EE 779 Advanced topics in Signal Processing

Project on Principal Component Analysis (PCA)

Submitted by: **Swrangsar Basumatary** Roll: **09d07040**

Problem 3(a)

We take an original image from the training set (orl-faces/s1/1.pgm) and project to the subspace spanned by eigenfaces. We then find a weight vector containing the weight of each eigenface in the original image. An estimate of the original image is reconstructed by summing up all the weighted eigenfaces(the weights from the weight vector).

The mean square error of this estimated image w.r.t the original is also calculated.

orlImgTrain

orlImgTrainEst

trainMSE = 0.529503153775707

Problem 3(b)

orlImgTest

orlImgTest = testDATA(:, 13);

orlImgTestEst

testMSE = 330.089201136936

Problem 3(c)

myImg

myImgEst

myImgMSE = 1053.22541281570

Time taken by PCA with subdimension "200" is 6.283311 seconds.

Comments

Since the original image from the training set is already in the training set, so it has the least MSE. Next the test image is not in the training set but we have other images of that person's face in the training set. So the eigenfaces have some information about the test image and it is not an unfamiliar face. But no image of mine was in the training set, so being a previously unencountered face it gave the highest MSE.

The code:

```
close all; clear all;

cd ~/Desktop/pcaProject/
```

```
tic
pca_orl(200);
toc

plotPath = './results/';
imgFormat = '-dtiffn';

testDATA = orldata_test; % Get test images from orldata

%% Case 1: Test using trained image
% Fill to complete 3.(a)
% Use one image from the training data set
load DATA;
orlImgTrain = DATA(:, 3); % fill
figure; imshow( reshape(orlImgTrain, 112, 92), []);
print(imgFormat, [plotPath 'orlImgTrainS']);

% % Reconstruct the above image using the PCs
load psi;
zeroMeanImage = orlImgTrain - psi;

load w;
weightVector = w' * zeroMeanImage;

orlImgTrainEst = zeros(size(w(:, 1)));
for k = 1:length(weightVector)
    orlImgTrainEst = orlImgTrainEst + (weightVector(k) * w(:,
k));
end

orlImgTrainEst = orlImgTrainEst + psi;
figure; imshow( reshape(orlImgTrainEst, 112, 92), []);
print(imgFormat, [plotPath 'orlImgTrainEstS']);
clear DATA; clear psi; clear w;

img1_err = norm(orlImgTrain - orlImgTrainEst); % fill
trainMSE = (img1_err * img1_err)/length(orlImgTrainEst(:));
save trainMSE trainMSE;



%% Case 2: Test using a test image from orl data base



orlImgTest = testDATA(:, 13);
```

```
% Fill to complete 3.(b)
figure; imshow( reshape(orlImgTest, 112, 92), []);
print(imgFormat, [plotPath 'orlImgTestS']);

% % Reconstruct the above image using the PCs
load psi;
zeroMeanImage = orlImgTest - psi;

load w;
weightVector = w' * zeroMeanImage;


orlImgTestEst = zeros(size(w(:, 1)));
for k = 1:length(weightVector)
    orlImgTestEst = orlImgTestEst + (weightVector(k) * w(:, k));
end

orlImgTestEst = orlImgTestEst + psi;
figure; imshow( reshape(orlImgTestEst, 112, 92), []);
print(imgFormat, [plotPath 'orlImgTestEstS']);
clear testDATA; clear psi; clear w;

imgTestError = norm(orlImgTest - orlImgTestEst); % fill
testMSE = (imgTestError * imgTestError)/
length(orlImgTestEst(:));
save testMSE testMSE;




%% Case 3: Test using your face image
% Read in your image
% Fill to complete 3.(c)

load myImg
myImg = double(myImg(:));
figure; imshow( reshape(myImg, 112, 92), []);
print(imgFormat, [plotPath 'myImgS']);

% % Reconstruct the above image using the PCs
load psi;
zeroMeanImage = myImg - psi;

load w;
weightVector = w' * zeroMeanImage;
```

```
myImgEst = zeros(size(w(:, 1)));
for k = 1:length(weightVector)
    myImgEst = myImgEst + (weightVector(k) * w(:, k));
end

myImgEst = myImgEst + psi;
figure; imshow( reshape(myImgEst, 112, 92), []);
print(imgFormat, [plotPath 'myImgEstS']);
clear psi; clear w;

myImgError = norm(myImg - myImgEst); % fill
myImgMSE = (myImgError * myImgError)/length(myImg);
save myImgMSE myImgMSE;

% Generate plots to show original and reconstructed images
```

Problem 4

Here we retain only 100 eigenfaces with the highest eigenvalues out of the 200 eigenfaces that can be built from the 200 training images.

orlImgTrain100

orlImgTrainEst100

trainMSE100 = 81.3904522066124

orlImgTest100

orlImgTestEst100

testMSE100 = 371.063908820780

myImg100

myImgEst100

myImgMSE100 = 1240.44576637774

Time taken by PCA with subdimension “100” is 5.990830 seconds.

Comments

Here the MSE’s are larger than the ones in Problem (3) because we are using less number of eigenfaces to represent them. We discarded the eigenfaces with relatively lower eigenvalues and thus we are losing some infrormation here. We also have less degree of freedom to represent face images when we use less number of eigenfaces. The estimates are less close to the original. But lesser number of eigenfaces makes the PCA implementation faster because we have less number of weights for eigenfaces to calculate.

The code:

```
close all; clear all;
```

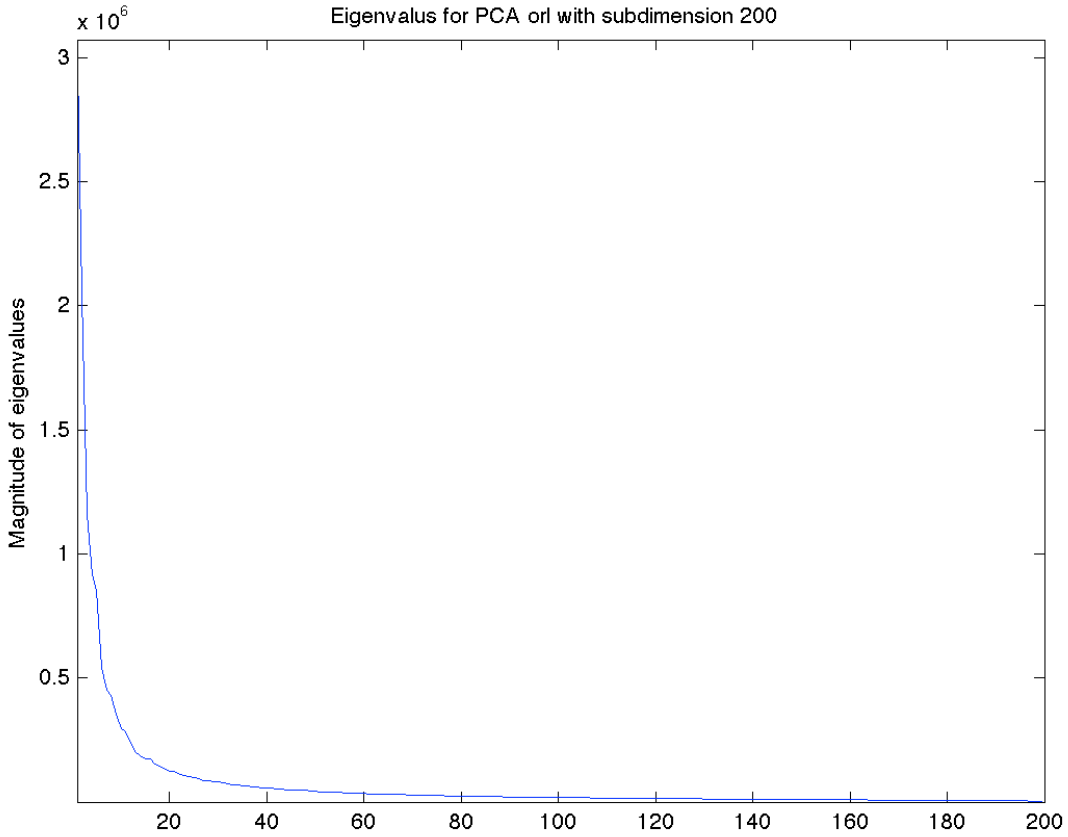
```
cd ~/Desktop/pcaProject/

tic
pca_orl(100);
toc

plotPath = './results/subDim100/';

testDATA = orldata_test; % Get test images from orldata

%% Case 1: Test using trained image
% Fill to complete 3.(a)
% Use one image from the training data set
load DATA;
orlImgTrain = DATA(:, 3); % fill
figure; imshow( reshape(orlImgTrain, 112, 92), []);
print('-dtiffn', [plotPath 'orlImgTrain100']);

% % Reconstruct the above image using the PCs
load psi;
zeroMeanImage = orlImgTrain - psi;

load w;
weightVector = w' * zeroMeanImage;

orlImgTrainEst = zeros(size(w(:, 1)));
for k = 1:length(weightVector)
    orlImgTrainEst = orlImgTrainEst + (weightVector(k) * w(:,
k));
end

orlImgTrainEst = orlImgTrainEst + psi;
figure; imshow( reshape(orlImgTrainEst, 112, 92), []);
print('-dtiffn', [plotPath 'orlImgTrainEst100']);
clear DATA; clear psi; clear w;

img1_err = norm(orlImgTrain - orlImgTrainEst); % fill
trainMSE100 = (img1_err * img1_err)/length(orlImgTrainEst(:));
save trainMSE100 trainMSE100;



%% Case 2: Test using a test image from orl data base
```

```
orlImgTest = testDATA(:, 13);
% Fill to complete 3.(b)
figure; imshow( reshape(orlImgTest, 112, 92), []);
print('-dtiffn', [plotPath 'orlImgTest100']);

% % Reconstruct the above image using the PCs
load psi;
zeroMeanImage = orlImgTest - psi;

load w;
weightVector = w' * zeroMeanImage;


orlImgTestEst = zeros(size(w(:, 1)));
for k = 1:length(weightVector)
    orlImgTestEst = orlImgTestEst + (weightVector(k) * w(:, k));
end

orlImgTestEst = orlImgTestEst + psi;
figure; imshow( reshape(orlImgTestEst, 112, 92), []);
print('-dtiffn', [plotPath 'orlImgTestEst100']);
clear testDATA; clear psi; clear w;

imgTestError = norm(orlImgTest - orlImgTestEst); % fill
testMSE100 = (imgTestError * imgTestError)/
length(orlImgTestEst(:));
save testMSE100 testMSE100;




%% Case 3: Test using your face image
% Read in your image
% Fill to complete 3.(c)

load myImg
myImg = double(myImg(:));
figure; imshow( reshape(myImg, 112, 92), []);
print('-dtiffn', [plotPath 'myImg100']);

% % Reconstruct the above image using the PCs
load psi;
zeroMeanImage = myImg - psi;

load w;
weightVector = w' * zeroMeanImage;
```

```
myImgEst = zeros(size(w(:, 1)));
for k = 1:length(weightVector)
    myImgEst = myImgEst + (weightVector(k) * w(:, k));
end

myImgEst = myImgEst + psi;
figure; imshow( reshape(myImgEst, 112, 92), []);
print('-dtiffn', [plotPath 'myImgEst100']);
clear psi; clear w;

myImgError = norm(myImg - myImgEst); % fill
myImgMSE100 = (myImgError * myImgError)/length(myImg);
save myImgMSE100 myImgMSE100;



% Generate plots to show original and reconstructed images
```

Problem 5

Plotting eigenvalues in descending order

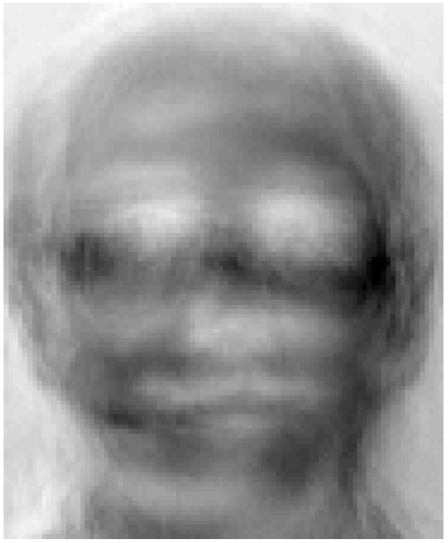


The first seven eigenfaces with decreasing order of eigenvalues

Eigenface 1

Eigenface 2

Eigenface 3



Eigenface 4

Eigenface 5

Eigenface 6

Eigenface 7

The script:

```
close all; clear all;

cd ~/Desktop/pcaProject/

tic
pca_orl(200);
toc

plotPath = './results/eigenFaces/';
imgFormat = '-dtiffn';



%% plotting eigenvalues
load pcaEigVals;
figure;
plot(pcaEigVals); axis tight;
title('Eigenvalus for PCA orl with subdimension 200');
```

```
ylabel('Magnitude of eigenvalues');
print(imgFormat, [plotPath 'eigenValues']);
clear pcaEigVals;


%% plotting eigenfaces

load w;
% plotting first 5 eigenfaces
numberOfEigenfaces = 7;

for k = 1:numberOfEigenfaces
    figure; imshow( reshape(w(:,k), 112, 92), []);
    print(imgFormat, [plotPath 'eigenface' num2str(k)]);
end
clear w;
```

Problem 6

Here we do the same things as in Problem (3) and (4) but using a custom PCA code. The non-linear iterative partial least squares (NIPALS) algorithm was used for calculating the eigenfaces.

Using subdimension = 100

modImgTrain100

modImgTrainEst100

trainModMSE100 = 81.3904522085586

modImgTest100

modImgTestEst100

testModMSE100 = 371.063908820763

modmyImg100

modmyImgEst100

myImgModMSE100 = 1240.44576637806

Time taken by ‘pcaModified ‘ function is 1734.462743 seconds.

Comments:

The iterative method of calculating eigenfaces (by PCA) is very very slow with respect to the ‘pca_orl’ function. But we have almost exact values of image estimate MSE’s for the same subdimensions whether we use this iterative and slow method or the faster ‘pca_orl’ method.

We cannot use the ordinary covariance matrix method or the SVD method for the PCA here. Because our XX’ turns out to be a 10304 x 10304 matrix(a large matrix) which is computationally intensive and stalls while implementing (because it takes a very long time) directly in a normal desktop system. So we have to use the technique used in ‘pca_orl’ function or go for an iterative method. Iterative methods are slow w.r.t the ‘pca_orl’ function. I even tried ‘randomized SVD’ method but MATLAB stalled with it also.

Now using sub-dimension = 50

modImgTrain50

modImgTrainEst50

trainModMSE50 = 252.546404732910

modImgTest50

modImgTestEst50

testModMSE50 = 429.548655119170

modmyImg50

modmyImgEst50

myImgModMSE50 = 1439.68664947852

Time taken by 'pcaModified()' for subDim = 50 is 649.330239 seconds.

Comments:

As expected with sub-dimension = 50 (we used 100 in the previous run) total time taken by the PCA function with NIPALS algorithm is lower than before. And we have higher MSE's because we have less number of eigenfaces to represent the faces.

The 'pca_orl' first calculates the eigenvectors of M x M matrix where M << N^2 and then forms the eigenfaces from them using the M available training set images. On the other hand, the iterative methods like NIPALS method use a lot of cycles to estimate a single eigenface. They keep iterating until some required variables converge and thus these iterations add to the overall time taken by the algorithms.

The functions:

```
close all; clear all;

cd ~/Desktop/pcaProject/

tic
pcaModified(50);
toc

plotPath = './resultsModified/';
imgFormat = '-dtiffn';

testDATA = orldata_test; % Get test images from orldata

%% Case 1: Test using trained image
% Fill to complete 3.(a)
% Use one image from the training data set
load DATA;
modImgTrain = DATA(:, 3); % fill
figure; imshow( reshape(modImgTrain, 112, 92), []);
print(imgFormat, [plotPath 'modImgTrainS']);

% % Reconstruct the above image using the PCs
load psi;
zeroMeanImage = modImgTrain - psi;

load eigenFaces;
weightVector = eigenFaces' * zeroMeanImage;

modImgTrainEst = zeros(size(eigenFaces(:, 1)));
for k = 1:length(weightVector)
    modImgTrainEst = modImgTrainEst + (weightVector(k) *
eigenFaces(:, k));
end

modImgTrainEst = modImgTrainEst + psi;
figure; imshow( reshape(modImgTrainEst, 112, 92), []);
print(imgFormat, [plotPath 'modImgTrainEstS']);
clear DATA; clear psi; clear eigenFaces;

img1_err = norm(modImgTrain - modImgTrainEst); % fill
trainModMSE = (img1_err * img1_err)/length(modImgTrainEst(:));
save trainModMSE trainModMSE;



%% Case 2: Test using a test image from orl data base
```

```
modImgTest = testDATA(:, 13);
% Fill to complete 3.(b)
figure; imshow( reshape(modImgTest, 112, 92), []);
print(imgFormat, [plotPath 'modImgTestS']);

% % Reconstruct the above image using the PCs
load psi;
zeroMeanImage = modImgTest - psi;

load eigenFaces;
weightVector = eigenFaces' * zeroMeanImage;


modImgTestEst = zeros(size(eigenFaces(:, 1)));
for k = 1:length(weightVector)
    modImgTestEst = modImgTestEst + (weightVector(k) *
eigenFaces(:, k));
end

modImgTestEst = modImgTestEst + psi;
figure; imshow( reshape(modImgTestEst, 112, 92), []);
print(imgFormat, [plotPath 'modImgTestEstS']);
clear testDATA; clear psi; clear eigenFaces;

imgTestError = norm(modImgTest - modImgTestEst); % fill
testModMSE = (imgTestError * imgTestError)/
length(modImgTestEst(:));
save testModMSE testModMSE;




%% Case 3: Test using your face image
% Read in your image
% Fill to complete 3.(c)

load myImg
modmyImg = double(myImg(:));
figure; imshow( reshape(modmyImg, 112, 92), []);
print(imgFormat, [plotPath 'modmyImgS']);

% % Reconstruct the above image using the PCs
load psi;
```

```
zeroMeanImage = modmyImg - psi;

load eigenFaces;
weightVector = eigenFaces' * zeroMeanImage;


modmyImgEst = zeros(size(eigenFaces(:, 1)));
for k = 1:length(weightVector)
    modmyImgEst = modmyImgEst + (weightVector(k) * eigenFaces(:,
k));
end

modmyImgEst = modmyImgEst + psi;
figure; imshow( reshape(modmyImgEst, 112, 92), []);
print(imgFormat, [plotPath 'modmyImgEstS']);
clear psi; clear eigenFaces;

myImgError = norm(modmyImg - modmyImgEst); % fill
myImgModMSE = (myImgError * myImgError)/length(modmyImg);
save myImgModMSE myImgModMSE;

% Generate plots to show original and reconstructed images


function pcaModified(subDim)

% using orldata_train code
%
% RELATED FUNCTIONS (SEE ALSO)
% pgma_read, orldata, orldata_test, orldata_train
%
% ABOUT
% Created:        03 Sep 2005
% Last Update:    -
% Revision:       1.0
%
% AUTHOR:   Kresimir Delac
% mailto:   kdelac@ieee.org
% URL:      http://www.vcl.fer.hr/kdelac
%
% WHEN PUBLISHING A PAPER AS A RESULT OF RESEARCH CONDUCTED BY
USING THIS CODE
% OR ANY PART OF IT, MAKE A REFERENCE TO THE FOLLOWING PAPER:
% Delac K., Grgic M., Grgic S., Independent Comparative Study of
PCA, ICA, and LDA
```

```
% on the FERET Data Set, International Journal of Imaging
Systems and Technology,
% Vol. 15, Issue 5, 2006, pp. 252-260
%

global imloadfunc;

imloadfunc =  'pgma_read';
disp(' ');
disp('Running PCA modified by Swrangsar Basumatary')

DATA = orldata_train;
save DATA DATA;
[~, numberOfImages] = size(DATA);
imageSpace = DATA;
save imageSpace imageSpace;
%clear DATA;

% Calculating mean face from training images
fprintf('Zero mean\n')
psi = mean(imageSpace, 2);
disp('psi calculated')
save psi psi;

dim = numberOfImages;
zeroMeanSpace = zeros(size(imageSpace));
for k = 1:dim
    zeroMeanSpace(:, k) = double(imageSpace(:, k)) - psi;
end
save zeroMeanSpace zeroMeanSpace;
clear imageSpace;
disp('zeroMeanSpace calculated')

disp('Control handed over to iterative PCA')
% randomizedSVD(zeroMeanSpace, subDim);
iterativePCA(zeroMeanSpace, subDim);
clear zeroMeanSpace;


end

function iterativePCA(data, subDim)

eigenFaces = getPrincipalComponentMatrix(data, subDim);
save eigenFaces eigenFaces;
```

```
% eigenValues = getEigenValueVector(data, eigenFaces);
% save eigenValues eigenValues;

end


%% get a principal component and the data for the next principal
component
% calculation
function [principalComponent, nextData] =
getPrincipalComponent(data, iterationNumber)

% generate random vector
% [imageSize, dim] = size(data);
data = data';
scoreVector = data(:, randi(size(data,2)));
prevScoreVector = scoreVector;
difference = 1;

while difference > 1.0e-8
    loadingVector = (data' * scoreVector) / (scoreVector' *
scoreVector);
    loadingVector = loadingVector / norm(loadingVector);
    prevScoreVector = scoreVector;
    scoreVector = (data * loadingVector) / (loadingVector' *
loadingVector);
    difference = norm(prevScoreVector - scoreVector);
    principalComponent = loadingVector;

    disp(['difference: ' num2str(iterationNumber) ' : '
num2str(difference)]);
end
disp('got a principal component...')

nextData = data - (scoreVector * principalComponent');
nextData = nextData';
principalComponent = principalComponent(:);

end



%% get principal component matrix

function principalComponentMatrix =
getPrincipalComponentMatrix(data, subDim)
```

```
imageSize = size(data, 1);

principalComponentMatrix = zeros(imageSize, subDim);
nextData = data;
for k = 1:subDim
    [principalComponent, nextData] =
getPrincipalComponent(nextData, k);
    principalComponentMatrix(:, k) = principalComponent;
end

end



% %% get eigenvalue for a principal component
%
% function eigenValue = getEigenValue(data, principalComponent)
%
% dim = size(data, 2);
% % before subtracting principal component
% dataMean = mean(data, 2);
% dataDistance = data - repmat(dataMean, 1, dim);
% dataDistanceSquared = dataDistance .* dataDistance;
% dataDistanceSquaredSum = sum(dataDistanceSquared(:));
% % after subtracting principal component
% newData = data - repmat(principalComponent, 1, dim);
% newDataMean = mean(newData, 2);
% newDataDistance = newData - repmat(newDataMean, 1, dim);
% newDataDistanceSquared = newDataDistance .* newDataDistance;
% newDataDistanceSquaredSum = sum(newDataDistanceSquared(:));
%
% eigenValue = abs(dataDistanceSquaredSum -
newDataDistanceSquaredSum);
%
% end
%
%
%
% %% get eigenvalue matrix
% function eigenValueVector = getEigenValueVector(data,
principalComponentMatrix)
%
% subDim = size(principalComponentMatrix, 2);
% eigenValueVector = zeros(subDim, 1);
%
```

```
% for k = 1:subDim
%     eigenValueVector(k) = getEigenValue(data,
principalComponentMatrix(:, k));
% end
%
% end
```