Face Recognition

Aryan Mokhtari, Santiago Paternain, and Alejandro Ribeiro

April 1, 2021

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 a dataset into a new space with smaller dimension in a way that minimizes the reconstruction error. Dimensionality reduction is specially useful for datasets in which the dimension of each sample points is large. So if we consider a dataset $\mathcal{D} := \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$, where each sample point $\mathbf{x}_i$ has dimension N, we want to map these signals into a space of dimension k with $k \ll N$. To do so, let us first recap the idea of PCA.

1 Principal Component Analysis

In PCA, we define a new unitary matrix based on the eigenvectors of the covariance matrix of a dataset. Before discussing covariance matrices, though, we need to the define the signal average.

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

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

Notice that this is simply the definition of the sample mean. We next define the empirical covariance matrix.

Definition 2 (Covariance matrix) *Let $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_M$ be M different points in a dataset. Then, their empirical covariance matrix is*

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

for $\bar{\mathbf{x}}$ *as in* (1).

We can define the PCA transformation by using eigenvectors of the empirical covariance matrix (2). Indeed, let $\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_{N-1}$ be the eigenvectors of (2) that correspond to the eigenvalues $\lambda_0, \ldots, \lambda_{N-1}$ respectively. We assume that the eigenvalues are ordered, i.e., $\lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{N-1}$. Then, as for the Discrete Fourier transform, define the unitary 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 i-th column is the i-th eigenvector of the empirical covariance matrix. The PCA transform is then written as a product between the matrix $\mathbf{T}^H$ and the centered signal, i.e.,

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

where $\mathbf{x}_i$ and $\bar{\mathbf{x}}$ are $N \times 1$ vectors. Just as with the DFT and the DCT, we can also define the inverse operation to PCA transform which gives us the signal $\mathbf{x}_i$ based on the coefficients $\mathbf{x}_i^{\text{PCA}}$. Since the transformation in (3) is unitary, the inverse transformation is given by $\mathbf{T}$, i.e.,

$$\tilde{\mathbf{x}}_i = \mathbf{T}\mathbf{x}_i^{PCA} + \bar{\mathbf{x}}. \tag{5}$$

Using the fact that $\mathbf{T}$ is unitary ($\mathbf{T}\mathbf{T}^H = \mathbf{I}$), you should be able to quickly prove that $\mathbf{x}_i = \tilde{\mathbf{x}}_i$, i.e., that (3) and (5) indeed "undo" each other.

2 Dimension reduction

When using the DFT and the DCT to compress signals, we kept the coefficients with the largest magnitude and supposed all others are zeros. In PCA, we do use a similar idea, but keep the coefficients that correspond to the eigenvectors with largest eigenvalues. In other words, we do not look at the values of $\mathbf{x}_i^{\text{PCA}}$ and instead rely on the eigenvalues $\lambda_0, \ldots, \lambda_{N-1}$. This scheme can be implemented by defining another PCA transform matrix:

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

As you can see we use only the eigenvectors that correspond to the k largest eigenvalues to create $\tilde{\mathbf{T}}_k$. The PCA coefficients of the signals are now given by

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

Notice that $\mathbf{x}_i^{\text{PCA}_k}$ is a vector of dimension $k < N$. Also, observe that (7) is equivalent to performing a full PCA as in (4) and then discarding all but the first k elements of $\mathbf{x}_i^{\text{PCA}}$. However, (7) is a more efficient way of doing it.

The inverse transformation of (7) is given by

$$\tilde{\mathbf{x}}_i^{\text{PCA}_k} = \tilde{\mathbf{T}}_k \mathbf{x}_i^{\text{PCA}_k} + \bar{\mathbf{x}}. \tag{8}$$

Naturally, as with the DFT and the DCT, the reconstruction error is not zero when we compress the signal. In other words, $\tilde{\mathbf{x}}_i^{\text{PCA}_k} \neq \mathbf{x}_i$.

3 Face Recognition

In face recognition, we have access to a dataset called the *training set* which contains pictures of faces labeled with people's names or IDs. In this lab, the label of each image is an integer number that identifies the person in the image. The goal of face recognition is assigning labels (integers) to a set of signals (images) for which we do not have labels. We call this set the *test set*. Different techniques can be used to perform this classification task and in this lab we will use the nearest neighbor method.

Consider a training set $\mathcal{D}_{\text{training}} := \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ where their labels are $\{y_1, \ldots, y_M\}$. Further, consider $\mathcal{D}_{\text{test}} := \{\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_P\}$ to be a test set for which the labels are not given. To determine the label of a sample point $\hat{\mathbf{x}}_i$, we use the nearest neighbor method. However, instead of using it directly on the images, we will apply the method to their PCA transforms. Hence, let $\mathbf{\Sigma}_{\text{training}}$ be the covariance matrix of the training set

$$\mathbf{\Sigma}_{\text{training}} = \frac{1}{M} \sum_{m=1}^{M} (\mathbf{x}_m - \bar{\mathbf{x}}_{\text{training}})(\mathbf{x}_m - \bar{\mathbf{x}}_{\text{training}})^T, \tag{9}$$

where $\bar{\mathbf{x}}_{\text{training}}$ is the average image of the training set. Consider the eigenvectors $\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}$ relative to the k largest eigenvalues of the training covariance matrix $\mathbf{\Sigma}_{\text{training}}$ and define the PCA transformation

$$\tilde{\mathbf{T}}_k = \begin{bmatrix} \mathbf{v}_0 & \mathbf{v}_1 & \cdots & \mathbf{v}_{k-1} \end{bmatrix} \in \mathbb{R}^{N \times k}. \tag{10}$$

Recall that k represents the "size" of the transform, i.e., the number of PCA coefficients we keep (i.e., $k \leq N$).

Using (10), we can project the training points $\mathbf{x}_i \in \mathcal{D}_{\text{training}}$ into the space of the first k principal components using

$$\mathbf{x}_i^{\text{PCA}_k} = \tilde{\mathbf{T}}_k^H \left(\mathbf{x}_i - \bar{\mathbf{x}}_{\text{training}} \right), \quad \text{for } \mathbf{x}_i \in \mathcal{D}_{training}. \tag{11}$$

The coefficients of each training images $\mathbf{x}_i^{\text{PCA}_k}$ form the transformed training set $\mathcal{D}_{\text{training}}^{\text{PCA}_k}$. We repeat this process for the images in the test set $\mathcal{D}_{\text{test}}$ to obtain

$$\hat{\mathbf{x}}_i^{\text{PCA}_k} = \tilde{\mathbf{T}}_k^H \left(\hat{\mathbf{x}}_i - \bar{\mathbf{x}}_{\text{training}} \right), \qquad \text{for } \hat{\mathbf{x}}_i \in \mathcal{D}_{\text{test}}. \tag{12}$$

Once again, the transformed images $\hat{\mathbf{x}}_i^{\text{PCA}_k}$ are collected to create the transformed test set $\mathcal{D}_{\text{test}}^{\text{PCA}^k}$.

We can now run the nearest neighbor algorithm on the transformed training and test sets. Explicitly,

$$\mathbf{x}_j^\star = \underset{\mathbf{x}_i^{\text{PCA}_k} \in \mathcal{D}_{\text{training}}^{\text{PCA}_k}}{\operatorname{argmin}} \left\| \hat{\mathbf{x}}_j^{\text{PCA}_k} - \mathbf{x}_i^{\text{PCA}_k} \right\|^2, \quad \hat{\mathbf{x}}_j^{\text{PCA}_k} \in \mathcal{D}_{\text{test}}^{\text{PCA}_k} \tag{13}$$

Note that $\mathbf{x}_j^\star$ is the closest neighbor in the training set of the transformed test point $\hat{\mathbf{x}}_j^{\text{PCA}_k}$. We can therefore simply assign the label y_j of image $\mathbf{x}_j^\star$ from the training set to the test image $\hat{\mathbf{x}}_j$.

4 Creating training and test sets

For this lab we use the dataset provided by the AT&T Laboratories Cambridge. The images of this dataset are shown in Fig. 1. This dataset is comprised of 10 images for each of 40 different people. The images are 112×92 pixel with 8-bit grey levels provided in `.pgm` format (Portable GrayMap). They are organized in 40 folders (one for each person) named `sX`, where X is a number between 1 and 40 that identifies each individual. Inside each folder, you will find 10 different images of the selected person named as `Y.pgm`, where Y is a number between 1 and 10. We will assign some of the images to the training set and use the rest as the test set in our face recognition experiments.

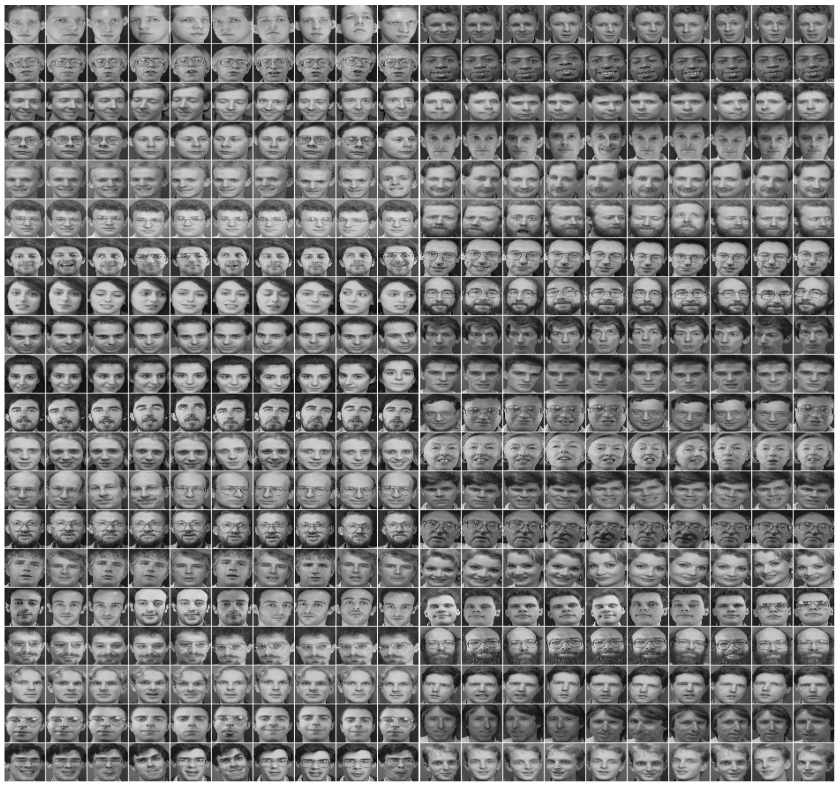

Figure 1. Images of AT&T Laboratories Cambridge dataset.

4.1 Generating training and test sets. Write a class that creates two matrices, $\mathbf{X}_{\text{train}}$ and $\mathbf{X}_{\text{test}}$, whose columns represent the data points for the training and test sets respectively. In other words, each column of $\mathbf{X}_{\text{train}}$ and $\mathbf{X}_{\text{test}}$ represents one image from the dataset. For each person, assign the first nine pictures to the training set and the last picture to the test set. In other words, your class must open each folder, load the first nine images, vectorize them, and save them into $\mathbf{X}_{\text{train}}$; then, open the last image, vectorize it, and save it into $\mathbf{X}_{\text{test}}$. Since the images are 112×92 pixels, their vector representation is 10304×1. Hence, $\mathbf{X}_{\text{train}}$ is a 10304×360 and $\mathbf{X}_{\text{test}}$ is 10304×40. (*Hint: See Python built in function* `matplotlib.pyplot.imread`)

5 PCA on the training and test sets

In this section we use PCA to reduce the dimensionality of the training samples (i.e., $\mathbf{X}_{\text{train}}$). We also map the samples in the test set into the principal components' space of training set.

5.1 PCA transform of training points. Write a Python class that given the number of principal components k and the training matrix $\mathbf{X}_{\text{train}}$, returns (i) the $k \times 360$ PCA transformed training matrix $\mathbf{X}_{\text{train}}^{\text{PCA}_k}$; (ii) the k-PCA transform matrix $\tilde{\mathbf{T}}_k$ from (7), which contains the eigenvectors corresponding to k largest eigenvalues of the empirical covariance matrix; and (iii) the mean vector $\bar{\mathbf{x}}$. Notice that the dimension of "eigenfaces matrix" $\tilde{\mathbf{T}}_k$ should be $10304 \times k$. Also recall that the eigenvectors in $\mathbf{X}_{\text{train}}^{\text{PCA}_k}$ are derived from the covariance matrix of the training set $\mathbf{\Sigma}_{\text{training}}$. Run this function with the training matrix $\mathbf{X}_{\text{train}}$ from Part 4.1 for $k = 1, 5, 10, 20$. Notice that since there are four different choices for the number of principal components, we expect four different transformed training matrices $\mathbf{X}_{\text{train}}^{\text{PCA}_k}$ and four different eigenfaces matrices $\tilde{\mathbf{T}}_k$.

5.2 PCA transform of test point. Write a Python class that given the test matrix $\mathbf{X}_{\text{test}}$, the mean vector $\bar{\mathbf{x}}$, and the eigenfaces matrix $\tilde{\mathbf{T}}_k$, returns the PCA transform of the test set $\mathbf{X}_{\text{test}}^{\text{PCA}_k}$. Use this function to obtain the PCA coefficients for $\mathbf{X}_{\text{test}}$ from Part 4.1 for $k = 1, 5, 10, 20$. The outcome of this section should be four different transformed test matrices $\mathbf{X}_{\text{test}}^{\text{PCA}_k}$ of sizes 1×40, 5×40, 10×40, and 20×40.

6 Nearest neighbor classification

Consider that we have an image from 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 class. Define a Python class that takes the transformed training set $\mathbf{X}_{\text{train}}^{\text{PCA}_k}$ and the transformed test set $\mathbf{X}_{\text{test}}^{\text{PCA}_k}$ and returns for each test sample, the index of the training sample closest to it. In other words, for each column of $\mathbf{X}_{\text{test}}^{\text{PCA}_k}$, it finds the index of the column in $\mathbf{X}_{\text{train}}^{\text{PCA}_k}$ that is closest to it. Formally, let $[\mathbf{M}]_i$ denote the i-th column of the matrix $\mathbf{M}$. Then,

$$y_j^\star = \underset{i=1,\ldots,N}{\operatorname{argmin}} \left\| \left[\mathbf{X}_{\text{test}}^{\text{PCA}_k}\right]_j - \left[\mathbf{X}_{\text{train}}^{\text{PCA}_k}\right]_i \right\|^2, \quad j = 1, \ldots, P, \tag{14}$$

Display some of the images in the test set and their nearest neighbors for different choices of k.