CSCC11 Fall 2015
Assignment 4

Student Name: Maoting Zhang
Student Number: 999590633
UTORid(mathlab username): zhangmao

- Print-outs

For full-dimensional KNN classifier:
Correct classification rate:
correctEye_knn =
0.9778
False positive rate:
falseEye_knn =
0.1125

For PCAcomp = 5,10,15,25, and 50:
PCA correct classification rate:
0.9051 0.9515 0.9511 0.9666 0.9824
PCA False positive rate:
0.0667 0.1401 0.1240 0.1286 0.1292
low-dimensional kNN correct classification rate:
0.9498 0.9758 0.9791 0.9804 0.9804
low-dimensional kNN False positive rate:
0.1281 0.0901 0.0880 0.0885 0.0932

- Discussion:

  - KNN low-dimensional classifier (when PCAcomp = 10)

- Compare with high-dimensional data, are the results better? worse? is this what you expected?
high-dimensional data: correctEye_knn = 0.9778; falseEye_knn = 0.1125
low-dimensional classifier: correctEye_knn = 0.9791; falseEye_knn = 0.0880
The results are better as expected, since we can see that low-dim data has higher correct classification rate and lower false positive rate than that of high-dim data.

- why do you think the results are like this?
This is because after we reduced the dimensions of the data, thus the classification process focused more on each data image as a whole, rather than a distributive set of points consisting an image.

- Is there a value for PCAcomps (or set of values) for which low-dimensional kNN is better than full dimensional kNN?
Yes when PCAcomps is 10, 25 and/or 50, the low-dimensional kNN performs better.

- why do you think that is?
When PCAcomps is too small, the original data is over-reduced and the subspace might not projects the original data as precise; yet a proper PCAcomps is able to capture all of the structure of the data and in some cases rebalances the weights of the data to give a better performance while improving the efficiency in the meantime.

- In General

- Which classifier gives the overall best performance?
The low-dimensional kNN classifier gives the overall best performance. As from figure 3 and 4, it has lower false positive rates and higher correct classification rates in general.

- What conclusions can you draw about the usefulness of dimensionality reduction?
Dimensionality reduction constrains the estimated values of data to lie in a low-dimensional subspace of the original data, which helped emphasizing the role that data plays as an entity, and reduced the effects of noises in the data as when the data item is a higher-dimension set consisting of potentially distributive data points.

- Which classifier would you use on a large training set consisting of high-dimensional data?
I will use the low-dimensional kNN classifier on large high-dimensional data, reducing the dimensionality improves the efficiency of the estimation without affecting the estimated outcome (and might even improve the correctness rate).

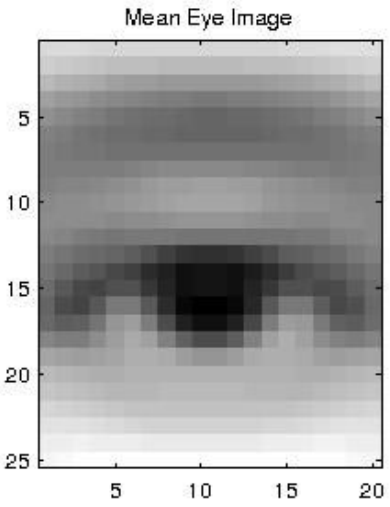
- Which classifier would you use on a large training set of low-dimensional (e.g. 3-D) data? why?
I will use the full-dimensional kNN classifier on large low-dimensional data, because low dimensional data are self-explanatory enough and not very consuming to estimate, reducing the dimensions could result to worse estimation as lower dimension will be over-focus and intolerant to deviations from the subspace.

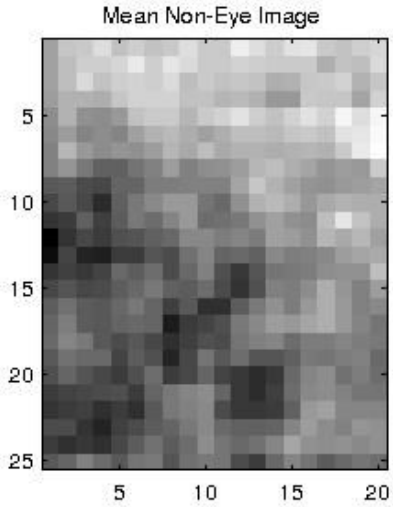
- Summarize the advantages/disadvantages of each classifier!

Classifier	Advantage	Disadvantage
Full-Dimensional KNN Classifier:	Precise when dimensional is low	might be slow
Low-Dimensional KNN Classifier:	Precise when dimensional is high; fast on large data sets	not very helpful when original data is low-dimensional
PCA-Construction-Based Classifier	fast and effective	might not be as accurate and precise

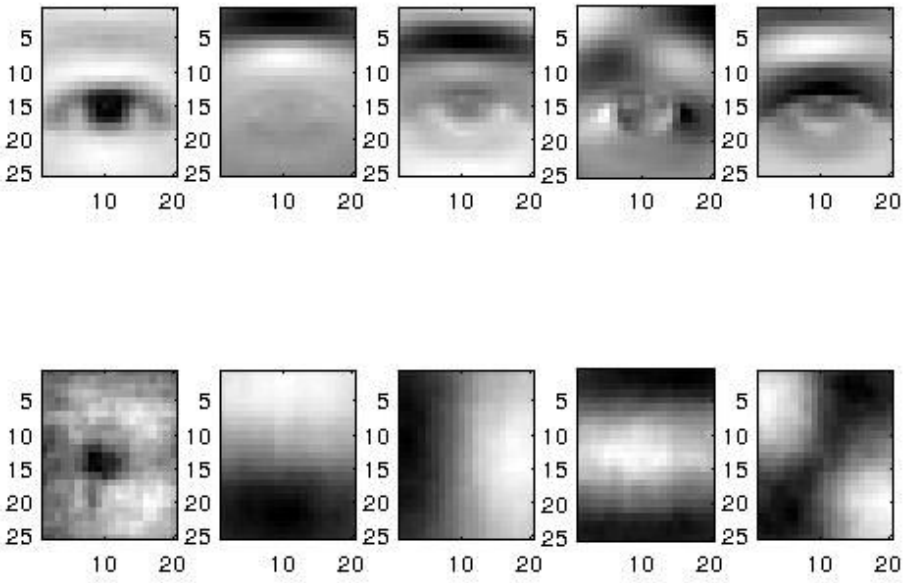
For Bonus classifier, please see the later submission on Dec 4th. :)

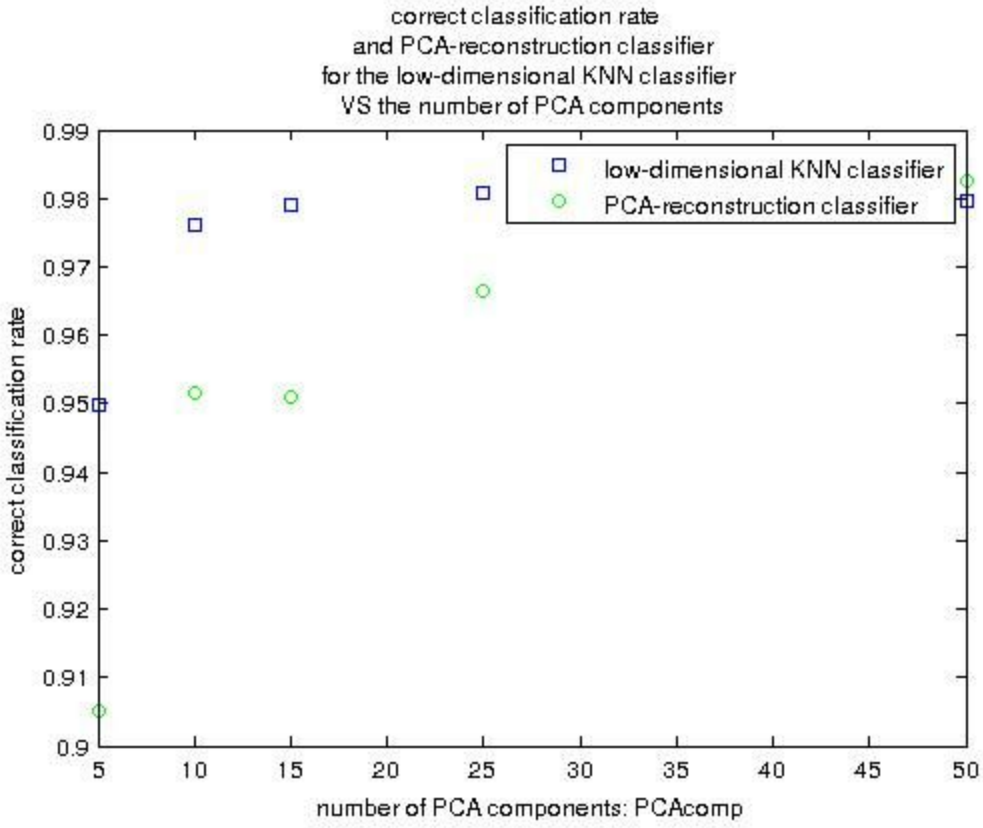
- Plots

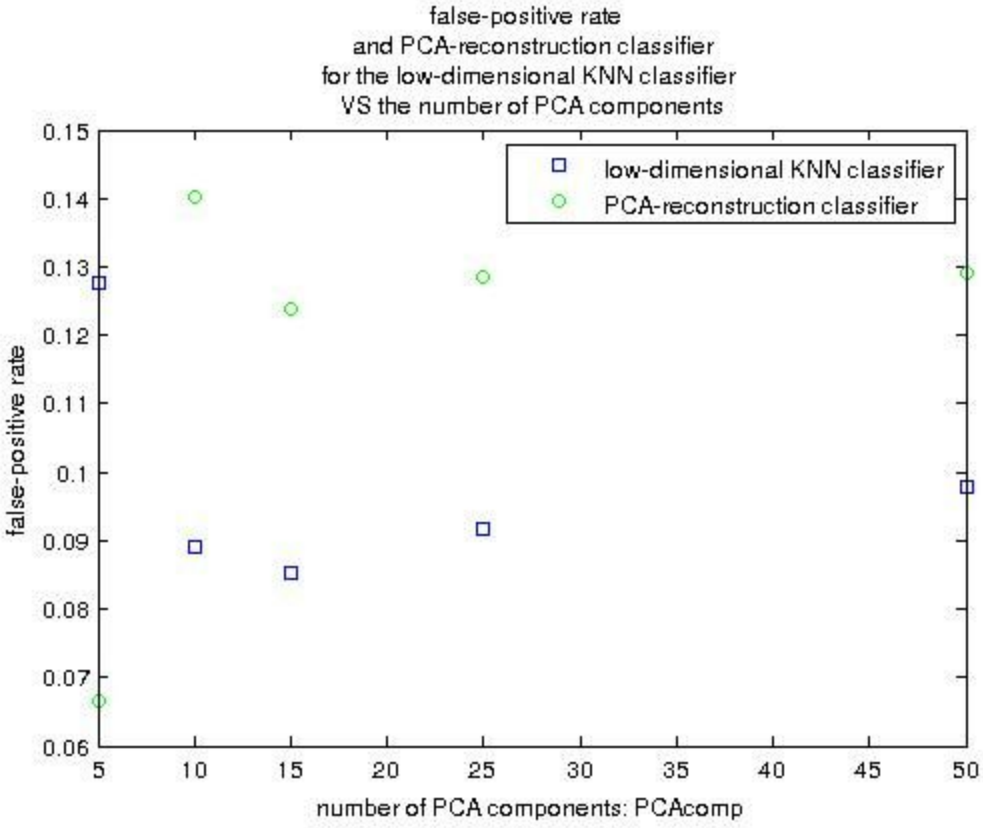


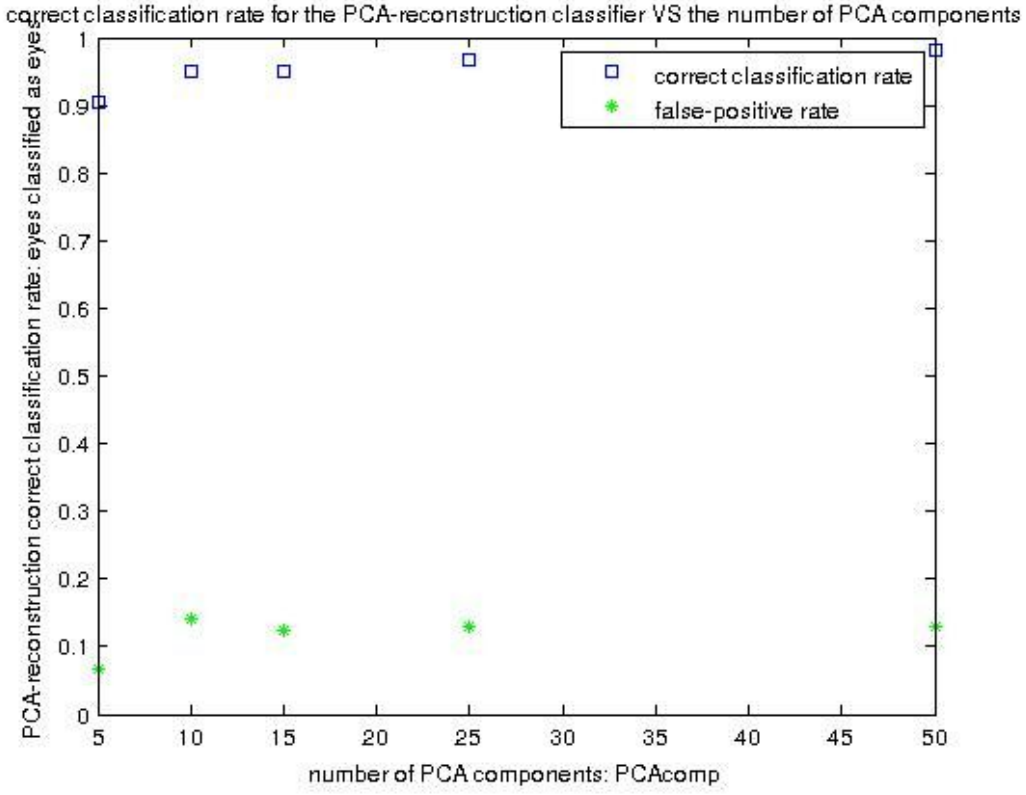


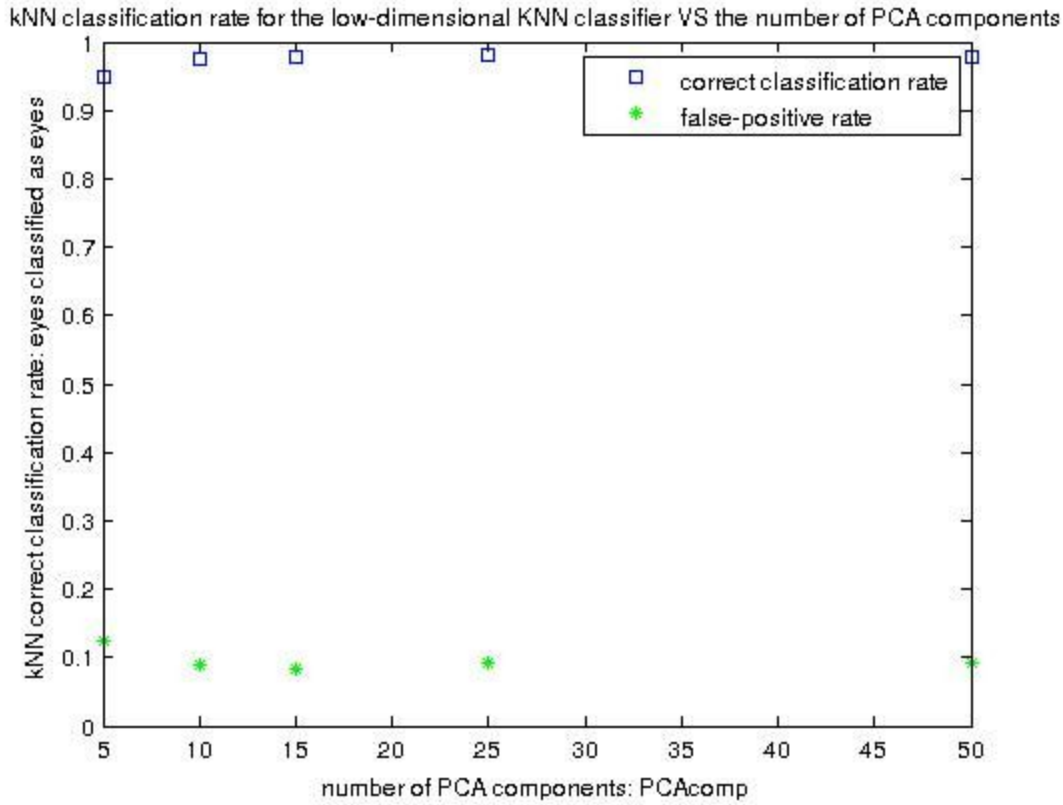
First 5 eigenvectors for eyes and non-eyes




correct classification rate
and PCA-reconstruction classifier
for the low-dimensional KNN classifier
VS the number of PCA components
low-dimensional KNN classifier
PCA-reconstruction classifier
correct classification rate
number of PCA components: PCAcomp
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.9
5
10
15
20
25
30
35
40
45
50


false-positive rate
and PCA-reconstruction classifier
for the low-dimensional KNN classifier
VS the number of PCA components
low-dimensional KNN classifier
PCA-reconstruction classifier
false-positive rate
number of PCA components: PCAcomp
0.15
0.14
0.13
0.12
0.11
0.1
0.09
0.08
0.07
0.06
5
10
15
20
25
30
35
40
45
50


correct classification rate for the PCA-reconstruction classifier VS the number of PCA components
PCA-reconstruction correct classification rate: eyes classified as eyes
number of PCA components: PCAcomp
correct classification rate
false-positive rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
5
10
15
20
25
30
35
40
45
50


kNN classification rate for the low-dimensional KNN classifier VS the number of PCA components
kNN correct classification rate: eyes classified as eyes
number of PCA components: PCAcomp
correct classification rate
false-positive rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
5
10
15
20
25
30
35
40
45
50

- Matlab Scripts & functions

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% PCA PART: Your task begins here!
% Do PCA on eyes and non-eyes to generate models for recognition

%%% TO DO:
% First, compute the mean over the eye and non-eye images
% i.e. compute the mean eye image, and the mean non-eye image
eyeMean = mean(eyeIm, 2);
noneyeMean = mean(nonIm, 2);

%%% TO PRINT:
% Plot the mean eye and mean non-eye images and hand the
% printouts in with your report.
figure(1);clf;
subplot(1, 2, 1);
imagesc(reshape(eyeMean,sizeIm));axis image;colormap(gray)
title('Mean Eye Image');

subplot(1, 2, 2);
imagesc(reshape(noneyeMean,sizeIm));axis image;colormap(gray)
title('Mean Non-Eye Image');

%%% TO DO:
% Now, perform the PCA computation as discussed in lecture over the
% training set. You will do this separately for eye images and non-eye
% images. This will produce a set of eigenvectors that represent eye
% images, and a different set of eigenvectors for non-eye images.

[eyeVec, eyeDiag] = computePCA(eyeIm, eyeMean);
[noneyeVec, noneyeDiag] = computePCA(nonIm, noneyeMean);

%%% TO PRINT:
% Display and print out the first 5 eigenvectors for eyes and non-eyes
% (i.e. the eigenvectors with LARGEST 5 eigenvalues, make sure you sort
% the eigenvectors by eigenvalues!)
% print (non)eye images!!!
% disp('first 5 eigenvectors for eyes')
% disp('first 5 eigenvectors for non-eyes')
figure(2);clf;
for e = 1:5

subplot(2, 5, e);
imagesc(reshape(eyeVec(:, e),sizeIm));axis image;colormap(gray)
hold on;
subplot(2, 5, e+5);
imagesc(reshape(noneyeVec(:, e),sizeIm));axis image;colormap(gray)

end
```

```
suptitle('First 5 eigenvectors for eyes and non-eyes');

%%% TO DO:
% Now you have two PCA models: one for eyes, and one for non-eyes.
% Next we will project our TEST data onto our eigenmodels to obtain
% a low-dimensional representation first we choose a number of PCA
% basis vectors (eigenvectors) to use:

% correct classification rates(ccr) and false positive rates(fpr) for PCA and KNN
ccrPCA = zeros(1, 5);
fprPCA = zeros(1, 5);
ccrKNN = zeros(1, 5);
fprKNN = zeros(1, 5);
n = 1;
for PCAcomp=[5,10,15,25,50]
    % To compute the low-dimensional representation for a given
    % entry in the test test, we must do 2 things. First, we subtract
    % the mean, and then we project that vector on the transpose of
    % the PCA basis vectors.
    %
    % You need to compute coefficients for BOTH the eye and non-eye
    % models for each testSet entry, i.e. for each testSet image you
    % will end up with (2*PCAcomp) coefficients which are the projection
    % of that test image onto the chosen eigenvectors for eyes and non-eyes.

    test_N = size(testSet, 1);
    coeffsEye = zeros(test_N, PCAcomp);
    coeffsNonEye = zeros(test_N, PCAcomp);
    for i=1:test_N
        vEye=testSet(i,:);
        coeffsEye(i, :)=eyeVec(:,1:PCAcomp)'*(vEye'-eyeMean);
        coeffsNonEye(i, :)=noneyeVec(:,1:PCAcomp)'*(vEye'-noneyeMean);
    end
    coeffs = [coeffsEye coeffsNonEye];

    %%% TO DO:
    % Then do the same for the training data.  That is, compute the
    % PCA coefficients for each training image using both of the models.
    % Then you will have low-dimensional test data and training data
    % ready for the application of KNN, just as we had in the KNN example
    % at the beginning of this script.
    train_N = size(trainSet, 1);
    coeffsEye_train = zeros(train_N, PCAcomp);
    coeffsNonEye_train = zeros(train_N, PCAcomp);
    for i=1:train_N
        vEye=trainSet(i,:);
        coeffsEye_train(i, :)=eyeVec(:,1:PCAcomp)'*(vEye'-eyeMean);
        coeffsNonEye_train(i, :)=noneyeVec(:,1:PCAcomp)'*(vEye'-noneyeMean);
    end
    coeffs_train = [coeffsEye_train coeffsNonEye_train];
```

```
    %%% TO DO
    % KNN classification:
    % Repeat the procedure at the beginning of this script, except
    % instead of using the original testSet data, use the
    % coefficients for the training and testing data, and the same
    % class labels for the training data that we had before
    %
    % Compute classification accuracy: We're interested in 2 numbers:
    % Correct classification rate - how many eyes were classified as eyes
    % False-positive rate: how many non-eyes were classified as eyes

    % Compute matrix of pairwise distances (this takes a while...)
    dl=som_eucdist2(coeffs,coeffs_train);

    [Cl,Pl]=knn(dl,trainClass,K);

    % Compute the class from C (we have 0s and 1s so it is easy)
    classLowDim=sum(Cl,2);                  % Add how many 1s there are
    classLowDim= (classLowDim>(K/2));   % Set to 1 if there are more than K/2
                           % ones. Otherwise it's zero

    % Compute classification accuracy and add computed rates to the rates array:
    % Correct classification rate - how many eyes were classified as eyes
    % False-positive rate: how many non-eyes were classified as eyes

correctEye_knnl=length(find(classLowDim(1:size(testEyeIm,2))==0))/size(testEyeIm,2);
    ccrKNN(n) = correctEye_knnl;

falseEye_knnl=length(find(classLowDim(size(testEyeIm,2)+1:end)==0))/size(testNonIm,2);
    fprKNN(n) = falseEye_knnl;

    %%% TO PRINT:
    % Print the classification accuracy and false-positive rates for the
    % kNN classification on low-dimensional data and compare with the
    % results on high-dimensional data.
    %
    fprintf(2,'Correct classification rate for Low-dimensional data when PCAcomp = %d
:\n', PCAcomp);
    disp(correctEye_knnl);
    fprintf(2,'False positive rate for Low-dimensional data when PCAcomp = %d :\n',
PCAcomp);
    disp(falseEye_knnl);
    % Discuss in your report:
    % - Are the results better? worse? is this what you expected?
    % - why do you think the results are like this?
    %

    %%% TO DO:
    % Finally, we will do classification directly from the PCA models
    % for eyes and non-eyes.
```

```
    %%% TO DO:
    %
    % Compute the reconstruction for each entry using the PCA model for eyes
    % and separately for non-eyes, compute the error between these 2
    % reconstructions and the original testSet entry, and select the class
    % that yields the smallest error.

    classPCA = zeros(test_N, 1);
    for i=1:test_N
        % vRecon_eye= eyeMean + sum_k (eye_coeff_k * eye_PCA_vector_k);
        vRecon_eye= eyeMean + eyeVec(:, 1:PCAcomp) * coeffsEye(i,:)';
        % vRecon_noneye= noneyeMean + sum_k (noneye_coeff_k * noneye_PCA_vector_k);
        vRecon_noneye= noneyeMean + noneyeVec(:, 1:PCAcomp) * coeffsNonEye(i,:)';
        % compute the square error between two reconstructions and the original
        % testSet entry
        dist_eye = norm(vRecon_eye - testSet(i,:)');
        dist_noneye = norm(vRecon_noneye - testSet(i,:)');
        % if noneye error is smaller, set the class to 1, o.w. 0
        classPCA(i) = dist_noneye < dist_eye;
    end

%     fprintf(2,'PCA correct classification rate for PCAcomp = %d :\n', PCAcomp);
    correctEye_pca=length(find(classPCA(1:size(testEyeIm,2))==0))/size(testEyeIm,2);
    ccrPCA(n) = correctEye_pca;

%     fprintf(2,'PCA False positive rate for PCAcomp = %d :\n', PCAcomp);
    falseEye_pca=length(find(classPCA(size(testEyeIm,2)+1:end)==0))/size(testNonIm,2);
    fprPCA(n) = falseEye_pca;

    n = n + 1;
end
%%% TO PRINT:
%
% Print the correct classification rate and false positive rate for
% the PCA based classifier and the low-dimensional kNN classifier
% using PCAcomps=5,10,15,25, and 50
fprintf(2,'For PCAcomp = 5,10,15,25, and 50:\n');
fprintf(2,'PCA correct classification rate:\n');
disp(ccrPCA);
fprintf(2,'PCA False positive rate:\n');
disp(fprPCA);
fprintf(2,'low-dimensional kNN correct classification rate:\n');
disp(ccrKNN);
fprintf(2,'low-dimensional kNN False positive rate:\n');
disp(fprKNN);
%
% Plot a graph of the kNN classification rate for the low-dimensional
% KNN classifier VS the number of PCA components (for the 5 values of
% PCAcomps requested).
```

```
figure(3);clf;

plot([5,10,15,25,50], ccrKNN, 'bs');
hold on;
plot([5,10,15,25,50], fprKNN, 'g*');

legend('correct classification rate', 'false-positive rate');
xlabel('number of PCA components: PCAcomp');
ylabel('correct classification rate');
title('kNN classification rate for the low-dimensional KNN classifier VS the number of
PCA components');

% Discuss in your Report:
% - Is there a value for PCAcomps (or set of values) for which low-dimensional
%   kNN is better than full dimensional kNN?
% - why do you think that is?
%
% Plot graphs of correct classification rate and the false-positive rate
% fr the PCA-reconstruction classifier vs the number of PCA components.
%

figure(4);clf;

plot([5,10,15,25,50], ccrPCA, 'bs');
hold on;
plot([5,10,15,25,50], fprPCA, 'g*');

legend('correct classification rate', 'false-positive rate');
xlabel('number of PCA components: PCAcomp');
ylabel('correct classification rate');
title('classification rate for the PCA-reconstruction classifier VS the number of PCA
components');



function [V, D] = computePCA(imgs, mean)

    training_N = size(imgs, 2);
    dimension = size(imgs, 1);
    K = zeros(dimension);
    for i=1:training_N
        difference = imgs(:, i) - mean;
        K = K + difference * difference';
    end
    K = K ./ training_N;
    [V_orig, D_orig] = eig(K);

    % refered to:
```

```
    %
http://www.mathworks.com/matlabcentral/fileexchange/18904-sort-eigenvectors---eigenvalu
es
    D=diag(sort(diag(D_orig),'descend')); % make diagonal matrix out of sorted diagonal
values of input D
    [c, ind]=sort(diag(D_orig),'descend'); % store the indices of which columns the
sorted eigenvalues come from
    V=V_orig(:,ind); % arrange the columns in this order

end
```