Matlab Project

Due: Jan. 14, 2019

Please show all work in order to get full credit.

I. THEORETICAL BACKGROUND

Linear algebra is an extremely useful tool in many applications. One such applications as you have learned in class is facial identification, where you will decide (detect) which face in your training database most resembles the image (which is assumed to be a face) in question. In this project, you are asked to program different parts of the "eigenface" facial identification and show your results quantitatively by plotting the so-called miss detection rate vs. signal-to-noise ratio (SNR). Note that this problem is different from facial recognition where you need to decide whether a certain image is a face at all.

In mathematical terms, we wish to find the principal components of the distribution of faces, that is eigenvectors of covariance matrix of set of images. Intuitively speaking, the covariance measures the similarity of one face to another face. In this project, each image is treated as one high-dimensional point. Eigenvectors of covariance matrix can be thought as a set of features that represent variation between different images. To be more specific, we treat each $N \times N$ pixels image as a point in N^2 dimensional space. We have M of those points and usually $M \ll N^2$ (e.g. $N^2 = 10,000, M = 4000$). Such huge dimension of data causes computational problem (so-called curse of dimensionality), even though number of data points is not that large. Hence, we can use idea of low rank approximation to describe our data.

For system doing facial identification you have to go through two stages: training and testing. In training stage your system 'learns' how to identify faces, based on a database of different face images (i.e. faces of same person in different poses, and also faces of different people) which we have provided. This data is called *training data*. After you have taught your system about what a face looks like, the next step is to test your system using *testing data*. For example, testing data can be distorted version of the training data, i.e. you add some noise to your face and see if your system is still able to identify this image containing a distorted version of a person's face is still that person. The generation of the testing data has already been implemented for you.

In the training phase, let $\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_M \in \mathbb{R}^{N^2}$ denote the image vectors (i.e. $\mathbf{f}_i$ is a image point in a N^2 dimensional space) and $\boldsymbol{\gamma} \in \mathbb{R}^{N^2}$ will be their mean. For technical reason which we will not explain here, we want our data to have zero-mean, hence, we compute a zero-mean face point as

$$\mathbf{g}_i = \mathbf{f}_i - \boldsymbol{\gamma}, \quad \forall 1, ..., M.$$

We concatenate $\mathbf{g}_i, \forall i \in 1, ..., M$ into matrix $\mathbf{G}$, i.e $\mathbf{G} = [\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_M]$. Next step is to calculate the corresponding correlation (covariance) matrix

$$\mathbf{C} \triangleq \mathbf{G}\mathbf{G}^T \in \mathbb{R}^{N^2 \times N^2} = \mathbf{U}_C \boldsymbol{\Lambda}_C^{1/2} \boldsymbol{\Lambda}_C^{T/2} \mathbf{U}_C^T.$$

You should note that $\mathbf{C}$ will be rank deficient because $N^2 \gg M$ and $\mathbf{U}_C(:, 1 : M) \in \mathbb{R}^{N^2 \times M}$ forms a basis matrix for $\mathbf{g}_i$, for $i \in 1, ..., M$. In other words, only M out of N^2 eigenvectors in $\mathbf{U}_C$ are needed to represent all training data due to the rank deficiency of $\mathbf{C}$. It is obvious that calculating $N^2 \times N^2$ $\mathbf{U}_C$ matrix is computationally inefficient and not necessary. However, we can consider an alternative. Consider $\mathbf{R} \triangleq \mathbf{G}^T\mathbf{G} \in \mathbb{R}^{M \times M}$ matrix.

$$\mathbf{R} = \mathbf{U}_R \boldsymbol{\Lambda}_R^{1/2} \boldsymbol{\Lambda}_R^{T/2} \mathbf{U}_R^T.$$

In vector form we can write as

$$\mathbf{R}\mathbf{u}_{R,i} = \mathbf{G}^T\mathbf{G}\mathbf{u}_{R,i} = \lambda_{R,i}\mathbf{u}_{R,i}.$$

Multiply $\mathbf{G}$ on the left, the above equation becomes

$$\begin{aligned}\mathbf{G}\mathbf{R}\mathbf{u}_{R,i} &= \mathbf{G}\mathbf{G}^T\mathbf{G}\mathbf{u}_{R,i} \\ &= \mathbf{C}\mathbf{G}\mathbf{u}_{R,i} \\ &= \mathbf{C}\mathbf{u}_{C,i} \\ &= \lambda_{C,i}\mathbf{u}_{C,i}, \quad \forall i \in 1, ..., M, \end{aligned} \tag{1}$$

with

$$\begin{aligned}\mathbf{u}_{C,i} &\triangleq \mathbf{G}\mathbf{u}_{R,i}, \quad \forall i \in 1, \ldots, M, \quad \text{and} \\ \mathbf{V} \triangleq \mathbf{U}_C(:, 1 : M) &= [\mathbf{u}_{C,1}, \mathbf{u}_{C,2}, \ldots, \mathbf{u}_{C,M}] \\ &= [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M] \in \mathbb{R}^{N^2 \times M}.\end{aligned}$$

Hence, according to (1), the eigenvectors for $\mathbf{C}$ can easily be computed by computing the eigenvectors of $\mathbf{R}$ and simply multiply them with $\mathbf{G}$, thus avoiding the computational expensive task of directly computing eigenvectors of $\mathbf{C}$.

Again, since $\mathbf{C}$ is rank deficient, $\mathbf{V}$ is sufficient to represent all of the face images in the training data set.

After the training is done, we move to the testing part. In this project, testing image set is the same as training data set but with added noise, resulting in a distorted set of images. We denote one of these images as $\mathbf{h}$ (note it should be zero-mean). We first transform the training images as $\mathbf{v}_i^T\mathbf{g}_i = g_{T,i}$, for $i = 1, ..., M$ and form the coefficient vector $\mathbf{g}_T = [g_{T,1}, g_{T,2}, ..., g_{T,M}]^T$. Similarly, we then form the i^{th} coefficient of the coefficient vector of the test image as $\mathbf{v}_i^T\mathbf{h} = h_{T,i}$, for $i = 1, \ldots, M$, so that the coefficient vector becomes $\mathbf{h}_T = [h_{T,1}, h_{T,2}, ..., h_{T,M}]^T$. To decide whether or not $\mathbf{h}_T$ is the person $\mathbf{g}_{T,i}$, it must be the "closest" to it. That is, it must satisfy

$$\mathbf{g}_{T,i,opt} = \text{argmin}_{\mathbf{g}_{T,i}} ||\mathbf{h}_T - \mathbf{g}_{T,i}||_2^2.$$

If image identified is incorrect, we call it a miss error. Hence if we test N_{test} number of images and obtain N_{errors} number of misses, then we can define a miss probability as

$$P_e \triangleq \frac{N_{errors}}{N_{test}}$$

and its value represents performance level of the eigenface algorithm at certain parameter values.

II. Project Assignment

In this project, you are asked to implement your own face identification system and test it under different amount of noise. You don't have to implement everything from scratch, because some of you may not be familiar with some aspects of this project. Hence, your task will be to complete some parts of it. In the zip file that is uploaded to the course website, you can find the MATLAB code package which includes a partial implementation of the eigenface facial identification algorithm. In it, you will find inside a readme file describing the functions of each file. Besides completing some of the files, you are also asked to answer some additional questions. Hence, your tasks are as follows

1) Complete functions for
   - Image I/O (10%) (load_set.m)
   - Efficient computation of singular vectors (15%) (comp_eig.m)
   - Min norm identification (15%) (min_norm.m)
2) Obtain a smooth and correct miss probability vs. SNR curve using all of the eigenvectors from the covariance matrix. (20%)
   The plotting function has already been written for you, all you have to do is use the code to plot the graph.

3) Implementing and using the algorithm using
    - 10% of the total number of eigenvector (10%)
    - 1% of the total number of eigenvector (10%)
4) How are the results from using a reduced number of eigenvectors different from the one using all of the eigenvectors? (10%)
5) Plot miss probability vs. SNR curve for all three cases (full set, 10% and 1% of eigenvectors). (10%)

Please put all your curves and your answers into a file and submit it together with your code via the E3 system.

You certainly **cannot** download code from the internet and submit it as your own, even if you have made minor modifications. You must write your own code by yourself from scratch. You are **strongly encouraged** to discuss with your classmates, your instructor, and/or TA about how the code can be implemented, but again, you must write your own code.