QEA Module 2: Facial Recognition

Braden Oh, Kyle Emmi

Abstract

Linear algebra and statistics can be applied together to perform facial recognition on pixel images. One major approach is to represent images as points in high dimensional space and to compare the Euclidian distance between an unknown image and each image in a known training set, in that space. This becomes computationally taxing for sizable images, however, as 256x256px images must each be represented in 256^2 dimensional space. The "Eigenface" method approaches this issue by using principle component analysis to reduce the number of dimensions that each image is represented in, computing the distance between far fewer dimensions in space. The "Fisherface" method optimizes the Eigenface approach, using linear discriminant analysis to change the bases of the Eigenface space so that the distance within classes (images of the same person) is minimized and the distance between classes is maximized, before the comparison is conducted. Both algorithms demonstrated an ability to recognize faces; Eigenfaces performed with 71.3% accuracy and Fisherfaces performed with 90.8% accuracy.

1 Introduction

In this demonstration, facial recognition was implemented as a comparison between vectorized 256x256px image matrices (each vector measures 256^2x1) where each element stores a pixel of data. Given an unknown facial vector, calculating the Euclidean distance between the unknown vector and every other facial vector in a set of known facial vectors will result in a correct guess with a high degree of accuracy but is extremely computationally intensive (as the magnitudes in each of the 256^2 dimensions must be calculated individually). Thus, in order to effectively compute the Euclidean distances in a timely manner, the dimensionality of the facial vectors must be reduced.

2 Eigenfaces

The chief problem in reducing the number of dimensions is deciding which dimensions are important enough to keep. PCA (Principle Component Analysis) provides a solution to this by using properties of the covariance matrix (which is symmetric) and its Eigenvectors (which will then always be orthogonal). By comparing only the n highest Eigenvalues' corresponding Eigenvectors, high-dimensional images can be effectively compared in a lower dimensional space for ease of computing Euclidian distances.

2.1 Pre-Processing

Given a set of pixel images, a number of pre-processing steps must be taken to format the data in a way that makes analysis easier. The first step often taken in image analysis is to reshape each lxw image matrix γ into an (lw)x1 vector of stacked columns Γ. Our images measured 256x256 pixels, thus the 256x256 image matrix given by

$$\gamma = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,256} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,256} \\ \cdots & \cdots & \cdots & \cdots \\ p_{256,1} & p_{256,2} & \cdots & p_{256,256} \end{bmatrix}$$

becomes the 256^2x1 column vector

$$\Gamma = \begin{bmatrix} p_{1,1} & p_{2,1} & \dots & p_{256,1} & p_{1,2} & p_{2,2} & \dots & p_{256,256} \end{bmatrix}^T$$

Given a set of M images, each reshaped image can be stored as a column in the data matrix L:

$$L = \begin{bmatrix} \Gamma_1 & \Gamma_2 & \dots & \Gamma_M \end{bmatrix}$$

The dimensions of L are 256^2xM; this can be thought of as M unique points in 256^2 dimensional space. The next step in pre-processing is to center this data cloud about the origin of our high-dimensional space. When Eigenvectors are calculated they exist with respect to an origin that lies outside the region of data, so pre-centering the data prevents the need to transpose Eigenvectors later.

Centering the data is achieved by subtracting the mean face vector. This mean vector μ, which measures 256^2x1, is generated by taking the simple average of all corresponding pixels (which are stored in rows) by

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

Centering the data about the origin is as simple as subtracting this average vector from each image vector in the data matrix. The final step in pre-processing is to multiply the data matrix by $\frac{1}{\sqrt{M}}$, which is a preparatory step for calculating the covariance matrix. The covariance matrix of any matrix B is given by

$$\frac{1}{n} BB^T$$

where n is the number of columns in the vector. By multiplying the data matrix by $\frac{1}{\sqrt{M}}$ we assure that simply multiplying the data matrix by its transpose generates a complete covariance matrix. Thus our final data matrix A is given by

$$A = \frac{1}{\sqrt{M}} \begin{bmatrix} \Gamma_1 - \mu & \Gamma_2 - \mu & \dots & \Gamma_M - \mu \end{bmatrix}$$

and the covariance matrix C for this data set is simply

$$C = AA^T$$

2.2 Principle Component Analysis (PCA)

The dataset A has the widest distribution of data along the directions of the principle component vectors. These directions can be determined by solving for the Eigenvectors of C. C is very large, however, having dimensions of 256^2x256^2, and so calculating the Eigenvectors directly is not computationally feasible. Singular value decomposition (SVD) provides a way to solve for these Eigenvectors indirectly in a much less computationally-intensive manner by taking advantage of the fact that AA^T and A^TA share a common subset of non-zero Eigenvalues while A^TA has the much smaller dimension MxM. Although AA^T and A^TA do not share Eigenvectors directly, their Eigenvectors are related by the equation

$$\frac{Av_i}{\sigma_i} = u_i$$

where v_i is an Eigenvector of (from the set of Eigenvectors of) A^TA, multiplied by A, and divided by the square root of its Eigenvalue, $\sigma_i = \sqrt{\lambda_i}$, to yield the corresponding Eigenvector of AA^T, u_i. Calculating each u_i for sigma values in descending order and storing these Eigenvectors as columns in a matrix U yields a complete and ordered set of the M largest non-zero Eigenvectors of the covariance matrix AA^T:

$$U = \begin{bmatrix} u_1 & u_2 & \dots & u_M \end{bmatrix} \mid \sigma_1 > \sigma_2 > \cdots > \sigma_M$$

U is a 256^2xM matrix where each column is an "Eigenface" of one of the original images in the dataset.

2.3 Dimensionality Reduction

These Eigenface vectors denote the axes of greatest (to least) variance in the dataset and provides a dimensional subspace that maximizes the distance between each piece of data, when each data vector is projected into that subspace. In other words, these Eigenface vectors provide a set of new axes that, when the dataset is projected into, spreads out the data as much as possible in that new coordinate frame.

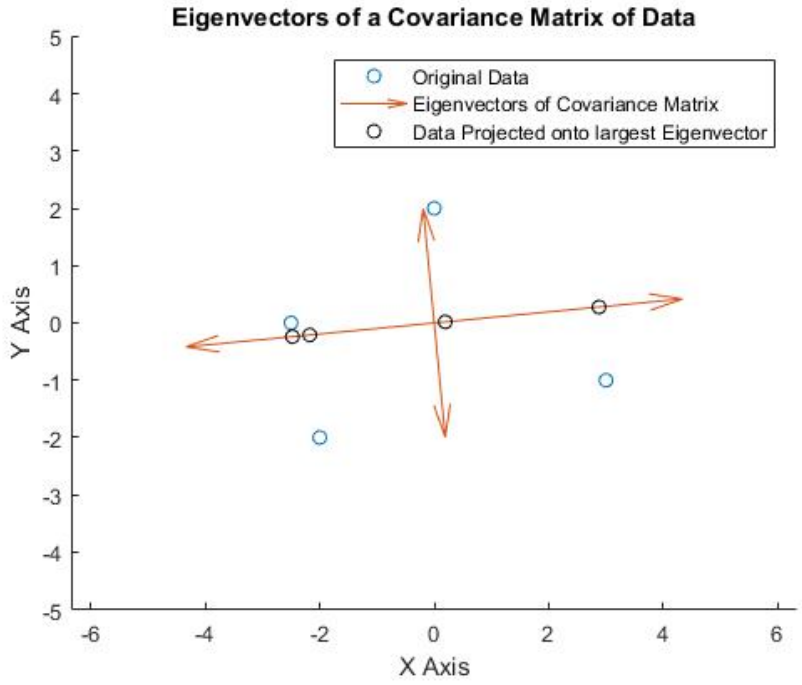

Figure 1: Pictured above are four points of data that are reduced from two dimensions to one dimension according to their covariance matrix's primary Eigenvector. Dimensional reduction makes comparisons between pieces of data significantly easier in this instance by compressing a point cloud to a line. A similar dimension reduction is occurring in the image dataset, but from one much higher dimensional space to another.

This subspace can ultimately consist of any number of Eigenface vectors: a greater number of vectors results in a higher accuracy of recognition but at the sacrifice of computational speed (likewise,

a smaller number of vectors can be computed more quickly at the sacrifice of accuracy). The Eigenface vectors are already ordered by single value magnitude in U, so any set of $\{u_1, u_2, \ldots u_n\} \mid n < M$ Eigenvectors can be used to compress the image data for purposes of this algorithm.

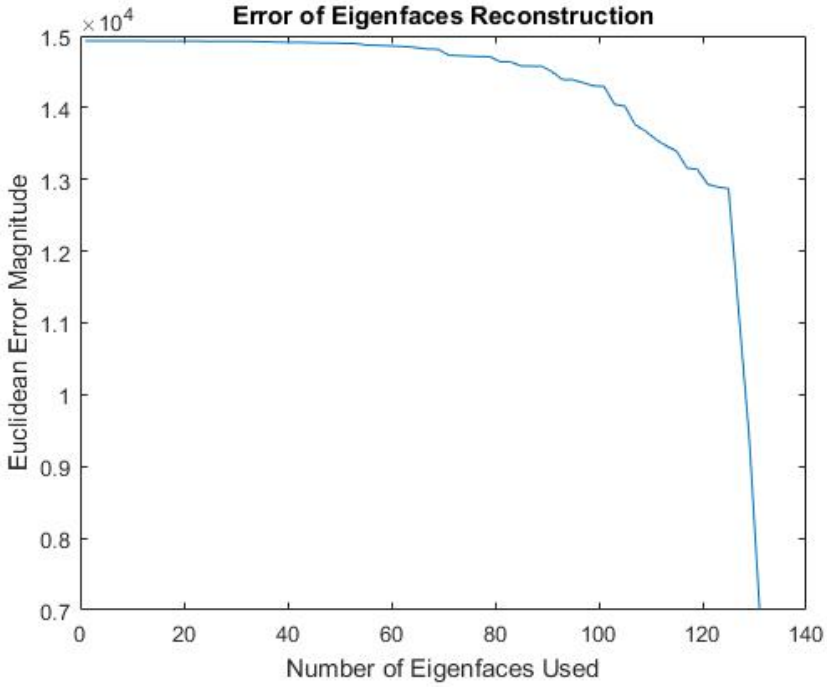

Figure 2: A face was reconstructed using varying values of n eigenface vectors and measured against a face reconstructed using M Eigenfaces.

Figure 2 shows that varying the value of n eigenfaces inside of U matters greatly when $n < M$, however as n increases, the need to continue adding more eigenfaces diminishes greatly. For this particular set of 132 eigenfaces, $n \approx 120$ is where the error starts to decrease dramatically.

A vectorized image can be compressed into a reduced subspace vector ω_i by multiplying U^T by the original image Γ_i translated toward the origin of the original coordinate space by subtracting the average face μ:

$$\omega_i = U^T \left[\Gamma_i - \mu\right]$$

Compressing each image from the training set and storing it as a column in a matrix Ω yields a compressed version of the entire original data set:

$$\Omega = \begin{bmatrix} \omega_1 & \omega_2 & \ldots & \omega_M \end{bmatrix}$$

2.4 Facial Recognition

"Recognizing" an unknown image Γ_x is as simple as compressing Γ_x into the Eigenface subspace and calculating the Euclidean distance between it and every other image in the compressed dataset. The comparison that yields the smallest magnitude (the least error) is the image from the training set that is the most similar to the unknown image.

The unknown image Γ_x is first compressed into the subspace by the same calculation used earlier to generate a compressed image, ω:

$$\omega_x = U^T \left[\Gamma_x - \mu\right]$$

where U^T is the same Eigenface matrix and μ is the same training set average as were used earlier. Calculating the Euclidean distance is achieved by taking the magnitude of the difference between a known image ω_i and the unknown image ω_x

$$e_i = ||\omega_i - \omega_x||$$

which can be done on every image in the original set Ω to generate an error vector E:

$$E = \begin{bmatrix} ||\omega_1 - \omega_x|| & ||\omega_2 - \omega_x|| & \dots & ||\omega_M - \omega_x|| \end{bmatrix}$$

The index of the smallest error is the index of the original image Γ_i that falls the closest to Γ_x.

3 Fisherfaces

3.1 Pre-Processing

The Fisherface algorithm eventually requires calculations to be performed on a matrix that measures (lw)x(lw) which is computationally infeasible for images from our training set, which measure 256x256 pixels (as the Fisherface algorithm will require operations on a 65,536x65,536 matrix that cannot be stored in a computer's RAM). As a result, data compression must be performed in advance.

The data compression scheme used in Eigenfaces captures the n axes of greatest variance in an image set, so will result in operations being performed on an nxn matrix, rather than an (lw)x(lw) matrix. Thus, the original dataset for this algorithm is the Ω matrix used in Eigenfaces.

3.2 Linear Discriminant Analysis (LDA)

The Fisherface algorithm seeks to project the original dataset into a subspace that maximizes between-class variance and minimizes within-class variance. As a result, the first step is to organize images into classes, where each class is a set of images of the same individual. A class c_i is simply the average of the images that compose that class:

$$c_i = \frac{1}{C} \sum_{n=1}^{C} \Gamma_n$$

where C is the number of elements in the class and Γ_n is an individual image in the class. Storing these class vectors as columns in a matrix yields a matrix of classes Ψ_c such that

$$\Psi_c = \begin{bmatrix} c_1 & c_2 & \dots & c_k \end{bmatrix}$$

Ψ_c has dimensions nxk where n is the length of a compressed image and k is the number of classes in the set. The next step is to quantify the within-class and between-class distributions of data. The between-class distribution S_b is calculated as

$$S_b = \sum_{i=1}^{k} C(\Psi_{ci} - \Psi)(\Psi_{ci} - \Psi)^T$$

where k is the number of classes, C is the number images in a class, and Ψ is the average image of the entire (compressed) dataset given by

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

For reference, each index for the summation operation on $k(\Psi_{ci} - \Psi)(\Psi_{ci} - \Psi)^T$ produces an nxn matrix. The sigma asks to simply sum these index matrices element-wise to generate the final matrix S_b.

The within-class distribution S_w is calculated as

$$S_w = \sum_{i=1}^{k}\sum_{\Gamma\in c_i}(\Gamma_k - \Psi_{ci})(\Gamma_k - \Psi_{ci})^T$$

which is a calculation performed in two parts; the inner sigma, notated by $\Gamma \in c_i$, denotes that the operation should be performed between every image Γ and the vector from Ψ_c that corresponds to its class while the outer sigma denotes that the operation should be performed on every class. In other words, for every image in the training set, subtract the Ψ_c vector that corresponds to the image's class when performing the calculation. The result of each operation $(\Gamma_k - \Psi_{ci})(\Gamma_k - \Psi_{ci})^T$ yields an nxn matrix; as before, simply sum these matrices element-wise to generate the final matrix, S_w, which will also measure nxn.

The final step is to determine the basis vectors which maximize the ratio between between-class and within-class separation. LDA states that, in general, the class separation S in a given direction $\vec{w}$ is given by

$$S = \frac{\vec{w}^T S_b \vec{w}}{\vec{w}^T S_w \vec{w}}$$

and thus maximizing this function will produce the basis vector that maximizes the ratio $\frac{S_b}{S_w}$. This calculation can, in fact, be simplified into a generalized Eigenvector equation of the form

$$S_b u = \lambda S_w u$$

which can be solved by MATLAB. The Eigenvectors with the largest magnitudes are in line with the axes that, when projected onto, maximize the ratio $\frac{S_b}{S_w}$.

3.3 Change of Base

Given the directions of the Eigenvectors u, projecting the images from the training set into this new "Fisherspace" orients the data in a way that makes it easy to form comparisons. A reoriented image ζ is formed by subtracting the average face Ψ from each image (thus centering the dataset about the origin) and multiplying each image ω_i from the training set by the transpose of the vector matrix u:

$$\zeta_i = u^T\left[\omega_i - \Psi\right]$$

Storing these reoriented images as columns in a data matrix Z yields a reoriented version of the entire data set:

$$Z = \begin{bmatrix}\zeta_1 & \zeta_2 & \dots & \zeta_M\end{bmatrix}$$

3.4 Facial Recognition

"Recognizing" an unknown image Γ_x is again achieved by calculating the Euclidean distance between the unknown image and every other image in the training set. An unknown image vector Γ_x must first have its dimensionality reduced to n by following the same PCA used to compress the Fisherface training set into matrix Ω (see $\omega_x = U^T \left[\Gamma_x - \mu\right]$ from section 3.3)

Once compressed, the unknown image ω_x must be centered about the origin of the training data coordinate frame by subtracting the average (compressed) face Ψ and then can be projected into the Fisherface subspace by multiplying by the transpose of the Eigenvector matrix u determined through LDA to generate an image vector ζ_x:

$$\zeta_x = u^T \left[\omega_x - \Psi\right]$$

Calculating the Euclidean distance is achieved by taking the magnitude of the difference between a known image ζ_i and the unknown image ζ_x

$$e_i = ||\zeta_i - \zeta_x||$$

which can be done on every image in the training set Z to generate an error vector E:

$$E = \begin{bmatrix} ||\zeta_1 - \zeta_x|| & ||\zeta_2 - \zeta_x|| & \dots & ||\zeta_M - \zeta_x|| \end{bmatrix}$$

The index of the smallest error is the index of the original image Γ_i that falls the closest to Γ_x.

4 Comparison of Performance

A "training set" of data was compiled by taking cropped, grayscale, photographs of students taking the QEA course and creating a vector of names that corresponded to each photo. For each algorithm, a linear transformation matrix (U and u, from above) were derived and used to compress/reorient the training set. A test set of images, different images of the same students photographed for the training set, were then transformed and the Euclidean distance from each test image to every point in the training set was calculated. The training image with the smallest distance to the test image indicated a match, and the name of the individual from that training image was returned and checked for correctness.

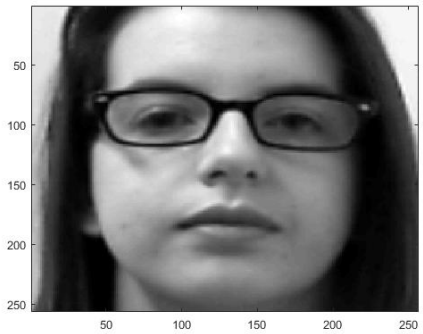

Figure 3: Example image of a training/test photo; the image is converted into grayscale and cropped to center the face.

4.1 Eigenfaces

The Eigenface training set consisted of 132 images: 66 students were each photographed twice, once with a blank face and once smiling. Via the mathematical processes described in section 2, the linear transformation matrix U was derived and multiplied through the training images to generate the comparison dataset Ω.

The Eigenface test set consisted of 122 images of various members of the QEA class; none of these images were present in the original training set. Each test image was transformed by matrix U (reducing the dimensionality) and the shortest Euclidean distance found between each reduced test image and each image in Ω. This algorithm correctly identified 87 of the test images, giving Eigenfaces an accuracy of 71.3%.

4.2 Fisherfaces

The Fisherface training set consisted of 172 images: 43 students were each represented four times. Via the mathematical processes described in section 3, the linear transformation matrix u was derived and multiplied through the training images to generate the comparison dataset Z.

The Fisherface test set consisted of 76 images of various members of the QEA class; again, none of these images were present in the original training set. Each test image was transformed by matrix u (reducing the dimensionality and changing the bases) and the shortest Euclidean distance found between each transformed test image and each image in Z. This algorithm correctly identified 69 of the test images, giving Fisherfaces an accuracy of 90.8%.

5 Conclusions

Eigenfaces and Fisherfaces are specific implementations of a recognition algorithm that represents data as points in high-dimensional space, extracts the dimensions of greatest variance, and performs simple Euclidean distance calculations in this reduced dimensional space. Both algorithms demonstrated a remarkable ability to recognize faces without performing any type of feature recognition (i.e. identifying eyes, nose, etc.), and are relatively efficient, as a transformation matrix only needs to be solved for one time before any number of comparisons can be performed.

That said, Eigenfaces and Fisherfaces work only on images that are extremely similar in nature; our entire dataset of facial images were taken head-on and cropped so that each face was approximately the same size. In addition, Eigenfaces is particularly vulnerable to changes in lighting. Fisherfaces can be made relatively robust to lighting differences with a sufficient variation of lighting within each set of class images. More advanced techniques, such as Elastic Bunch Graph Matching, must be employed to solve the issue of non-head-on facial recognition.

Despite these weaknesses, the ability to quickly perform recognition on sets of vectorized data is an incredibly powerful algorithm with applications much farther reaching than facial recognition: the same algorithm used by Eigenfaces and Fisherfaces can be applied to the field of voice recognition (as is performed by voice assistants like Siri and Alexa), sound identification (a particularly interesting use case being Shazam's music recognizer), and even in abstract applications such as gait identification (as one NINJA from last year even managed to identify whose pocket an accelerometer was in).

6 Letter to the Editor

We rewrote and changed the format of the Abstract to better accomplish the goal of being an abstract of our paper, whereas it was just an introduction beforehand. Because of this change, we merged the old abstract and the introduction into a proper Introduction.

We changed the notation of U in section 2.2 to reflect that the ordering of eigenvectors is due to their singular values rather than their magnitudes (all of which are unit length regularly).

We inserted a figure into section 2.3 to explain how the dimensionality of U can be reduced without much error in reconstruction. To accompany this we added a paragraph further explaining this acceptable dimensionality reduction.

We added a Comparison of Performance section using out data from in class tests. After this, we added a Conclusion outlining how each performed and relating our algorithms to other applications.