Pattern Recognition Coursework 1

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

Jacobus Jukka Hertzog
CID:
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.

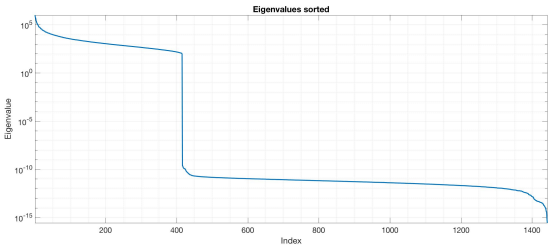

Figure 1. Sorted Eigenvalues of Covariance Matrix S

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`

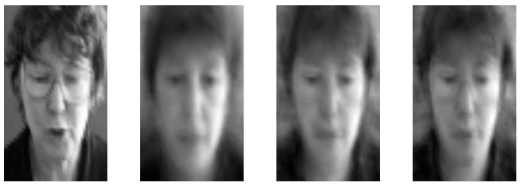

Figure 4. Comparison of Reconstruted Images: Original Image, 20 Eigenfaces, 50 Eigenfaces, 80 Eigenfaces

We have determined empirically that using just 80 out of 415 eigenvectors is suitable for facial recognition - Figure 4. 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 5 proves the above claim.

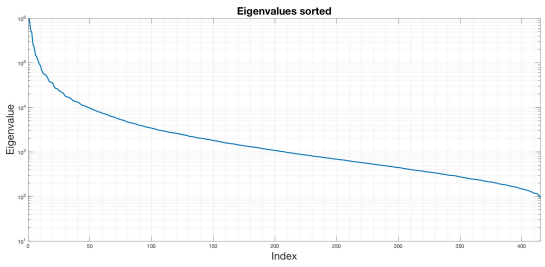

Figure 5. Eignevalues of S_T

Figure 5 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 6 . Note the direction reversal in the middle image.

Figure 6. Top 3 Eignefaces

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 $O(n^2 log(n) + (nlog^2(n))log(b)$, 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 **??** . INSERT TIMES HERE IN A TABLE

3. Applications of Eigenfaces

4. DAYYYUUM

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