Human Face Reorganization with Principal Component Analysis In Unsupervised Learning

Jiaying He, Cognitive Science, University of California, San Diego

Abstract

ISIS threat is rapidly increased these days, and terrorists are hard to catch. Camera (public CCTV, personal phone) and social media provide the chance to capture the pictures when attack happens. I want to use one picture of one person to find other pictures even in different angels that he or she involved. The computation approach taken in this paper is motivated by both PCA and Euclidean distance. I did not cluster the dataset using KNN, I use Euclidean distance to calculate the nearest neighbor and find the corresponding image. The recognition experiments are performed using the AT&T face recognition dataset. The dataset contains 40 people with 10 different angels and lightings picture, total 400 images. I find that calculate the minimum distance and find nearest neighbor is able to find the corresponding image of human face accurately and quickly.

1. Introduction

Face recognition is interesting and widely applied because it can contribute not only to theoretical insights but also to practical application. Computers that recognize faces could be applied to a wide variety of problems, including criminal identification, security systems, image and film processing, and human-computer interaction. [1] For example, the ability to capture a face and distinguish it from a large dataset such as criminal dataset would make it possible to help police to catch terrorists. And if this models simple enough, it could also apply on household, such as face recognition lock, residents' faces will be scanned and store in a data center. And if anyone want to get into the house, he or she need a face scan and see if the face is matching to the stored dataset. I focus on applying this method to a simple, fast, and accurate in constrained environments that everyone can use. The ideal result is using this algorithm can successfully learn a new face and later recognize it.

2. Methodology

2.1 PCA, eigenvector, eigenvalue, eigenfaces

PCA is a great way to conduct dimensionality reduction in unsupervised learning, which is used for compression and face recognition problems, and is a common technique for finding patterns in data of high dimension. It is also known as eigenspace projection or karhunen-loeve transformation. [2] It enables me to reduce the dimension of 400 faces to two dimensions matrix and faster analysis with computational cost. In linear algebra, an eigenvector or characteristic vector of a square matrix is a vector that does not change its direction under the associated linear transformation; the evaluable information will not be changed.

First I want to get the mean face to calculate the deviation of each image from mean image. Then conduct PCA to calculate eigenvector and eigenvalue. Second I sort and eliminating eigenvalues, all eigenvalues of matrix are sorted and those who are less than a specified threshold are eliminated. Third, Projecting centered image vectors into facespace. All centered images are projected into facespace by multiplying in Eigenface basis's. Projected vector of each face will be its corresponding feature vector. I project face images onto a feature space that spans the significant variations among know face images. Eigenfaces, the significant features, are the eigenvectors (principal components) of the set of faces; they do not necessarily correspond to features such as eyes, ears, and noses. The projection operation characterizes an individual face by a weighted sum of the eigenface features and so to recognize a particular face it is necessary only to compare these weights to those of known individuals. Some particular advantages of this approach are that it provides for te ability to learn and later recognize new faces in an unsupervised manner, and that it is easy to implement using a neural network architecture. [1]
Eigenvectors and Eigenvalues are dependent on the concept of orthogonal linear transformation. An Eigenvector is basically a non-zero vector. The dominant Eigenvector of a matrix is the one corresponding to the largest Eigenvalue of that matrix. This dominant Eigenvector is important for many real world applications. Steps used to find the features for facial recognition [3]

- Organizing the data set- Consider the data having a set of M variables that are arranged as a set of N data vectors. Thus the whole data is put into a single matrix X of dimensions M x N.
- Calculating the mean-

  $$\mu_x = \frac{1}{N}\sum_{n=1}^{N} X[m,n]$$

  where is the mean of the matrix X; m and n are indices and m=1, 2… M and n=1, 2… N

- Subtracting off the mean for each dimension-

  $$X = X - \mu_x$$

- Calculating the covariance matrix- Covariance has the same formula as that of the variance. Assume we have a 3-dimensional data set (p, q, r), then we can measure the covariance either between p and q, q and r or r and p dimensions. But measuring the covariance between p and p, q and q, r and r dimensions gives the value of variance of the respective p, q, r dimension. Variance is measured on a single dimension whereas covariance on multidimensions.
  - For 1-dimension,

$$Cov(x) = Var(x) = \frac{\sum_{i=1}^{N}(X-\mu_x)(X-\mu_x)}{N-1}$$

- Calculating the Eigenvectors and Eigenvalues of the covariance matrix- For computing the matrix of Eigenvectors that diagonalizes the covariance matrix C

$$E \cdot Cov \cdot E^{-1} = D$$

2.2 Euclidean distance

The classifier [3] based on the Euclidean distance has been used which is obtained by calculating the distance between the image which are to be tested and the already available images

used as the training images. Then the minimum distance is observed from the set of values. In testing, the Euclidean distance (ED) has been computed between the new (testing) image Eigenvector and the Eigen subspaces for each faces, and minimum Euclidean distance based classification is done to recognize the faces of the input image. The formula for the Euclidean distance is given by

$$ED = \sqrt{\sum(x_2 - x_1)^2}$$

Similar to K nearest neighbor, the way I used to find the similar image to the given face is to compare the distance between images and select the image with the closest Euclidean distance. So I first extracted the PCA features from test image, then calculated the Euclidean distances between the projected test image and the projection of all centered training images are calculated. Test image was supposed to have minimum distance with its corresponding image in the training database. For any give face, I can also find similar faces according to their distance within the face space

2.3 Dataset

The dataset is The dataset contain 40 folders/people, and each folder contains 10 images which taken under different lightings, angels, facial expressions(open/closed eyes, smiling/ not smiling), and facial details (glasses/ no glasses) all the images were taken against a dark homogeneous background with the subjects in an upright, frontal position[1]. I transform each image to a column vector and combine 400 columns as a huge matrix, which enable me easier to calculate mean, eigenvector, knn, etc. I randomly pick an image from the database as testing set, and use the rest 399 of the images for training. I use the randomly selected picture to test the algorithm.

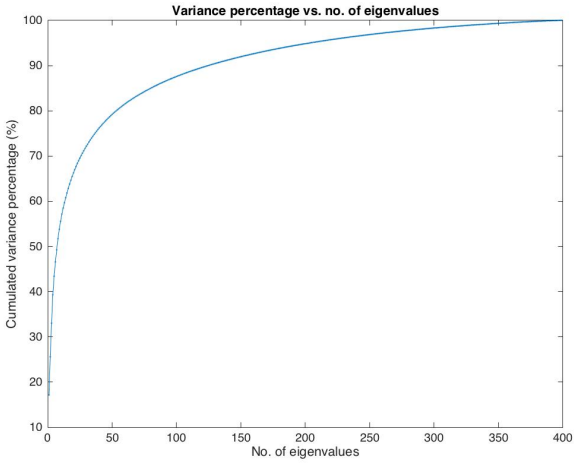
3. Result

For any given face, this algorithm is able to find the similar faces according to distances between new face and given face, and select the minimum distance face.

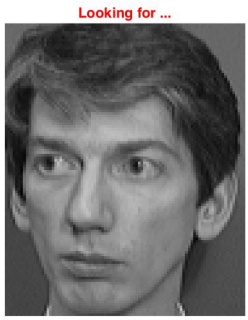
The graph(1) is the average of all faces. Graph (2) is the eigenfaces.
Graph (3) is the plot of percentage of total variance versus number of eigenvalues to get an idea as how PCA can "squeeze" the variance into the first few eigenvalues
Graph (4) is the test image and its corresponding recognized image.

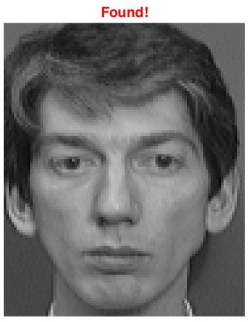
Graph (1)

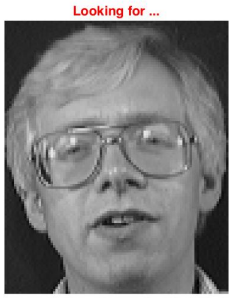
Graph (2)

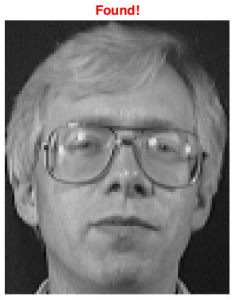


Graph (3)









5. appendices

createDataset.m

```
level1list = dir('*');
level1list(~[level1list.isdir]) = [];
tf =
ismember({level1list.name},{'.','..'});
level1list(tf) = [];
faceData = []
cur = 1;
for K = 1:length(level1list);
    subdir =
dir(fullfile(level1list(K).name,'*.pgm'))
;
    if isempty(subdir);
        continue
    end
    for i = 1:length(subdir);
        I =
imread(fullfile(level1list(K).name,
subdir(i).name));
        I = im2double(I);
        A = zeros(112*92,1);
        A(1:end) = I(1:end);
        faceData = [faceData A];
    end
end
save faceData
```

PCA.m

```
load 'faceData.mat'
%% Initializations
ri=round(400*rand(1,1));
test_img=faceData(:,ri);
train_img=double(faceData(:,[1:ri-1
ri+1:end]));

%% Get mean face
mean_face =
reshape(mean(train_img,2),112,92) ;
mean_column =
reshape(mean_face,[10304,1]);
figure(1)
imagesc(mean_face) ;
colormap(gray) ;

%% Calculating the deviation of each
image from mean image
A = [];
for i = 1 : size(train_img,2)
    temp = double(train_img(:,i)) -
mean_column;
    A = [A temp];
end

%% PCA
x = mean(train_img,2);
C = A*A'/size(train_img,2);
L = A'*A;
[V,D] = eig(L);
[sv si] = sort(diag(D),'descend');
Vs = V(:,si);
%%  plot the percentage of total variance
versus number of eigenvalues to get an
idea as how PCA can "squeeze" the
variance into the first few eigenvalues
 [rowDim, colDim]=size(faceData);
 meanFace=mean(double(cat(3, faceData)),
3);
 fprintf(' ===> %.2f sec\n', toc);
 fprintf('Perform PCA... '); tic
 [A2, eigVec, eigValue]=pca(A);
 fprintf(' ===> %.2f sec\n', toc);
 cumVar=cumsum(eigValue);
 cumVarPercent=cumVar/cumVar(end)*100;
 plot(cumVarPercent, '.-');
 xlabel('No. of eigenvalues');
 ylabel('Cumulated variance percentage
(%)');
 title('Variance percentage vs. no. of
eigenvalues');
 fprintf('Saving results into
eigenFaceResult.mat...\n');
 save eigenFaceResult A2 eigVec
cumVarPercent rowDim colDim
 DS.input=A2;

DS.outputName=unique({faceData.parentDir}
);
 DS.output=zeros(1, size(DS.input,2));
 for i=1:length(DS.output)

DS.output(i)=find(strcmp(DS.outputName,
faceData(i).parentDir));
    DS.annotation{i}=faceData(i).path;
 end
myTic=tic;
maxDim=100;
rr=pcaPerfViaKnncLoo(DS, maxDim, 1);
plot(1:maxDim, cumVarPercent(1:maxDim),
```

```
'.-', 1:maxDim, rr*100, '.-'); grid on
xlabel('No. of eigenfaces');
ylabel('LOO recog. rate & cumulated
variance percentage');
[maxValue, maxIndex]=max(rr);
line(maxIndex, maxValue*100, 'marker',
'o', 'color', 'r');
legend('Cumulated variance percentage',
'LOO recog. rate', 'location',
'southeast');
fprintf('Optimum number of eigenvectors
= %d, with recog. rate = %.2f%%\n',
maxIndex, maxValue*100);
 toc(myTic)

%% Sorting and eliminating eigenvalues
L_eig_vec = [];
for i = 1 : size(V,2)
    if( D(i,i)>1 )
        L_eig_vec = [L_eig_vec V(:,i)];
    end
end
Eigenfaces = A*L_eig_vec;
%% Projecting centered image vectors into
facespace
ProjectedImages = [];
Train_Number = size(Eigenfaces,2);
for i = 1 : Train_Number
    temp = Eigenfaces'*A(:,i);
    ProjectedImages = [ProjectedImages
temp];
end
%% Extracting the PCA features from test
image
temp = test_img(:,:,1);
[irow, icol] = size(temp);
InImage = reshape(temp',irow*icol,1);
Difference = double(InImage)-mean_column;
ProjectedTestImage =
Eigenfaces'*Difference;
%% Calculating Euclidean distances
Euc_dist = [];
for i = 1 : Train_Number
    q = ProjectedImages(:,i);
    temp = ( norm( ProjectedTestImage -
q ) )^2;
    Euc_dist = [Euc_dist temp];
end

[Euc_dist_min , Recognized_index] =
min(Euc_dist);
OutputName =
strcat(int2str(Recognized_index),'.pgm');
%% display
figure(4)
subplot(1,2,1)
imshow(reshape(test_img,112,92));
title('Looking
for ...','FontWeight','bold','Fontsize',1
6,'color','red');
subplot(122);
subplot(122);
imshow(reshape(faceData(:,Recognized_inde
x),112,92));
title('Found!','FontWeight','bold','Fonts
ize',16,'color','red');
```

6. References

[1] Matthew Turk, Alex Pentland, “Eigenfaces for Recognition”, The Media Laboratory. 1991, Journal of Cognitive Neuroscience, Volume 3, Number 1

[2] Satinkar Suhas S, “Face Recognition using Principal Component Analysis and Linear Discriminant Analysis on Holistic Approach in Facial Images Databse”, IORS Journal of Engineering, Vol 2, Issue 12 Dec. 2012

[3] Jeemoni Kalita, “Recognition of Facial Expression Using Eigenvector Based Distributed Features and Euclidean Distance Based Decision Making Technique”, International Journal of Advanced Computer Science and Applications, Vo;. 4, No. 2, 2013