Pattern Recognition Coursework 1

Jakub Mateusz Szypicyn
CID: 00846006
EEE4
jms13@ic.ac.uk

Jacobus Jukka Hertzog
CID: 00828711
EEE4
jjh13@ic.ac.uk

Abstract

Line1

Line3

Line5

Line7

1. Introduction

It is often desirable to be able to quickly and accurately transform handwritten text into computer text or to assign name to a person based on their face using computer programs. The process requires the computer to have some prior knowledge of what it is trying to compute. This is known as training data. Based on the training data we can build mathematical models which will allows us to recognise faces or letters.

In this paper we are investigating and describing basic methods of training and testing, such as Principal Coefficient Analysis (PCA), Nearest Neighbour classification (NN) and multiclass Support Vector Machine (SVM) classification including binary class SVM.

2. Eigenfaces

2.1. Data partition

A Matlab file containing face data `face.mat` has been provided for the purpose of this coursework. The file contains a 2576×520 matrix of face images. Each image is stored in a column. Given that the matrix has 520 columns there are 520 pictures of faces. Those pictures belong to 52 distinct persons. Therefore there are 10 pictures per person. Furthermore each picture has dimensions of 56×46 pixels.

In order to divide the data set into training and testing subsets, we have decide to preserve as much variance int he training data as possible. This would ensure that each set of faces is separated as far as possible, which potentially ensures higher identification rate.

The data was divided in the following ratio of testing to training: 20% to 80%. From each set of 10 pictures we have thus taken two most average pictures, based on the average pixel values. The two sets will be heron referred to as `training` (2576×416 matrix) and `testing` (2576×104 matrix).

2.2. PCA of face data

2.2.1 AA^T

Following the algorithm for Principal Component Analysis, we have first detrended the face images by subtracting a mean vector from all columns of `training`, which resulted in a matrix A, whose rows are now zero-mean. Following the above, the covariance matrix $S = \frac{1}{416} \times AA^T$ has been calculated. S has dimensions of 2576×2576.

The covariance matrix S uniquely describes the data by calculating its spread or variance denoted σ and its orientation. For face recognition we would like to make use of both of those properties. Namely, we would like to identify and keep vectors along which the data spread is the largest, disposing of dimensions which do not carry any spread information. This helps us to reduce problem size, decrease memory usage and increase performance.

The dimensions corresponding to largest data spread are given to us by calculating the eigenvalues and eigenvectors of S. We expect that there will be at most 416 non-zero eigenvalues. This follows from [1]. Given a rectangular matrix A, $S_1 = AA^T$ and $S_2 = A^TA$ share all non-zero eigenvalues. This means that the larger of the two matrices will have as many non-zero eigenvalues as the smaller one. Given that the dimensions of the smaller matrix are in our case 416×416, we expect that the larger matrix of 2576×2576 will return at most 416 non-zero eigenvalues. It of course can be the case, that there will be fewer non-zero eigenvalues. This proves to be the case with `training`. The resulting covariance matrix produces 415 significant eigenvalues. This can be accredited to one of two things:

1. The data is such that variance in one of the dimensions is actually zero.

2. The precision of `double float` calculations is insufficient. Since the data is very large, none of the 'zero' eigenvalues are actually equal to zero. They are however very small varying between 10^{-10} and 10^{-14}. This is shown in Figure 1 below.
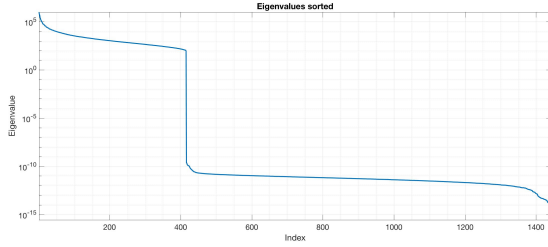

Figure 1. Sorted Eigenvalues of Covariance Matrix S

It can be seen that first 415 values are much greater than 1. The 416$^{\text{th}}$ value is around 10^{-10}. The three best eigenvectors, or eigenfaces corresponding to the three highest eigenvalues are shown below in Figure 2 . Finally the mean face which was initially subtracted from the face data is shown in Figure 3 .

Figure 2. Best 3 Eigenfaces of Covariance Matrix S

Figure 3. Mean Face from `training`

The 95% confidence bound would require us to use 121 eigenfaces. This constitutes a compromise between accuracy and performance, by reducing the problem dimensionality.

2.2.2 $A^T A$

Alternatively as suggested earlier we could compute a covariance matrix $S_T = \frac{1}{416} \times A^T A$, which now has dimensions of 416×416 instead of 2576×2576. We know [1]

that both matrices produce the same (meaningful) eigenvalues. Their plot in the descending order in Figure 4 proves the above claim.
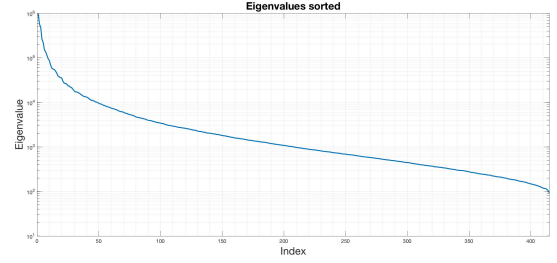

Figure 4. Eignevalues of S_T

Figure 4 allows to to deduce that the 416$^{\text{th}}$ value is actually zero. If it was non-zero the two methods of calculating a covariance matrix would result in very tiny, yet identical values. However the first method resulted in eigenvalue 416 being equal to 2.3×10^{-10}, whereas the second method gave a value of -1.9×10^{-12}.

We know however that eigenvectors will be different and therefore more computation is required to find the eigenfaces. Starting from the well-known equation:

$$A^T A x = \lambda x \tag{1}$$

Let us multiply both sides by A:

$$A A^T (Ax) = \lambda(Ax) \tag{2}$$

We deduce that the eigenvectors of AA^T are $u_i = Ax_i, \forall \lambda_i \neq 0$, where x_i is the ith eigenvector of $A^T A$. Having thus calculated x_i, we must multiply each of the vectors by the original zero-mean matrix `training` in order to obtain eigenfaces. However as we do it, we find that some of the images have inverted colours - i.e. the direction of the vector has been reversed. We have tried explaining that using Singular Vector Decomposition, however there is no mathematical reason for this behaviour. We believe that the MATLAB `eig` function causes this reversal. This is however not an issue, e.g. vectors $x = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and $y = \begin{bmatrix} -1 & -1 \end{bmatrix}$ are parallel. The best 3 eigenfaces are shown in Figure 5 . Note the direction reversal in the middle image.

Figure 5. Top 3 Eigenfaces

2

2.2.3 Comparison

We know that the method presented in section 2.2.1 is accurate. It is however more time consuming to calculate the eigenvectors of matrix with dimensions 2576×2576 rather than those of a 416×416 matrix. It is shown [2] that eigenproblem complexity is bounded by $O(n^2 log(n) + (n log^2(n)) log(b)$, where b is a measure of accuracy in bits 2^{-b}. However it should be noted that the eigenvectors of the second matrix are not what we are trying to calculate. Thus the latter method introduces an extra step. We have thus timed the full execution of both methods from implementing the `eig` function to having a dataset with properly ordered eigenvectors. The results are shown in Table 1 .

Algorithm	Time
AA^T	3.59 sec
$A^T A$	0.176 sec

Table 1. Computation Time for Both Algorithms

3. Applications of Eigenfaces

3.1. Face reconstruction

For the remainder of the work, the second algorithm (2.2.2) has been used.

Having determined the PCA bases we are now able to reconstruct faces. We expect that those from the training set will be reconstructed more accurately for any given number of PCA components.

Figure 6. Reconstruction of a Training Face: Original, 50 bases, 121 bases, 180 bases

Figure 7. Reconstruction of a Test Face: Original, 50 bases, 121 bases, 180 bases

The reconstruction errors measured as the Euclidean norm of the difference of two vectors: $OriginalFace$ and $ReconstructedFace$ of the images shown in Figures 6, 7 and 8 are presented in the Table 2 below.
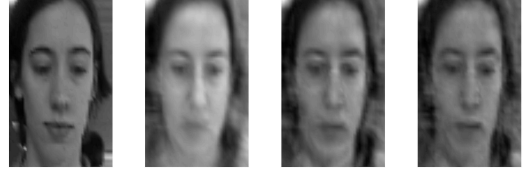

Figure 8. Reconstruction of a Test Face 2: Original, 50 bases, 121 bases, 180 bases

Image	Number of bases	Reconstruction Error
Training Face	50	884.4740
Training Face	121	556.3322
Training Face	180	347.7328
Training Face	400	18.1174
Test Face 1	50	864.7057
Test Face 1	121	721.7824
Test Face 1	180	676.7093
Test Face 1	400	582.9752
Test Face 2	50	816.2902
Test Face 2	121	649.3541
Test Face 2	180	575.2193
Test Face 2	400	474.1061

Table 2. Reconstruction error for various images and number of PCA bases

Additionally, we can determine the distortion measure of the reconstruction. For any given number of PCA bases, the distortion measure will be the same for any image from the same training set. The distortion measure has been calculated as: $J = \sum_{n=M+1}^{D} \lambda_n$, where M represents number of PCA bases used to reconstruct the image and D it the total number of eigenfaces. It should also be noted that the eigenvalues are ordered in the descending order s.t. $\lambda_1 > \lambda_2 > ... > \lambda_{M+1} > ... > \lambda_D$.
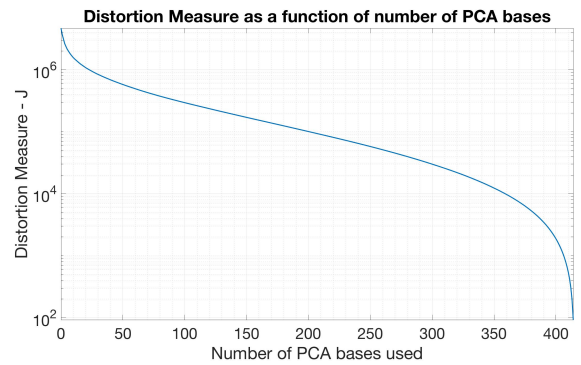

Figure 9. Distortion Measure for varying number of PCA bases

From the Figures 6, 7 and 8 we can infer that 121 eigenfaces (the 95% bound calculated earlier) is sufficient to a human eye in order to assign class to a given image. 50 PCA bases results in a very fuzzy image, though the main characteristics of the class can be identified. Finally, 180 eigenfaces produce a relatively sharp image, though it isn't much different from the one obtained with 121 bases, as shown in Table 2.

The test results shown in Table 2 show as expected a monotonous trend. As we increase the number of eigenfaces the reconstruction error (or the D-dimensional distance between original and reconstructed images) decreases. We also observe that both face images from the `testing` set produce larger errors than the face from `training` set for larger amount of PCA bases. The reconstruction error of the training faces can actually reach zero, when we use all of 415 eigenfaces. This however cannot be said about the test faces.

Similarly, we observe that the distortion is monotonously decreasing as we increase number of the PCA bases. This is expected as, with each extra PCA base used to reconstruct we take away the biggest eigenvalue from the total sum.

3.2. Nearest Neighbour (NN) classification

In order to perform the PCA-based face recognition, the Nearest Neighbour classification method was used. Having calculated the projections of each normalised training face onto the eigenspace, we could take a new image and classify it into a class. This was done by calculating the new image's projection onto the eigenspace, and compare it to the existing projections. The new image was given the class of the projection that it is closest to. The following classifier equation was used to acheive this:

$$e = min_n||\omega - \omega_n||, \quad n = 1, ..., N \qquad (3)$$

where ω is the projection of the new image onto the eigenspace, ω_n is the projection of the training vector x_n and N is the number of training vectors.
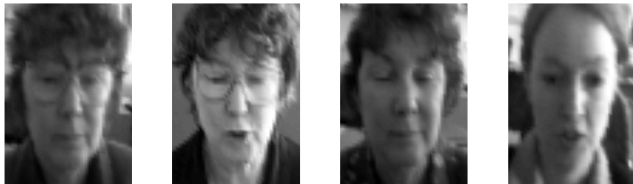
Figure 10. Examples of a successful classification and a failed classification: Successfully Classified Testing Face, Example from Assigned Class, Unsuccessfully Classified Testing Face, Example from Assigned Class,

Using this method, each testing vector was classified. In 70.1293%, the classification was successful and testing images were assigned to the correct class. This method is clas-

sification works only on how similar the testing image is to another similar image, so it is easily fooled by incorrect pictures with similar lighting, or testing images that have a particular difference from the training data, i.e. the removal of the glasses in the failure example in Figure 10.

4. Multi-Class SVM

References

[1] Inderjit Dhillon. *CS 391D Data Mining: A Mathematical Perspective Fall 2009*. The University of Texas at Austin, September 2009.

[2] Victor Y.Pan, Zhao Q. Chen, Ailong Zheng. *The Complexity of the Algebraic Eigenproblem*. Lehman College and Graduate Center, CUNY, NY, December 1998