**Project 3: Principal Component Analysis and Eigenfaces for Face Recognition**

**CS479: Pattern Recognition**

**Terence Henriod**

**April 28, 2014**

**Introduction**

In this project the Principle Component Analysis (PCA) technique was used to demonstrate a dimensionality reduction technique as applied to face recognition.

The data used was a set of 48x60 grayscale images provided by the instructor, originally from the FERET database. 1204 images were used for “training” and 1196 images were used for testing the application developed by the student. The FERET database provides a diverse selection of faces in terms of gender and ethnicity.

The techniques and data described above were used to develop and test an application that found the mean (average) face, various “eigenfaces,” various matching or mismatching occurrences, as well as rejection and acceptance rates in face comparisons.

**Technical Discussion**

*Principal Component Analysis*

Principal Component Analysis seeks to reduce the dimensionality of the data by finding a way to represent the data in a space that still preserves most of the information, yet requires far fewer dimensions to do this.

PCA accomplishes this by finding the unit-length eigenvectors of a data set, and then keeping a subset of the eigenvectors corresponding to a subset of the largest eigenvalues. This a linear combination of the subset of eigenvectors will preserve a large amount of the data, usually with far fewer vectors.

*Computation of Eigenvalues and Eigenvectors*

In order to find the eigenspace representation of images, we need to combine them in a way that will be conducive to PCA. We can accomplish this by first vectorizing the images (accomplished by mapping the pixel values from a 2 dimensional representation to a 1 dimensional one), then computing the mean face (by averaging all pixel values from the training images), and then by creating a series of vectors by subtracting the mean face from each training face vector. Concatenating these column vectors gives a matrix A which will be used in computations.

This matrix A is used in finding the eigenvalues and eigenvectors of the space by making a square matrix by multiplying A by its transpose to get a square matrix of the size of all of the dimensions we wish to represent. From this we can find the eigenvalues and eigenvectors, and therefore the eigenspace needed to properly represent the image with non-redundant dimensions.

Finally, in order to reduce the dimensionality of our representation, we can exclude eigenvalues and their corresponding eigenvectors while still preserving a significant portion of the information. The amount of information preserved is found by

|  |  |
| --- | --- |
| Percentage of preserved information = | [1] |

where K is the number of kept eigenvalues, N is the total number of eigenvalues, and the eigenvalues/vectors are sorted in descending order by eigenvalue.

*Computation Reduction Techniques*

Note that AAT is a very large matrix (N2, the number of pixels per image). This results in very intensive computation for finding the matrix and its eigenvalues/vectors *u*. If we instead consider the matrix ATA, it is of size (M, the number of training images). This results in far fewer computations, but it produces a different set of eigenvalues/vectors, *v*. However, the eigenvectors will be the same for both approaches, and to recover the target eigenvalues *u* use:

|  |  |
| --- | --- |
|  | [2] |

and the original eigenvalues are recovered! This results in much better computational performance.

*Determining Similarity*

In order to determine how similar two face images are, we can compute the difference between their representations in the eigenspace. Since each image is represented as a linear combination in the eigenspace, two images differ by the weights used in their respective linear combination representations. Thus we can use the weights of the linear combination *w* (where *wi = uitᶲi*) in determining the “distance” between images. The Euclidean distance can be used:

|  |  |
| --- | --- |
| Distance between images | [3] |

where *k* is used to indicate items that correspond to a training image. However, the Mahalanobis distance can be used for better performance since it takes into account the shape of a distribution rather than the strict linear distance between objects:

|  |  |
| --- | --- |
| Distance between images | [4] |

where represents the eigenvalue corresponding with the eigenvector used with the given weight in the linear combination.

*Thresholds and ROC Curves*

In this project, when classifying face images, arbitrary distance thresholds are systematically used, with distance being defined in the previous section. This means that many arbitrary distance values were chosen, and if that distance was less than some arbitrary threshold value, then the faces were considered a match; alternatively, if the distance between two faces is greater than the threshold, then the faces are not considered matching.

By computing the rates of false acceptance and false rejection, a Receiver Operating Characteristic curve was generated, which is useful in helping to estimate the ideal threshold to be used for classification.

It should be noted that the term ROC curve is usually applied to tracking the discriminability between multiple classes, but in our case we only test against one class so the curve should be referred to as an *Operating Characteristic curve*.

**Results**

It should be noted that when it comes to images, that if one is expecting a light colored image, it is possible that extreme white values were turned black in the conversion from data to image due to the thresholding of .pgm formats, but the numerical data should be intact in the program/file’s data stores. Images are used to allow for rapid human understanding and not for actual analysis purposes. (This mostly refers to the “eigenfaces”, as the numerical data was stored in text files, not images, for robustness).

*Preliminary Results*



Figure 1: The mean (average) face from the set fa\_H.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| (a) | (b) | (c) | (d) | (e) |
|  |  |  |  |  |
|  |  |  |  |  |
| (f) | (g) | (h) | (i) | (j) |

Figure 2 (a – j): The “eigen-faces” corresponding to the 10 largest eigenvalues/eigenvectors.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| (a) | (b) | (c) | (d) | (e) |
|  |  |  |  |  |
|  |  |  |  |  |
| (f) | (g) | (h) | (i) | (j) |

Figure 3 (a – j): The “eigen-faces” corresponding to the 10 smallest eigenvalues/eigenvectors.

*Matching Exercise Results*

Unable to complete due to time constraints.

*Receiver Operating Characteristic Exercise*

Unable to complete due to time constraints.

**Appendix**

*Program Listings*

The student would like to thank the makers of the Eigen linear algebra C++ library for their open-source contribution to the world. Hours were saved in coding and debugging thanks to their contribution. Eigen can be found at <http://eigen.tuxfamily.org/>.

The project was coded in C++. This included the use of the C++ STL, particularly the vector template container.

Project\_3\_driver.cpp: The driver program for the assignment. This includes all routines used to accomplish the assignment.

*Complete program listings are attached to this document.*