Gamma}}_m = \mathbf{\Omega}_m \mathbf{U}^t + \mathbf{\Psi}$$

The loss of information for the  $m^{\text{th}}$  training vector is provided by the following equation:

$$e_m(\mathbf{\Gamma}_m, \tilde{\mathbf{\Gamma}}_m) = \|\mathbf{\Gamma}_m - \tilde{\mathbf{\Gamma}}_m\|$$

If wish to be able to perfectly reconstruct all the training data images, the full eigenspace would have dimension  $M \times D$ , i.e.,  $M$  vectors each having  $D$ -dimensions which jointly space the *entire* linear vector space defined by the training image data and corresponds to simply a change in coordinates for the space of digit images. Where the maximum dimensionality for the space is  $D \times D$  which occurs when  $M \geq D$ . However, such large values for  $M$  usually correspond to circumstances that are typically intractable in terms of computational resources having both prohibitive computational complexity  $O(D^3)$  and memory requirements.

## Classification

Classification proceeds in a straightforward manner by projecting the zero-centered test vector into the eigenspace and performing a  $k$ -nearest neighbor search within the eigenspace where  $k =$ .

1. Load a test image  $\mathbf{\Gamma}_z$ .
2. Convert this into a zero-mean image  $\mathbf{\Phi}_z = \mathbf{\Gamma}_z - \mathbf{\Psi}$ .
3. Project the test image into the eigenspace.

$$\mathbf{\Omega}_z = \mathbf{\Phi}_z^t \mathbf{U} = (\mathbf{\Gamma}_z - \mathbf{\Psi})^t \mathbf{U}$$

This will generate a  $1 \times K$  vector of weights,  $\mathbf{\Omega}_z = [\omega_{z,1}, \omega_{z,2}, \dots, \omega_{z,K}]$ , which is the dot product between each eigenvector,  $\mathbf{u}_i$ , and the zero-centered test image,  $\mathbf{\Phi}_z$ .

4. Find the eigenspace training vector closest to the projection of the test vector into the eigenspace  $\mathbf{\Omega}_z$ :

$$\hat{\mathbf{\Omega}} = \min_{1 \leq m \leq M} \|\mathbf{\Omega}_m - \mathbf{\Omega}_z\|$$

5. We then take as the classification for  $\mathbf{\Omega}_z$  the class label,  $\hat{l}$ , associated with the minimum distance training vector  $\hat{\mathbf{\Omega}}$ .

## Structure of your Code

You will write your classifier using MATLAB. The main project code will reside in a MATLAB script file: *project3.m*. This file will make use of the utility functions which you will write to complete the eigenspace recognition. The first aspect of the recognition problem is to compute the eigenvectors for the complete eigenspace. You will do this by writing a MATLAB function with the following function prototype:

```
function [Ufull,lambda,psi] = computeFullEigenSpace(gammaMatrix)
```

whose input is the complete set of test images in a  $D \times M$  matrix, **gammaMatrix**, and returns the  $D \times M$  matrix of eigenvectors of the full eigenspace in a orthogonal matrix **Ufull**, the associated eigenvalues in a  $M \times 1$  vector **lambda**, and the  $D \times 1$  mean image vector **psi**.

You will then reduce the dimensionality of the full eigenspace by calling a MATLAB function with the following function prototype:

```
function [omegaMatrix,Ureduced] = reduceEigenSpace(Ufull,lambda,phiMatrix)
```

whose input is the the  $D \times M$  matrix of zero-centered training images **phiMatrix**, the  $D \times M$  matrix of eigenvectors for the full eigenspace **Ufull**, and the associated eigenvalues in a  $M \times 1$  vector **lambda**. Values returned from this function include the  $M \times K$  matrix of reduced eigenspace training vectors **omegaMatrix**, and the  $D \times K$  matrix of eigenvectors associated with the  $K$ -dimensional eigenspace **Ureduced**.

We may then use the computed eigenspace for recognition. To do so, we must write a MATLAB function with the following function prototype:

```
function estimatedLabel = eigenspaceClassify(I_test, omegaMatrix, L, Ureduced, psi)
```

whose input is a test image, **I\_test**, and the parameters of the eigenspace which consist of the eigenspace training vectors **omegaMatrix**, their associated class labels **L**, the eigenspace vectors **Ureduced**, and the mean training image **psi**. A single value is returned which is the estimated label value for the test image **estimatedLabel**.

Within the *eigenspaceClassify()* function you will need to solve the nearest neighbor problem again (step (4) in *Classification*). Note that you can (and should) re-use your k-NN classifier functions from project 2. However, since the features are not organized by class, you will need to modify the function prototypes to those below:

```
function estimatedLabel = kNNClassify(I_test,k,featureValues,featureLabels)
```

where **I\_test** is the test image  $\Gamma_z$ , the parameter **k** denotes the number of nearest neighbors for the classifier, **featureValues** is an  $M \times K$  matrix of the eigenspace training vectors (one vector per matrix row), and **featureLabels** is an  $M \times 1$  vector of labels such that element  $m$  of the **featureLabels** vector indicates the class label for the training vector at row  $m$  of the **featureLabels** matrix. The function returns the k-NN classification result, i.e., the estimated label value (digit value) for the input test image.

```
function [xNN,xlabels] = findkNN(x_test,k,featureValues,featureLabels)
```

where **x\_test** is the vector  $\Omega_z$ , the parameter **k** denotes the number of nearest neighbors for the classifier, **featureValues** is an  $M \times K$  matrix of the eigenspace training vectors (one vector per matrix row), and **featureLabels** is an  $M \times 1$  vector of labels such that element  $m$  of the **featureLabels** vector indicates the class label for the training vector at row  $m$  of the **featureLabels** matrix. The returned value **xNN** is a  $k \times K$  matrix where each row has dimension  $K$  and corresponds to one of the training vectors  $\Omega_m$  that is among the  $k$  nearest neighbors of **x\_test**. The second returned value **xlabels** indicates the label for each feature vector in the matrix **xNN**. Note you will only need to call this function for  $k = 1$ .

## To turn in

Turn in the source code to your program which consists of the main program in *project3.m* and the four functions *computeFullEigenSpace()*, *reduceEigenSpace()*, *eigenspaceClassify()*, *kNNClassify()*, and *findkNN()*.

1. Train your system using  $N_t = 5$  examples for each digit from the training data. Using your “*Eigendigit*” classifier, classify the first  $T = 3000$  images from the test data. Output the classifier **error rates** for each of the digits and the **overall error rate**.
2. Same as in (1), let  $N_t = 10$ .
3. Same as in (1), let  $N_t = 20$ .
4. Same as in (1), let  $N_t = 30$ .

5. Write a brief discussion analyzing your results for classifying digits.

Complete your code by commenting your main program and each of the functions such that when you type *help* `<functionname>` into the MATLAB prompt, a description of your corresponding function with parameters and return values is printed to the screen. Clarify your presented results by annotating your plots using the MATLAB *xlabel()* and *ylabel()* commands.

## Extra Credit

1. Change the nearest neighbor search parameter such that  $k > 1$  to see if you can improve your recognition performance on the digits. Repeat steps (1-4) from the section *To turn in* above with your new value for  $k$  and write a paragraph comparing your results to those for  $k = 1$ .
2. Apply your eigenspace recognition program to a collection of face images and perform steps (1-5) from the section *To turn in* above. The face image data is available upon request.