Principal Component Analysis

Huiji (Chelsea) Ying & Yuhao Wan

March 14, 2018

Abstract

Most of the real world data, including images, can be represented as vectors in high dimensional space. Vector representation of high dimensional data allows us to explore their geometric and linear algebraic properties. However, our intuition of high dimensional space is very limited. Thus, matrix factorization and dimension reduction is extremely helpful for our understanding. Singular value decomposition (SVD) is arguably the best tool to deal with matrix data, and we focus on one of its most useful applications, principal component analysis (PCA) for our project. In this expository paper, we explore the problem of 2-dimensional face recognition and face classification through principal component analysis.

1 Introduction

To tackle the 2-dimensional face recognition problem, one naive approach would be to classify the images directly. One common method is nearest neighbor search, which is to find the closest face among all faces. The steps are first represent each face as a single entity, then construct distance between faces, and lastly find the one with smallest distance with the given face. However, this naive approach has two main limitations:

1) It is computationally heavy because the number of dimensions represented is the number of pixels of the image, which is usually on the order of 10^3 and more. When calculating distances, one has to perform as many computations as the number of dimensions.

2) The distance measure takes into account of noise which would mislead the result. To be more specific, noise is problematic because many pixels are not that important. In the face problem, we want to focus on important pixels that are associated with important facial features like our eyes, nose, ears, facial hair, etc., and we wish to ignore unimportant pixels in determining a face such as those consisting the background. Thus, distance measure based on those distracting unimportant pixels would distort the actual distance between faces, which then lead to inaccurate classification of faces.

Therefore, it is crucial for us to eliminate noise and also control computation complexity. One smart way to achieve both is a concept called dimension reduction, which is closely linked to principal component analysis.

Before moving on the details of PCA, we first recap our understanding of singular value decomposition. SVD finds the best-fitting k-dimensional subspace for the set of n data points. Consider a 1-dimensional subspace, namely a line through the origin. We know the best-fitting k-dimensional subspace can be found by k applications of the best fitting line algorithm, where on the i^{th} iteration we find the best fit line perpendicular to the previous $i-1$ lines. When k reaches the rank of the matrix, from these operations we get an exact SVD of the matrix.

Often a data matrix A is close to a low rank matrix and it is useful to find a good low rank approximation to A. We have seen in the class that for any k, the singular value decomposition of A gives the best rank-k approximation to A in a well-defined sense. This decomposition of A can be viewed as analogous to writing a vector x in some orthonormal basis $(v_1, v_2, v_3, \cdots, v_d)$. The coordinates of $x = (x \cdot v_1, x \cdot v_2, \cdots, x \cdot v_d)$ are the projections of x onto the v_i's. For SVD, this basis has the property that for any k, the first k vectors of this basis produce the least possible total sum of squares error for that value of k.

Thus, the goal of PCA is to identify the most meaningful orthonormal basis to re-express a data set as a linear combination of the basis vectors. The hope is that this new basis will filter out the noise and reveal the hidden structure of the data.

2 Algorithm

2.1 Vectorizing Face Images

Say an grey-scale image has $p \times q$ pixels. It can be described as a 2-dimensional p-by-q matrix $I(p, q)$ with each entry I_{xy} representing the intensity of each grey pixel in the image.

Figure 1: Example face images in the dataset.

We can also reshape such a matrix I into a vector with dimension $p \times q$, so that a typical 100-by-100 image becomes a vector of dimension 10,000, or equivalently, a point in 10,000-dimensional space. Then, a set of n faces, with the same $m = p \times q$ size, then maps to a collection of points in this high-dimensional space.

In this high-dimensional space, each pixel location is represented by a dimension. The entire dataset of n faces can then be represented as a single m-by-n matrix X with each row representing all measured pixel values at a pixel location, and each column representing a face image vector in the m-dimensional space.

2.2 Calculating Eigenfaces

Since all face images have similar features specific to human facial structure, the collection of points representing face images will not simply be randomly distributed in this high-dimensional space. Thus, we can describe them by a relatively low dimension subspace. Intuitively, we wish to find the principal components.

2.2.1 Covariance Matrix

Definition 2.1. (**Covariance**) For two variables x_1 and x_2, the covariance measures how strongly the two variables are associated:

$$Cov(x_1, x_2) = \frac{1}{n-1} \sum_{i=1}^{n} (x_1^{(i)} - \bar{x}_1)(x_2^{(i)} - \bar{x}_2) \tag{1}$$

where n denote the number of samples, and $\bar{x}_1$ and $\bar{x}_2$ denote the sample mean of x_1 and x_2 respectively.

Definition 2.2. (**Covariance Matrix**) A covariance matrix C for a set of variables $x_1, x_2, \cdots, x_n$ is a $n \times n$ matrix where C_{ij} is the covariance between x_i and x_j.
(If $i = j$, C_{ii} is the variance of x_i.)

Note that the concept of covariance is essential in PCA, as we are just looking for the directions on which the variables are mostly correlated. To derive a compact formula for the covariance matrix, we store our data as an $m \times n$ matrix X, where m is the number of features (or variables) and n is the number of observations (or samples).

The first step we perform is to standardize the dataset so that every feature has sample mean 0. The reason we want to do this is that we are interested in finding the best fitting affine subspace, where an affine subspace is a linear translation of a subspace. An affine space is a geometric structure that generalizes the properties of Euclidean spaces in such a way that these are independent of the concepts of distance, keeping only the properties related to parallelism and ratio of lengths for parallel line segments. So the reason we want to best-fit an affine space is to gather the statistics of the data and how it varies with its mean. Thus, we need to subtract the centroid from the data so the data mean would center at 0.

Figure 2: Mean face calculated from the input images superset of Figure 1 .

After this demean standardization, one can write the covariance matrix as

$$C = \frac{1}{n-1} X X^T \tag{2}$$

Proof. Note that the covariance between two variable x_i, x_j is given by

$$cov(x_i, x_j) = \frac{1}{n-1} \sum_n (x_i^{(n)} - \overline{x_i})(x_j^{(n)} - \overline{x_j})$$

where $x_i^{(n)}$ is the nth observation of the feature x_i, and $\overline{x_i}$ is the sample mean across all observations. If the dataset is standardized, then $\overline{x_i} = \overline{x_j} = 0$. Then, we can write

$$cov(x_i, x_j) = \frac{1}{n-1} \sum_n x_i^{(n)} x_j^{(n)} = \frac{1}{n-1} x_i^T x_j$$

Since our dataset X is stored as the columns are different features. $x_i^T x_j$ is the ith column dot product with jth column. We can use matrix multiplication and get that

$$C = \frac{1}{n-1} X X^T$$

□

Then we will be looking for the eigenbasis of the covariance matrix C. First we recall the following fact:

Theorem 2.3. *For any matrix A, let $A = USV'$ be the SVD decomposition of A. Then the right singular vectors (the column vectors of V) are exactly the eigenvectors of AA^T.*

Proof. See class notes. □

Thus one can directly apply some SVD decomposition algorithm on our the transformation of our data matrix $Y = \frac{X^T}{\sqrt{n-1}}$ and then obtain the eigenbasis of

$$Y^T Y = \frac{X}{\sqrt{n-1}} \cdot \frac{X^T}{\sqrt{n-1}} = C$$

by looking at the right singular vectors of the SVD result. Then we select the appropriate number of those right singular vectors to be our principle components, i.e. eigenfaces in this application. Figure 3 shows the top ten eigenfaces derived from input images superset of Figure 1.

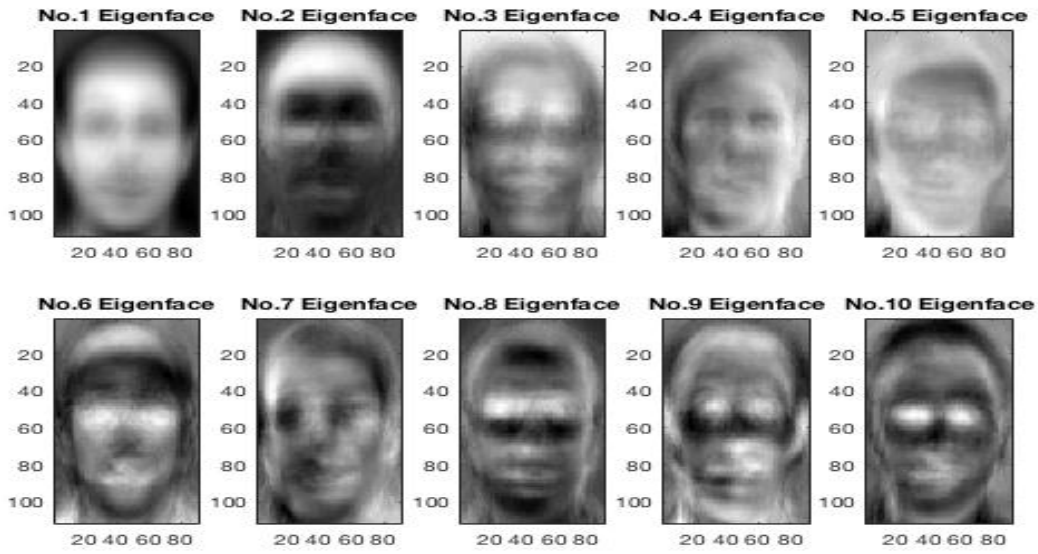

Figure 3: Top ten eigenfaces calculated from the input images superset of Figure 1 .

2.3 Using Eigenfaces to Classify Face Images

Eigenfaces are the eigenvectors of the face dataset matrix. They identify axes with maximum variance in the dataset. Any face image in the dataset, which is a column in the original matrix, can be constructed using a linear combination of the eigenfaces. This means we can represent any face with a set of weights by projecting all columns of the original face matrix onto the new set of eigenbasis.

One thing to note is that while some eigenfaces seem to correspond to real features such as hair or glasses, in general, eigenfaces are not human faces. They are just differences images used to recreate any face in the dataset.

Since any face image in the dataset can be reconstructed with the set of eigenfaces, and accurate construction is not a requirement for recognition, here we want to explore how many eigenfaces is appropriate for the task of classification and recognition.

The eigenfaces need to be computed only once, and then all faces in the database can be represented by a set of weights, aka a low dimension vector, a much more compact representation.

As we've shown above, we have the right singular vectors of $Y = U\Sigma V'$ as the new set of eigenbasis for our data. To find the appropriate number of principal components, we set

the amount of variance we want to keep, i.e. the amount of details we want to preserve, in the original dataset. We define the cumulative energy content

$$E_k = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i}$$

for the kth eigenvector as the sum of the corresponding 1st through kth eigenvalues divided by the sum of all eigenvalues. So E_n will be 1 as if we keep all n eigenvectors as our principal components, we have fully described our dataset. Then, we find r number of principal components to keep if the E_r passes the variance threshold we defined previously.

Then we take $P = V_{1:r}$, the columns 1 through r of V, $r < n$, as our lower-dimensional principle components, i.e. eigenfaces. To come up with weights (W_k) for a real face image (X_k) in the dataset, we need to project the 10304 dimensional input vector to the set of eigenfaces by

$$X_k = PW_k \Longrightarrow W_k = P^T X_k$$

and we do that for $k = 1, ..., n$ to project all images. Figure 4 shows a face image and its projection onto a ten-dimensional eigenface space. Then, the new set of weight vectors W represents the entire dataset and defines how the predefined face classes are distributed in this lower-dimensional space with eigenfaces as its basis.

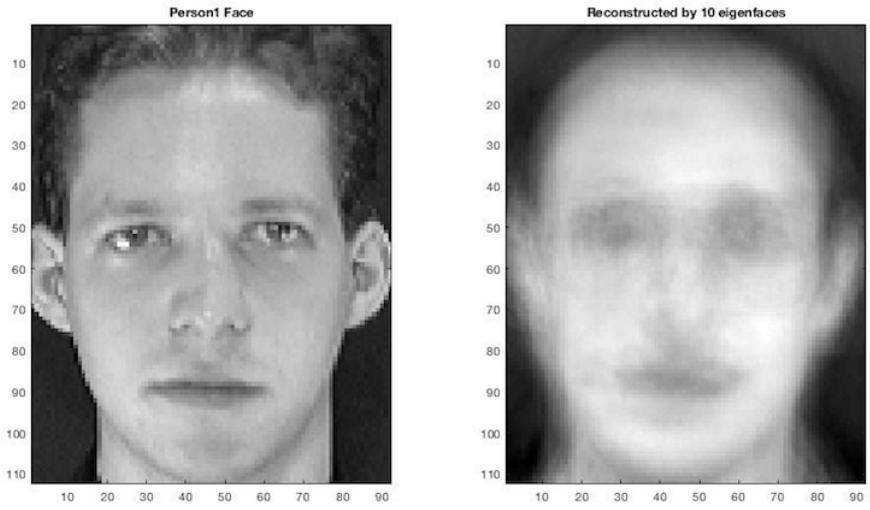

Figure 4: The original face and its projection onto the face space defined by Figure 3 .

To classify a new image (x'), we use the same projection $w' = P^T x'$ to get the set of weights representing the new incoming face image. This vector w' can in turn be used in a

standard classification, or pattern recognition, algorithm to determine which of a number of predefined face classes best describes the face.

Here, we use a simple 1-nearest-neighbor (1nn) classifier to predicts the class of the new face image. 1nn classifier assigns, to the new image, the class of its nearest neighbor. The distance metric is defined as Euclidean distance

$$d(x', X_j) = \|P^T x' - P^T X_j\|_2$$

and the nearest neighbor of the new image (x') is found by

$$\arg\min_j d(x', X_j)$$

After finding an existing image that minimizes the distance to the new image, we say both faces belong to the same person.

3 Results

We evaluated the performance of our classifer using 10-fold cross validation on the true positive rate of this classifier, i.e. if the predicted class of a face in the test set matches its predefined label. We also varied the amount of variance to keep to find the best number of eigenfaces we want to use for this classifier.

VAR Kept	Number of Eigenvectors needed (Dim)	TPR
0.5	2	37.75%
0.6	4	68.25%
0.7	8	91.00%
0.8	21	96.25%
0.9	69	92.75%
1	399	2.5%

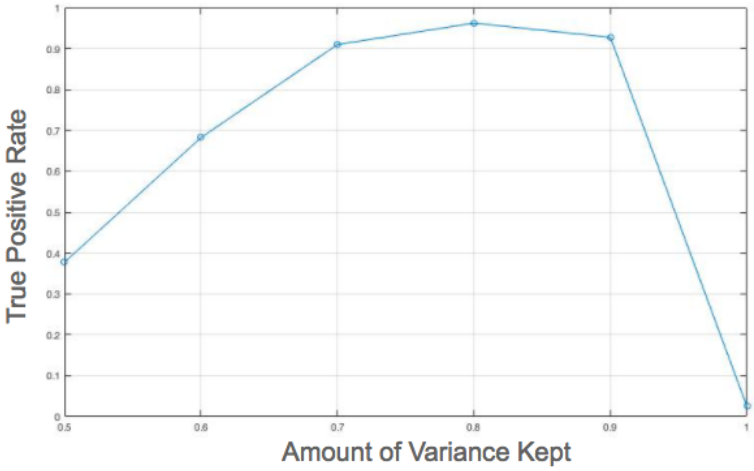

Figure 5: True Positive Rate evaluated using 10-fold cross validation

From the above table or figure, one can see that the true positive rate increases in general as the amount of variance (and the number of eigenvectors) we are keeping. This makes sense because we are capture more and more information of the data (at the expense of more computation time and memory). However, the increasing rate gradually decreases as the marginal effect of the variation/information we are getting by keeping more eigenvectors declines as the amount of information we already have. The extreme case happens when we keep all the variance (VAR Kept = 1). Note that the true positive rate decreases dramatically to 2.5% which precisely illustrates our previous point about the large amount of noise presented in a high-dimensional dataset. By keeping nearly all the variance, we would inevitably include the dimensions that are basically random noise and not so meaningful for making a classification decision. Therefore, applying a classifier on such a dataset would not generally give us meaningful results as shown in our case.

4 Discussion

In the future, we may want to implement other more sophisticated classifiers like a k-NN model instead of 1-NN model to improve the accuracy rate. We may also want to implement a faster and numerically stable SVD algorithm instead of using the MATLAB default option. For example, using a Householder triangulation and then a QR iteration may achieve a better numerical stability. We may also want to gather more data to increase the applicability of our facial recognition program. However, there is not so much improvement one can do using a simple PCA method.

PCA is a general method to capture the main variations in a dataset. It is a global method in the sense that it doesn't specifically weight any local patterns. In order for it to work well, the data must exhibit certain independence property across the dimensions, which may or may not be the case in the real world. If the data generating process is nonlinear, then it's unlikely that such independence assumption could be met. In particular, the application of image recognition and specifically face recognition, local patterns and local similarity plays a huge role. In most applications, we are not interested in identifying the face with one certain angle, we are rather interested in classifying faces from various possible

angles. When the image rotates, the local features does not change much, but the global features change drastically. PCA would not be able to capture such change well, and thus it would not be able to make use of large amount of local information.

In the domain of image recognition, convolutional neural networks (CNN) rely on the local features of a image, processing the image a few pixels a step, and through thousands of layers, in order to grasp all the details of the images. On the current frontier of image recognition, majority of classification models is based on CNN models. However, PCA has its advantage in the simplicity of implementation and the ease to understand and apply without much efforts on tuning any hyper-parameters.

5 Appendix

All the Matlab code files can be accessed here: `https://github.com/yinghj/FaceRecognitionPCA`

```
% load_faces.m
function dataset = load_faces(input_folder, ...
    sample_cnt, face_cnt, pixel_cnt)

dataset = zeros(pixel_cnt, sample_cnt*face_cnt);
i = 1;
for f_ind = 1:sample_cnt
    subfolder = strcat(input_folder, '/s', num2str(f_ind));
    for img_ind = 1:face_cnt
        % Import a pmg image.
        path = strcat(subfolder, '/', num2str(img_ind), '.pgm');
        pic = imread(path);
        % Convert the image from integer entries to doubles
        % so we can do linear algebra on it.
        graypic = im2double(pic);
        % Convert a pixel matrix to a column vector
        img_vec = reshape(graypic, [], 1);
        % Add image vector to matrix
        dataset(:, i) = img_vec;
        i = i + 1;
    end
end

%eigenfaces_pca2.m
% Load faces
IN_FOLDER = 'orl_faces_att';
SAMPLE_CNT = 40;
```

```
FACE_CNT = 10;
h = 112; w = 92;

faces = load_faces(IN_FOLDER, SAMPLE_CNT, FACE_CNT, h*w);
[pixels, numFaces] = size(faces);
[signal, PC, V] = pca(faces);
% Pull out eigen values and vectors
eigVals = V;
eigVecs = PC;

% Plot eigenvalues
figure; plot(log(eigVals));
title('Eigenvalues (log) in descending order');
disp('Press any key to display mean face.')
pause;
meanFace = mean(faces, 2);

% Plot the mean sample and
% the first 'EIGENSTOSHOW' number of principal components
figure; imshow(imresize(reshape(meanFace, h, w),3)); title('Mean Face');
disp('Press any key to display eigenfaces.')
pause;

% Define the number of eigenfaces to display
EIGENSTOSHOW = 10;
figure;
for i = 0:EIGENSTOSHOW/2-1
    subplot(2, EIGENSTOSHOW/2, mod(i, EIGENSTOSHOW/2)+1);
    imagesc(reshape(eigVecs(:, i+1), h, w)); colormap(gray);
    title(strcat('No. ', string(i+1),' Eigenface'));
```

```
    subplot(2, EIGENSTOSHOW/2, mod(i, EIGENSTOSHOW/2)+1+EIGENSTOSHOW/2);
    imagesc(reshape(eigVecs(:, i+1+EIGENSTOSHOW/2), h, w)); colormap(gray);
    title(strcat('No. ', string(i+1+EIGENSTOSHOW/2),' Eigenface'));
end

% Define a better higher-rank (still low-rank compare to full) approx.
% The cumulative energy content for the m'th eigenvector
% is the sum of the energy content across eigenvalues 1:m
for i = 1:numFaces
    energy(i) = sum(eigVals(1:i));
end
propEnergy = energy./energy(end);
% Determine the number of principal components
% required to model 90% of data variance
percentMark = find(propEnergy > 0.9, 1 );

% Pick those principal components
eigenVecs = eigVecs(:, 1:percentMark);
eigenVecs_low = eigVecs(:, 1:EIGENSTOSHOW);

% Project each faces onto the corresponding eigenfaces
% to reconstruct the faces in lower dimension.
person = zeros(pixels, SAMPLE_CNT);
personW = zeros(percentMark, 1);
personW_low = zeros(EIGENSTOSHOW, 1);

for i=1:SAMPLE_CNT
    person(:, i) = faces(:, (i-1)*10+1);
    personW(:, i) = eigenVecs\person(:, i);
```

```
    personW_low(:, i) = eigenVecs_low\person(:, i);
end
disp('Press any key to reconstruct faces.')
pause;
for i=1:SAMPLE_CNT
    figure;
    subplot(1, 3, 1); imagesc(reshape(person(:, i), h, w));
    colormap(gray);
    title(strcat('Person ', string(i),' Face'));
    recon_low_ = eigenVecs_low*personW_low(:, i);
    subplot(1, 3, 2); imagesc(reshape(recon_low_(:,1), h, w));
    colormap(gray);
    title('Reconstructed by 10 eigenfaces');
    recon_ = eigenVecs*personW(:, i);
    subplot(1, 3, 3); imagesc(reshape(recon_(:,1), h, w));
    colormap(gray);
    title('Reconstructed 90% Variance');
    pause;
end


%eigenfaces_predict.m
function C = eigenfaces_predict(Xtrain, eigenfaces, mean, ...
    Xtest, Ytrain, k)
%% Predicts nearest neighbor for given Eigenfaces model.
%% Args:
%% Xtrain [pixel x num_faces] training matrix that produces the
    %%  eigenfaces and reference for classification
    %%  eigenfaces [pixel x dim] basis of the new face space
    %%  mean [pixel x 1] the mean face of all training data
%% Xtest [dim x 1] test vector to predict
```

```
    %%  numClasses [int] the number of faces you want to recognize
    %%  k [int] k in knn
test_proj = eigenfaces \ (Xtest - mean);
    train_proj = eigenfaces \ (Xtrain - repmat(mean, 1, size(Xtrain,2)));
C = knn(train_proj, Ytrain, test_proj, k);
end
```

6 References

- Dataset: AT&T "The Database of Faces" (formerly "The ORL Database of Faces")
- F. Samaria and A. Harter "Parameterisation of a stochastic model for human face identification" 2nd IEEE Workshop on Applications of Computer Vision December 1994, Sarasota (Florida).
- Eigenfaces for recognition. M. Turk, A. Pentland, J Cogn Neurosci. 1991 Winter; 3(1): 7186. doi: 10.1162/jocn.1991.3.1.71
- Two-dimensional PCA: a new approach to appearance-based face representation and recognition. J. Yang. 2004; IEEE Volume: 26 Issue: 1.
- A tutorial on principal component analysis. J. Shlens. 2014; arXiv: 1404.1100.