Face Recognition

Aryan Mokhtari, Santiago Paternain, and Alejandro Ribeiro

April 4, 2016

The goal of this lab is to implement face recognition using Principal Component Analysis (PCA). One of the most important applications of the PCA is mapping the dataset into a new space with smaller dimension such that the error of reconstruction is minimized. Map reduction is specially useful for the datasets where the dimension of sample points are large. Consider the dataset $\mathcal{D} := \{\mathbf{x}_1, \dots, \mathbf{x}_M\}$ where each sample point $\mathbf{x}_i$ has dimension N. We map the signals into a space of dimension k where $k << N$. To do this first we recap the idea of PCA in the following section.

1 Principal Component Analysis

In PCA decomposition we define a new Hermitian matrix based on the eigenvectors of the covariance matrix of a dataset. Before defining the covariance matrix we need to the define the mean signal.

Definition 1 *Let $\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_M$ be M different vectorized points in a dataset, then we can define the mean signal as*

$$\boldsymbol{\mu} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{x}_m. \tag{1}$$

We next define the empirical covariance matrix.

Definition 2 (Covariance matrix) *Let $\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_M$ be M different points in a dataset, then we can define the covariance matrix as*

$$\Sigma = \frac{1}{M} \sum_{m=1}^{M} (\mathbf{x}_m - \boldsymbol{\mu})(\mathbf{x}_m - \boldsymbol{\mu})^T. \tag{2}$$

Let $\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_{N-1}$ be the eigenvectors of the covariance matrix that they correspond to the eigenvalues $\lambda_0, \ldots, \lambda_{N-1}$, respectively. Notice that the eigenvalues are ordered as $\lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{N-1}$. Then, as in the Discrete Fourier transform define the Hermitian matrix

$$\mathbf{T} = \begin{bmatrix} \mathbf{v}_0 & \mathbf{v}_1 & \cdots & \mathbf{v}_{N-1} \end{bmatrix} \in \mathbb{R}^{N \times N} \tag{3}$$

where the ith column is the ith eigenvector of the covariance matrix. Then the PCA decomposition can be written as a product between the matrix $\mathbf{T}^H$ and the difference between the signal and the mean signal i.e

$$\mathbf{X}_i^{PCA} = \mathbf{T}^H (\mathbf{x}_i - \boldsymbol{\mu}) \quad \text{for } i = 1, \ldots, M. \tag{4}$$

where $\mathbf{x}$ and $\boldsymbol{\mu}$ are $N \times 1$ vectors. Just as with the DFT and the DCT we can define the inverse operation to PCA decomposition, which gives us the signal based on $\mathbf{X}_{PCA}$. For this we need to compute $\mathbf{T}$. Then the inverse transformation is given by

$$\tilde{\mathbf{x}}_i = \mathbf{T}\mathbf{X}_i^{PCA} + \boldsymbol{\mu} \tag{5}$$

As we discussed before, when we use all the eigenvectors $\mathbf{v}_0, \ldots, \mathbf{v}_{N-1}$ for making the PCA transform matrix $\mathbf{T}$, the reconstructed vectors are equal to the original signals, i.e., $\mathbf{x}_i = \tilde{\mathbf{x}}_i$.

2 Dimension reduction

As we observed before, in DFT and DCT for compressing the signal we keep the coefficients with the largest value. In PCA we use a different mechanism for compressing the signal. We keep the coefficients that corresponds to the largest eigenvectors. This scheme can be implemented by redefining the PCA transform matrix as

$$\tilde{\mathbf{T}} = \begin{bmatrix} \mathbf{v}_0 & \mathbf{v}_1 & \cdots & \mathbf{v}_{k-1} \end{bmatrix} \in \mathbb{R}^{N \times k} \tag{6}$$

As you can see we only use the eigenvectors that correspond to the k largest eigenvalues for creating the transform matrix $\tilde{\mathbf{T}}$. Therefore, the PCA-ed signals dimension is k which is smaller than N. The PCA-ed signals are given by

$$\tilde{\mathbf{X}}_i^{PCA} = \tilde{\mathbf{T}}^H (\mathbf{x}_i - \boldsymbol{\mu}) \quad \text{for } i = 1, \ldots, M. \tag{7}$$

To reconstruct the signal we use the inverse transformation which is given by

$$\tilde{\mathbf{x}}_i = \tilde{\mathbf{T}}\tilde{\mathbf{X}}_i^{PCA} + \boldsymbol{\mu} \tag{8}$$

Observe that as with the DFT and the DCT the reconstruction error is not zero for PCA when we compress the signal. The average reconstruction error can be computed as

$$\frac{1}{M}\sum_{i=1}^{M}\|\mathbf{x}_i-\tilde{\mathbf{x}}_i\|^2. \tag{9}$$

3 Face Recognition

In face recognition we have access to a set of data called *Training set* which we know their labels. The labels can be binary values, integer numbers, etc. In the face recognition application that we consider in this lab, the label of each image is the person who is in the image. The goal of face recognition is assigning labels to a set of signals (images) called *Test set* which they are not classified. Different techniques can be used for classification, but in this lab we use the nearest neighbor method.

Consider a training set $\mathcal{D}_{training} := \{\mathbf{x}_1,\dots,\mathbf{x}_M\}$ where their labels are $\{y_1,\dots,y_M\}$. Further, consider $\mathcal{D}_{test} := \{\hat{\mathbf{x}}_1,\dots,\hat{\mathbf{x}}_p\}$ as a test set that the labels of sample points are not given. To classify the label of a sample point $\hat{\mathbf{x}}_i$, we compute the distance of $\hat{\mathbf{x}}_i$ from all the points in the training set and find the one that has the minimum distance,

$$\mathbf{x}_i^* = \underset{\mathbf{x}\in\mathcal{D}_{training}}{\operatorname{argmin}} \|\hat{\mathbf{x}}_i-\mathbf{x}\| \tag{10}$$

Observe that $\mathbf{x}_i^*$ is the closet training point to the test point $\hat{\mathbf{x}}_i$, therefore, we assign the label of $\mathbf{x}_i^*$ (which is y_i) to the test point $\hat{\mathbf{x}}_i$.

Notice that each pixel of images contains noise. If we map the images into the space of principal components, we can improve the accuracy of classification. To do this, we map all the training points and test points into the space of the training points principal components and redo the nearest neighbor algorithm for classifying the points in the test set. This works because mapping into the space of principle components reduces the energy defined by noise/variability (see face recognition slide), increasing classification accuracy. Define $\mathbf{\Sigma}_{training}$ as the covariance matrix of the training set,

$$\mathbf{\Sigma}_{training} = \frac{1}{M}\sum_{m=1}^{M}(\mathbf{x}_m-\boldsymbol{\mu}_{training})(\mathbf{x}_m-\boldsymbol{\mu}_{training})^T, \tag{11}$$

where $\boldsymbol{\mu}_{training}$ is the average image of the training set. Consider the k eigenvectors $\mathbf{v}_0,\dots,\mathbf{v}_{k-1}$ of the training covariance matrix $\mathbf{\Sigma}_{training}$ to

define the PCA transformation as

$$\tilde{\mathbf{T}} = \begin{bmatrix} \mathbf{v}_0 & \mathbf{v}_1 & \cdots & \mathbf{v}_{k-1} \end{bmatrix} \in \mathbb{R}^{N \times k} \tag{12}$$

Remember that k is the chosen number of eigenvectors we want to use (default is N). We map all the training points $\mathbf{x}_i \in \mathcal{D}_{training}$ into the space of k principal components as

$$\tilde{\mathbf{X}}_i^{PCA} = \tilde{\mathbf{T}}^H \left(\mathbf{x}_i - \boldsymbol{\mu}_{training} \right) \qquad \text{for } \mathbf{x}_i \in \mathcal{D}_{training}. \tag{13}$$

Mapped training images $\tilde{\mathbf{X}}_i^{PCA}$ form a PCA-ed training set $\mathcal{D}_{training}^{PCA}$. We do the same process for the images in the test point $\mathcal{D}_{test}$ as

$$\hat{\mathbf{X}}_i^{PCA} = \tilde{\mathbf{T}}^H \left(\hat{\mathbf{x}}_i - \boldsymbol{\mu}_{training} \right) \qquad \text{for } \hat{\mathbf{x}}_i \in \mathcal{D}_{test}. \tag{14}$$

Likewise, the mapped images $\hat{\mathbf{X}}_i^{PCA}$ create the PCA-ed test set $\mathcal{D}_{test}^{PCA}$. Now we implement the nearest neighbor algorithm as in (16) for the PCA-ed training and test sets. I.e.,

$$\mathbf{X}_i^* = \underset{\mathbf{X} \in \mathcal{D}_{training}^{PCA}}{\operatorname{argmin}} \| \hat{\mathbf{X}}_i^{PCA} - \mathbf{X} \| \qquad \text{for all} \quad \hat{\mathbf{X}}_i^{PCA} \in \mathcal{D}_{test}^{PCA} \tag{15}$$

Notice that $\mathbf{X}_i^*$ is the closest neighbor of the PCA-ed test point $\hat{\mathbf{X}}_i^{PCA}$. Therefore, we assign the label y_i of image $\mathbf{x}_i^*$ in the training set to the image $\hat{\mathbf{x}}_i$.

4 Creating training and test sets

For this lab we use the dataset provided by AT&T Laboratories Cambridge. The images of the dataset are shown in Fig 1. As it is shown the dataset contains 10 different images of 40 people. The images are in ".pgm" format and the size of each image is 112×92, 8-bit grey levels. The images are organized in 40 directories (one for each person) named as "sX", where X indicates the person number (between 1 and 40). In each directory there are 10 different images of the selected person named as "Y.pgm", where Y indicates which image for the specific person (between 1 and 10). We pick some of the images to create a training set and use the rest as a test set for face recognition.

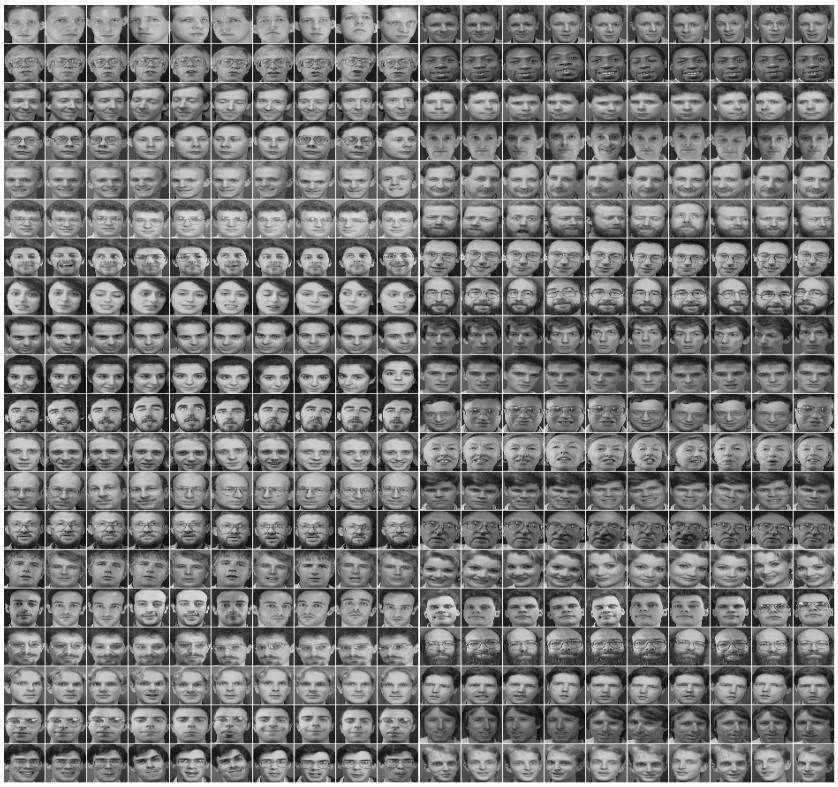

Figure 1. Images of AT&T Laboratories Cambridge dataset.

4.1 Generating training and test sets.

Consider the vector ″num_sample_training″ with size 1×40 that contains the number of images required for creating the training set from each directory. E.g., the *first* component of vector ″num_sample_training″ shows the number of required images from directory s1 for creating the training set. The rest of the sample points in the dataset are considered as the test points. Write a Matlab function that takes as input vector ″num_sample_training″ and returns as output a matrix called ″faces_training″ with dimension ″$112 \times 92 \times$ size of the training set″ and a matrix called ″faces_test″ with dimension ″$112 \times 92 \times$ size of the test set″. Notice that the function should open each directory and store a specified number of images from each directory in the matrix ″faces_training″ and

the rest in the matrix "faces_test". If for example, "num_sample_training" was a 1×40 vector of all 9s, then "faces_training" would be a $112 \times 92 \times 360$ size matrix and "faces_test" would be a $112 \times 92 \times 40$ size matrix.

4.2 Generating training and test sets. Use the function in Part 4.1 to store all the first 9 images of 40 different categories in matrix "faces_training" and the rest in "faces_test". Reminder that the size of generated matrices should be $112 \times 92 \times 360$ and $112 \times 92 \times 40$, respectively.

4.3 Generating training and test matrices. Note that the size of images is 112×92, therefore, each image can be represented as a vector of length 10304. By concatenating sample vectors of size 10304 in a matrix we can generate a training matrix of size "10304×360" and a test matrix of size "10304×40" . Write a Matlab function that takes as input matrices "faces_training" and "faces_test" and returns training matrix X_{train} where each column contains one image of the training set and test matrix X_{test} where each column contains one image of the test set. If you vectorized each image into a row, remember to change them into column vectors before concatenating.

4.4 Generating training matrix. Use the function in Part 4.3 and the matrices "faces_training" and "faces_test" generated in Part 4.2 to create the training and test matrices X_{train} and X_{test}.

5 PCA on the training and test sets

In this section we use Principal Component Analysis to reduce the size of features of samples in the training matrix. We also map the images in the test set into the space of training set principal components.

5.1 PCA function. Write a Matlab function that given the number of principal components k and the training matrix X_{train}, returns the PCA-ed training matrix X_{train}^{PCA}, the k eigenvectors $\mathbf{V}_k = [\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_{k-1}]$ corresponding to the k largest eigenvalues, and the mean vector $\boldsymbol{\mu}$. Notice that the dimension of eigenfaces matrix $\mathbf{V}_k$ should be $10304 \times k$. Also remember that the eigenvectors for $\mathbf{V}_k$ are derived from the covariance matrix of the training matrix $\boldsymbol{\Sigma}_{training}$.

5.2 PCA for different choices of k. Use the training matrix X_{train} generated in Part 4.4 as the input of function in Part 5.1 to generate the PCA-ed training set. For the number of principal components consider $k = 1, 5, 10, 20$. Notice that since there are 4 different choices for the number of principal components we expect 4 different PCA-ed training matrices X_{train}^{PCA} with sizes 1×360, 5×360, 10×360, and 20×360 and their corresponding eigenfaces matrices $\mathbf{V}_k$ of sizes 10304×1, 10304×5, 10304×10, and 10304×20, respectively.

5.3 PCAed test point. Write a Matlab function that given the test matrix X_{test}, mean value of features $\boldsymbol{\mu}$, and the eigenfaces matrix $\mathbf{V}_k$, returns the PCA-ed test set as X_{test}^{PCA}.

5.4 More PCAed test point. Use the function in Part 5.3 and outputs of Part 5.2 to apply principal component analysis on the tests point for different values of principal components $k = 1, 5, 10, 20$. The outcome of this section should be 4 different PCAed test matrices X_{test}^{PCA} with sizes 1×40, 5×40, 10×40, and 20×40.

6 Nearest neighbor classification

Consider that we have an image form the test set. Our goal is to find the image in the training set which is the most similar to the test image. To classify the closest image we use the nearest neighbor algorithm.

6.1 Nearest Neighbor function. write a MATLAB function that given the PCA-ed training set $X_{training}^{PCA}$ and the PCA-ed test set X_{test}^{PCA} returns the sample of PCA-ed training set $\mathbf{X}$ that has the minimum distance to the PCA-ed test point. In other words, for every column i of X_{test}^{PCA}, find the column j of $X_{training}^{PCA}$ that minimizes distance between the two, then set column i of $\mathbf{X}$ as column j of $X_{training}^{PCA}$. This is also viewed as

$$\mathbf{X}_i = \underset{X_j^{PCA} \in X_{training}^{PCA}}{\operatorname{argmin}} \|X_i^{PCA} - X_j^{PCA}\| \tag{16}$$

6.2 More Nearest Neighbor. Use the function in Part 6.1 to find the nearest neighbors of PCA-ed test points in the test matrix X_{test}^{PCA} given the PCA-ed training points in the matrix X_{train}^{PCA}. Display some of the images in the test set and their nearest neighbors for different choices of k.