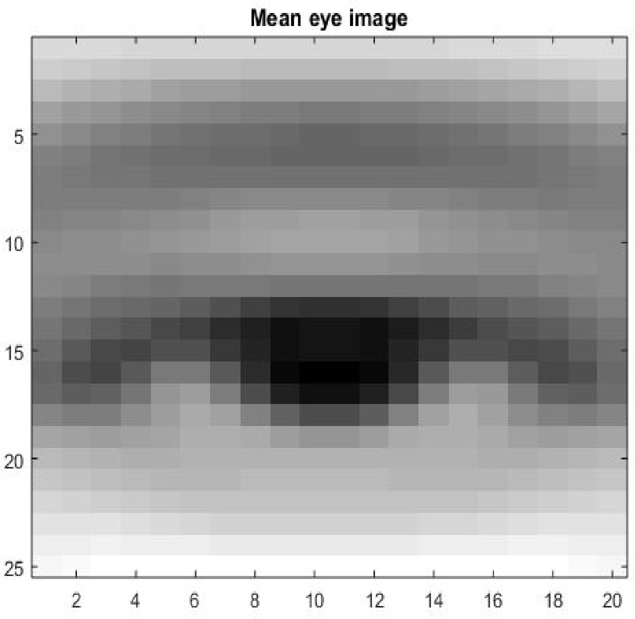
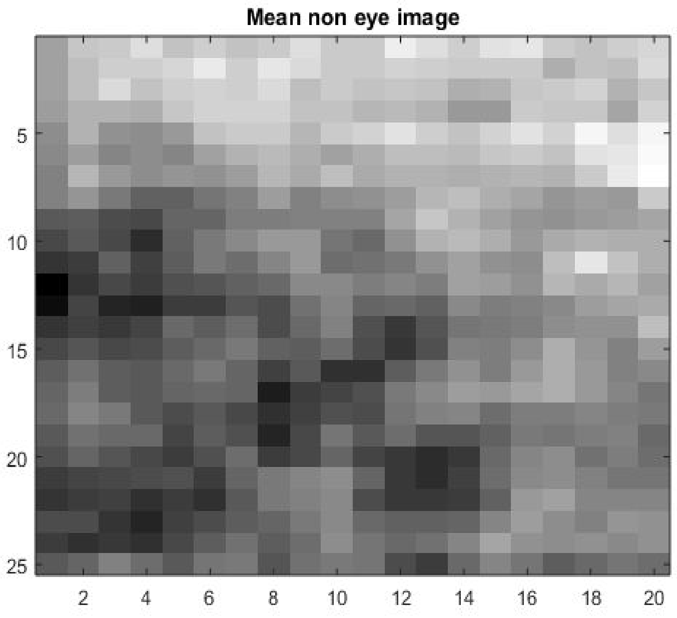
CSCC11 Assignment 4

Huan Ma
1000955120

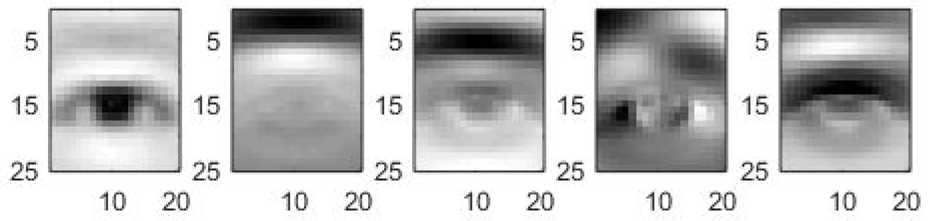
1.Print the average of the eye images

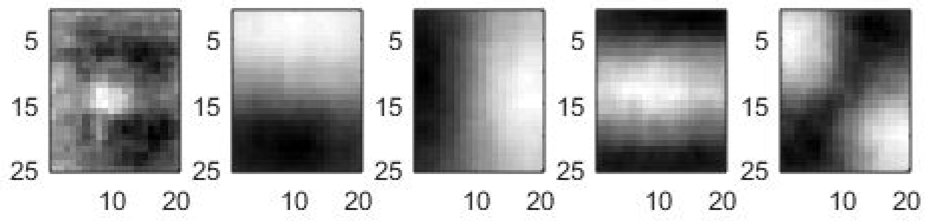


2. Print the average of the non eye images



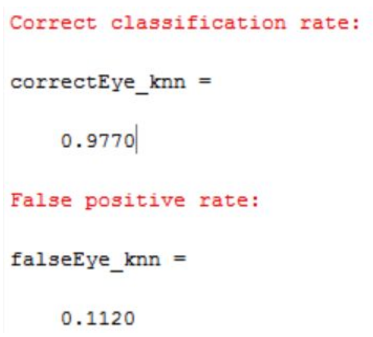
3. Print the first 5 eigenvectors for eye and non eye images





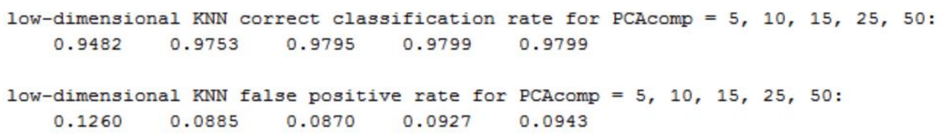
4. Compute Classification Rates for a kNN Classifier. Compute the correct classification rate, and the false-positive rate for the kNN classifier based on the low-dimensional PCA representations and compare with these results to those based on the original high-dimensional data (already computed for you earlier in EyeDetector.m). Are the results better or worse? Why?

Answer:
KNN classifier on the original high dimensional data:



```
Correct classification rate:

correctEye_knn =

    0.9770

False positive rate:

falseEye_knn =

    0.1120
```

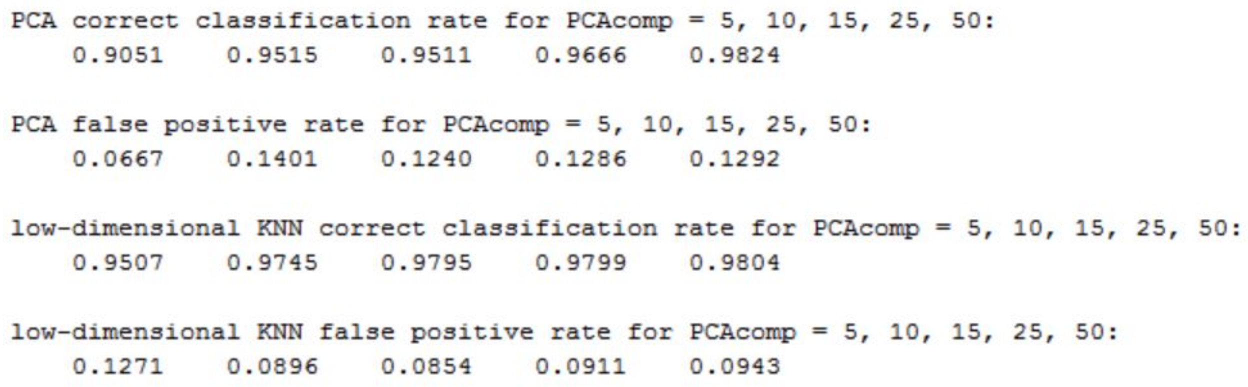
KNN classifier on the low dimensional PCA representations:



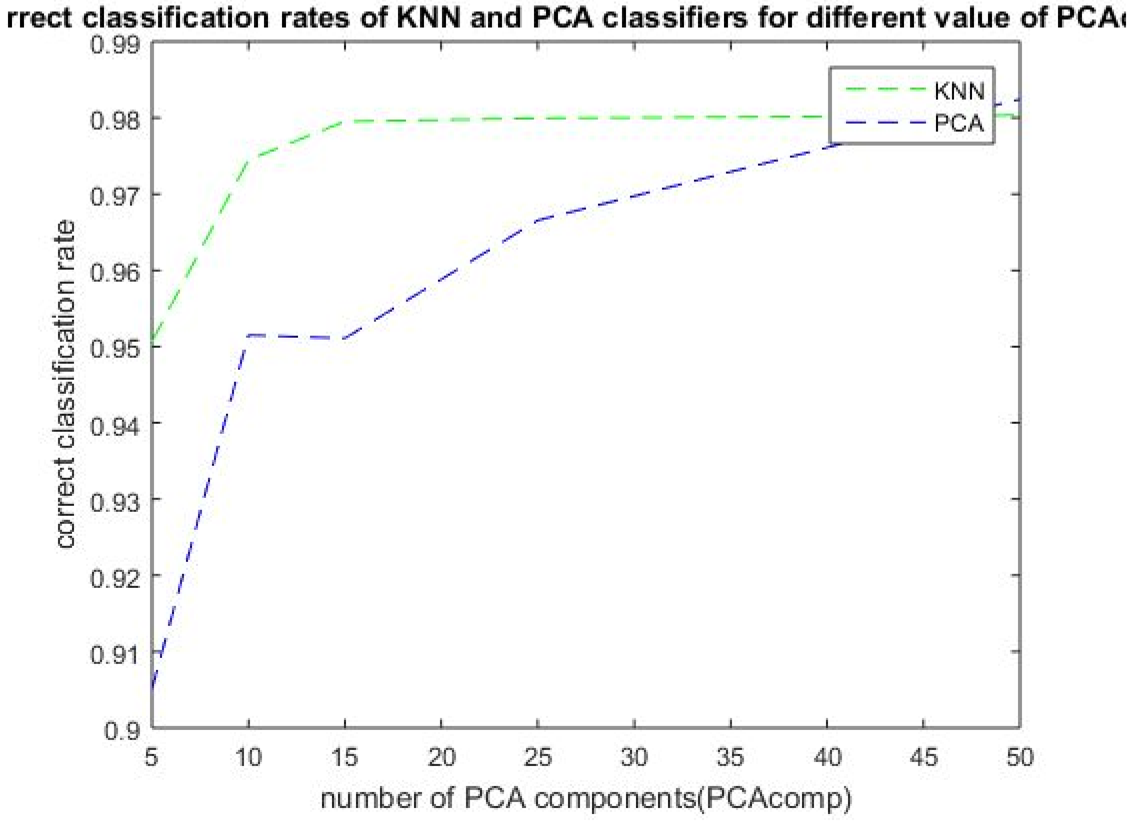
```
low-dimensional KNN correct classification rate for PCAcomp = 5, 10, 15, 25, 50:
    0.9482    0.9753    0.9795    0.9799    0.9799

low-dimensional KNN false positive rate for PCAcomp = 5, 10, 15, 25, 50:
    0.1260    0.0885    0.0870    0.0927    0.0943
```

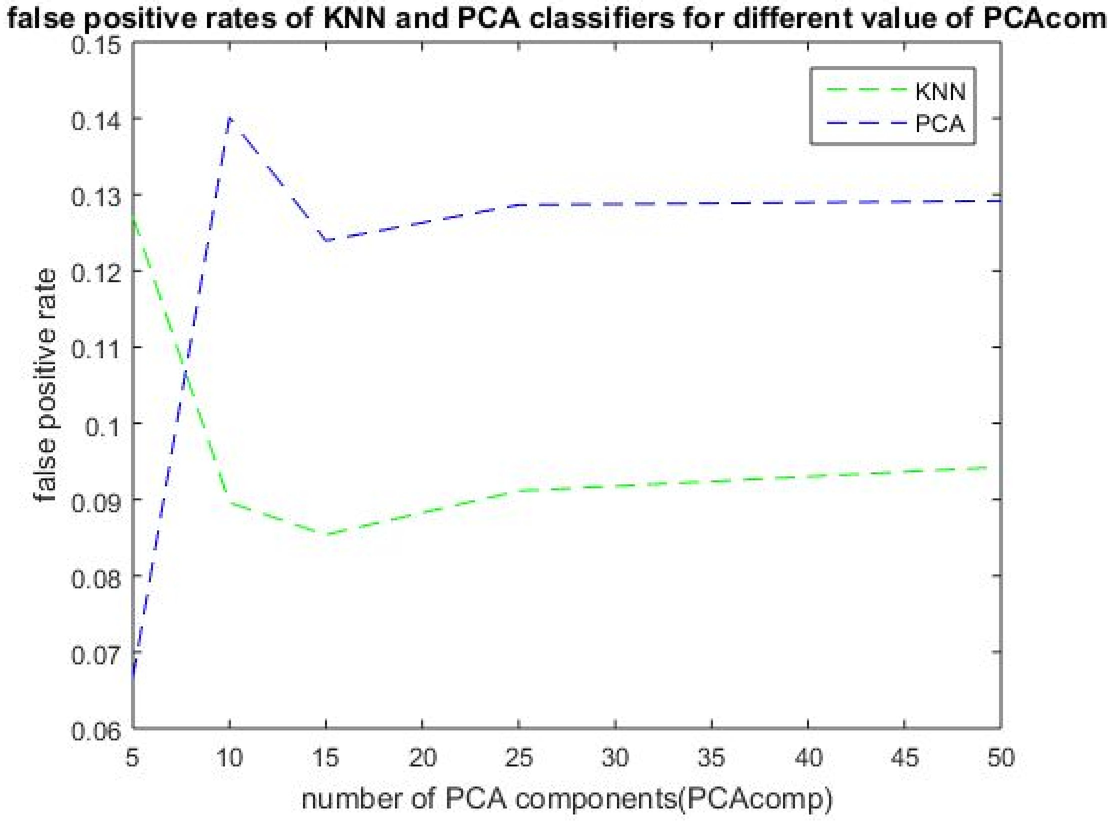
The results at PCAcomp = 5 is worse because the number of PCA components is too small that the PCA transformed data can't account for as much as of the variation of the original data . When PCAcomp = 10, the result has a worse correct classification rate but a better false positive rate than KNN trained on the original data. I think this is because that the number of PCA components is not large enough, large loss of information on the data exists. When PCAcomp larger than 10, the result has a better correct classification rate and false positive rate than the original KNN classifiers because the number of PCA components is large enough to accounts for as much as the variation of the original data as possible without large loss of information.

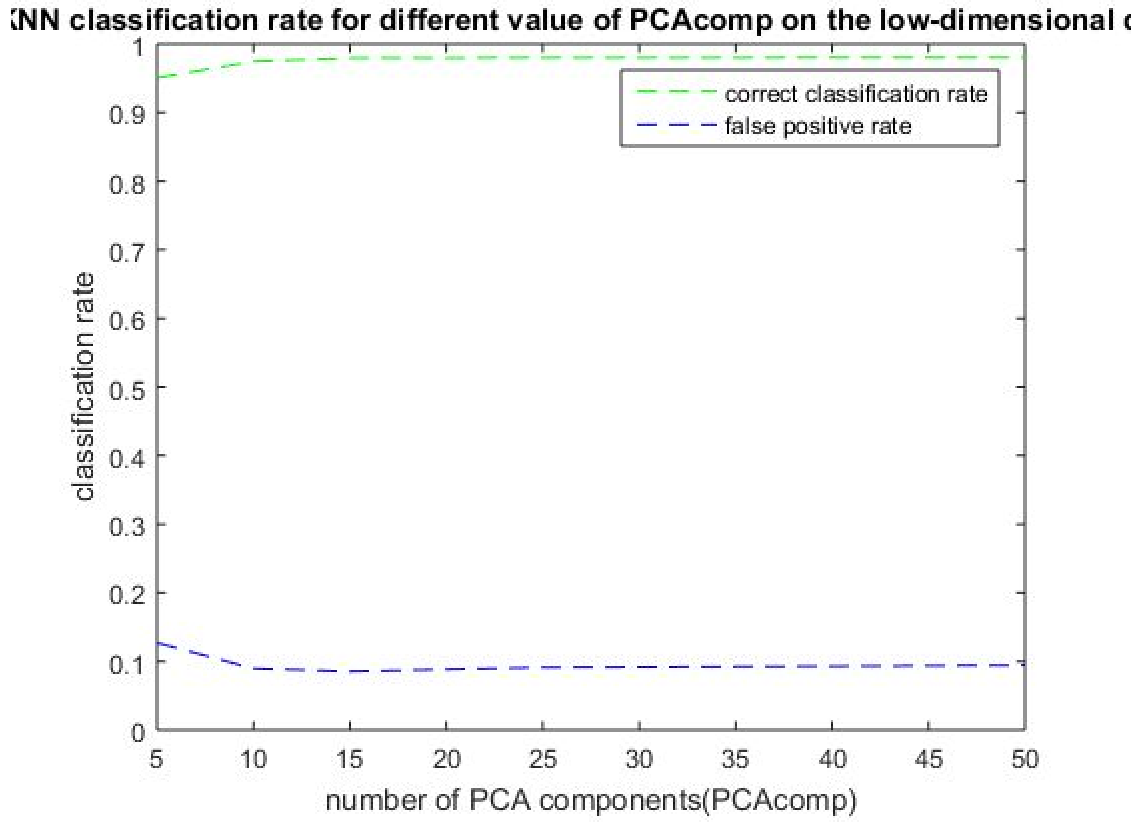
5. Print the correct classification rate and the false positive rate for this classifier, and compare them to the low-dimensional kNN classifier. Do this for models with different subspace dimensions; use PCAcomp equal to 5, 10, 15, 25 and 50. Plot graphs of the correct classification rates vs the number of PCA components for the two types of classifier. Do the same for the false-positive rate vs the number of PCA components.

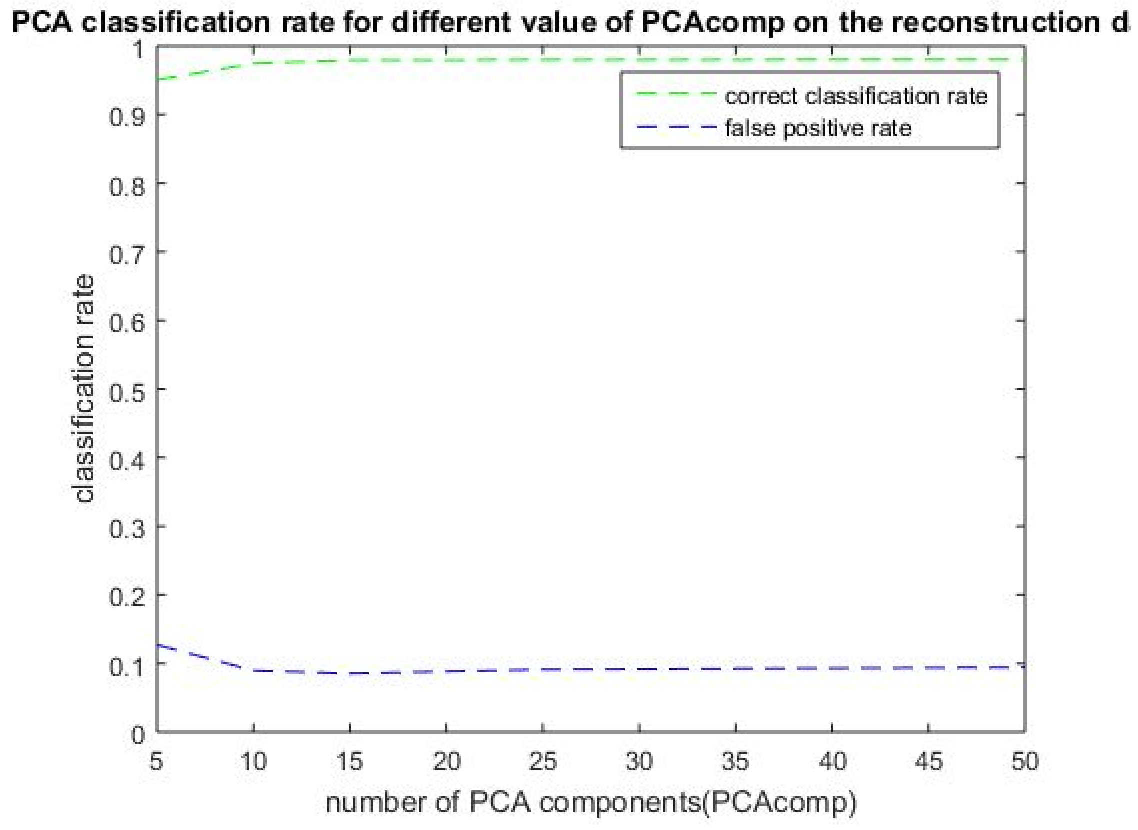


```
PCA correct classification rate for PCAcomp = 5, 10, 15, 25, 50:
    0.9051    0.9515    0.9511    0.9666    0.9824

PCA false positive rate for PCAcomp = 5, 10, 15, 25, 50:
    0.0667    0.1401    0.1240    0.1286    0.1292

low-dimensional KNN correct classification rate for PCAcomp = 5, 10, 15, 25, 50:
    0.9507    0.9745    0.9795    0.9799    0.9804

low-dimensional KNN false positive rate for PCAcomp = 5, 10, 15, 25, 50:
    0.1271    0.0896    0.0854    0.0911    0.0943
```


rrect classification rates of KNN and PCA classifiers for different value of PCA
correct classification rate
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
number of PCA components(PCAcomp)
KNN
PCA


false positive rates of KNN and PCA classifiers for different value of PCAcom
false positive rate
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
number of PCA components(PCAcomp)
KNN
PCA


KNN classification rate for different value of PCAcomp on the low-dimensional
correct classification rate
false positive rate
classification rate
number of PCA components(PCAcomp)
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


PCA classification rate for different value of PCAcomp on the reconstruction d
correct classification rate
false positive rate
classification rate
number of PCA components(PCAcomp)
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

6. 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?
Answer:
Yes. When PCAcomp is larger than 10, the low-dimensional KNN is better than full-dimensional KNN. I think it is because that the PCA components number is large enough so that there is only little loss of information on the data and it reduces the effect of noises significantly so that low-dimensional KNN has better performance than full-dimensional KNN.

7. Which classifier gives the overall best performance?
Answer:
The low-dimensional kNN classifier has the overall best performance, it has the lowest false positive rate and highest correct classification rate when PCAcomp is larger than 10 among 3 classifiers.

8. What conclusions can you draw about the usefulness of dimensionality reduction?
Answer:
Dimensionality reduction can reduce the effect of noises and extract the most important information on the data so that the KNN and PCA classifiers based on PCA transformed data will have a better performance than the KNN classifier on the original data.

9. Which classifier would you use on a large training set consisting of high-dimensional data?
Answer:
kNN classifier based on the low-dimensional PCA representation. Dimensionality reduction on the large training set consisting of high-dimensional data will reduce the effect of noises and improve the efficiency of the training process as well as the classification rate without large loss of information if we choose the number of PCAcomp wisely.

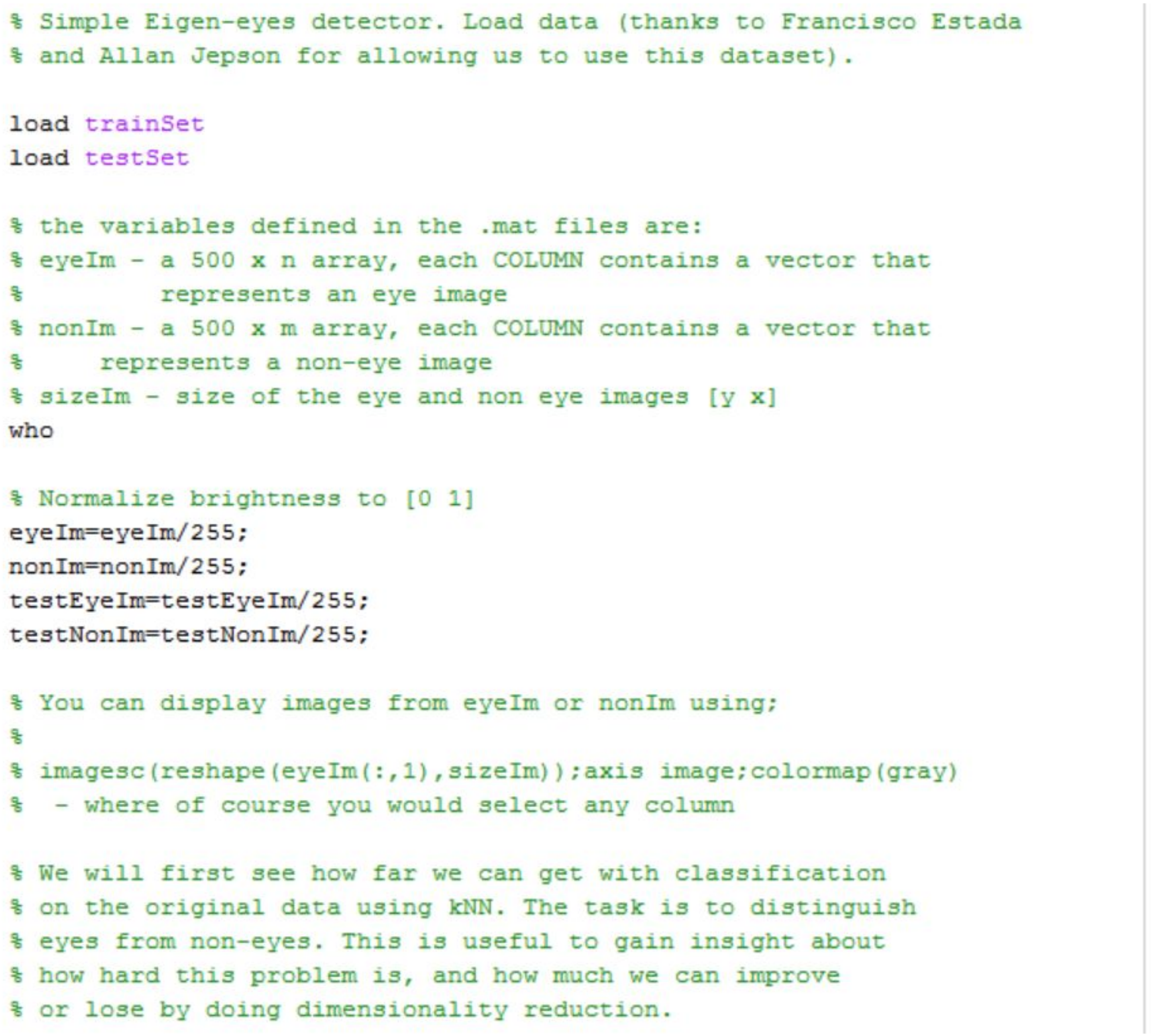
10. Which classifier would you use on a large training set of low-dimensional (e.g. 4D) data? Why?
Answer:
kNN classifier based on the full dimensional data. Because the training set is low-dimensional data. If we perform PCA on the data and choose only a subset of the PCA components, there could be large loss of information which will lead to much lower classification rate.

11. Summarize the advantages/disadvantages of each classifier!
Answer:
Advantage:
Low-dimensional KNN classifier: High result accuracy on high dimensional data, fast when data set is large

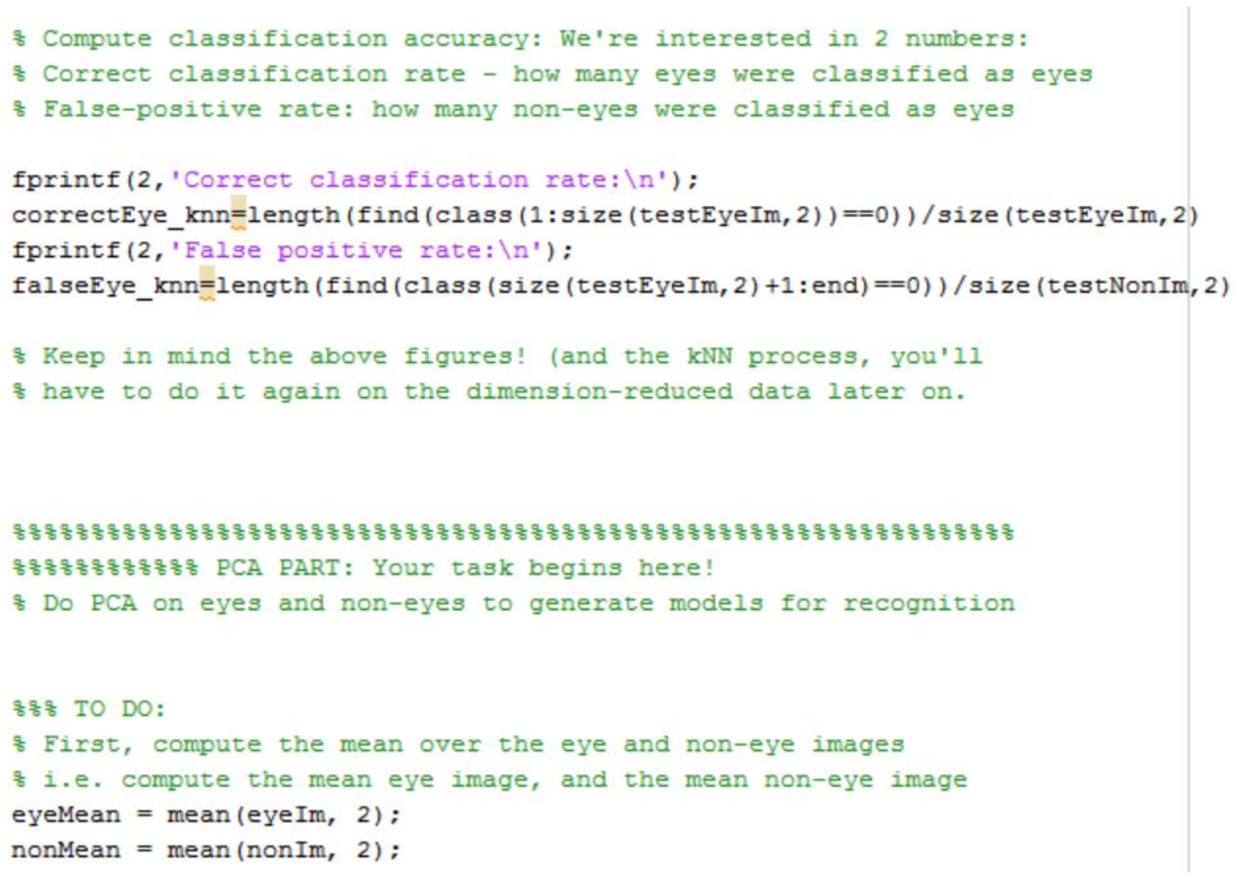
Full-dimensional KNN classifier: High result accuracy on low dimensional data normally, the performance on high dimensional data is not very inaccurate too.
PCA classifier: fast when data set is large, intuitive and easy to implement the algorithm

Disadvantage:
Low-dimensional KNN classifier: not suitable for low dimensional data, need to choose the number of PCA components appropriately
Full-dimensional KNN classifier: slow when data set is large, not as precise as low-dimensional KNN classifier on high dimensional data
PCA classifier: not as accurate as the above two kind of classifiers, need to choose the number of PCA components appropriately too

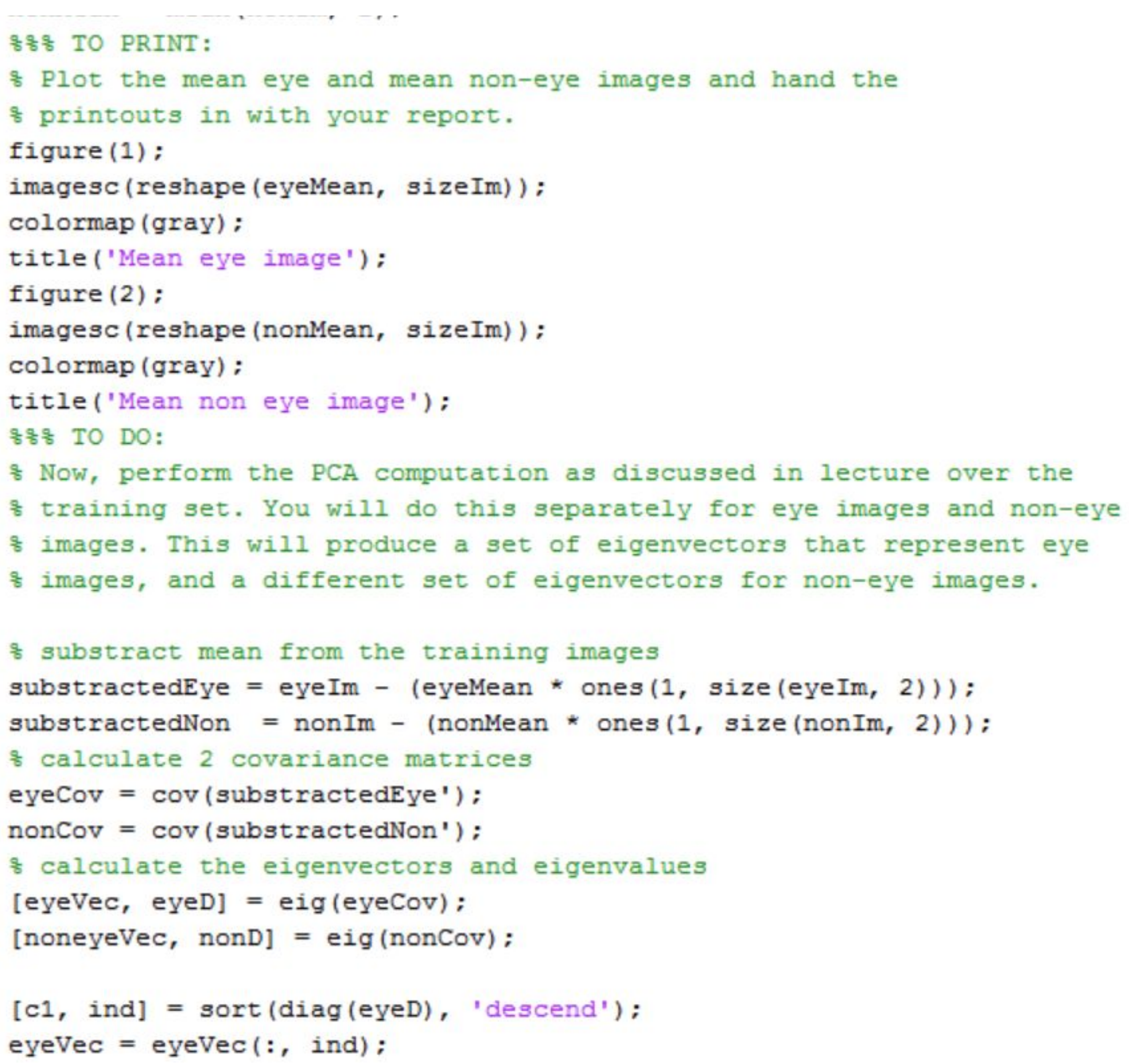
Code:



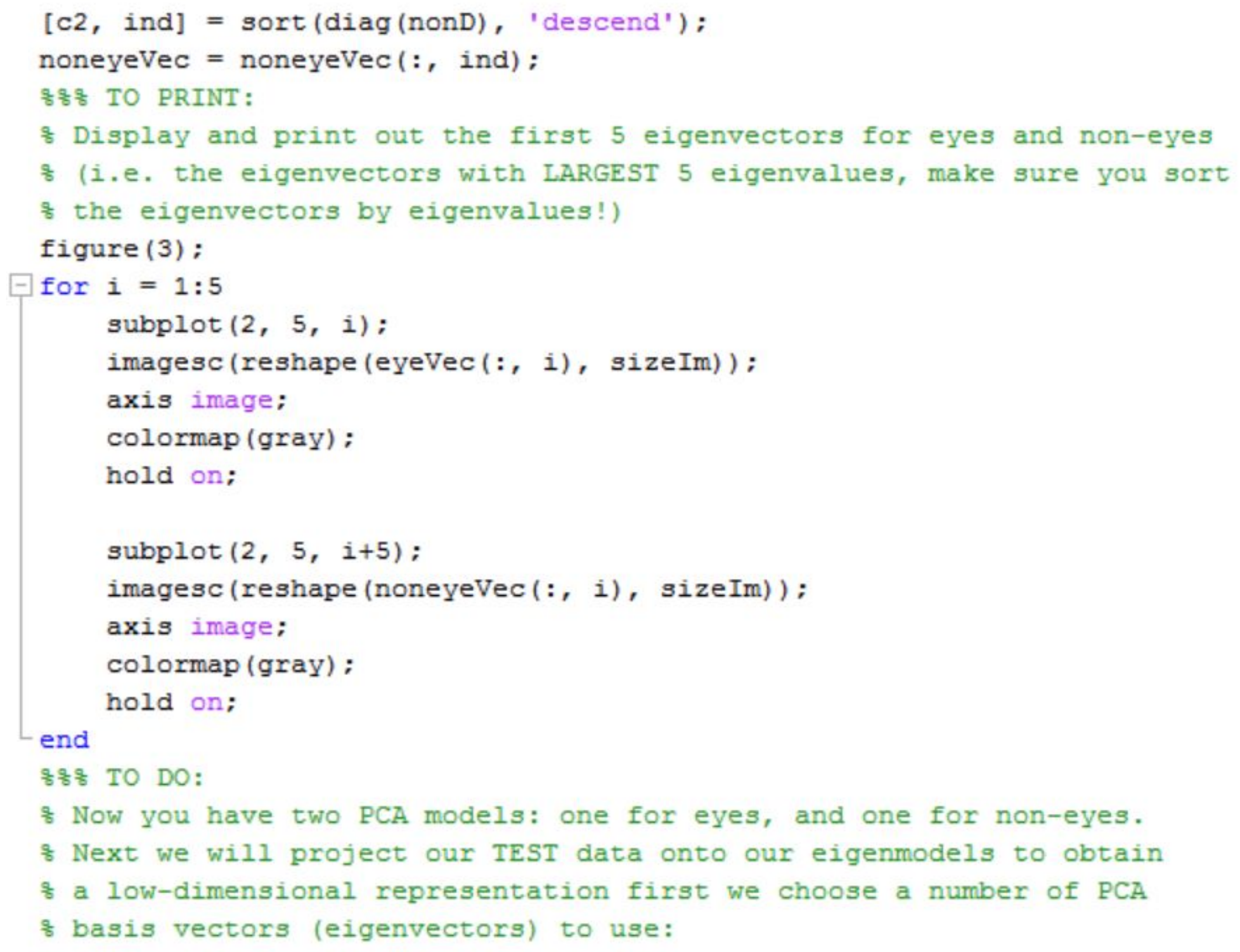
```
% Simple Eigen-eyes detector. Load data (thanks to Francisco Estada
% and Allan Jepson for allowing us to use this dataset).

load trainSet
load testSet

% the variables defined in the .mat files are:
% eyeIm - a 500 x n array, each COLUMN contains a vector that
%         represents an eye image
% nonIm - a 500 x m array, each COLUMN contains a vector that
%     represents a non-eye image
% sizeIm - size of the eye and non eye images [y x]
who

% Normalize brightness to [0 1]
eyeIm=eyeIm/255;
nonIm=nonIm/255;
testEyeIm=testEyeIm/255;
testNonIm=testNonIm/255;

% You can display images from eyeIm or nonIm using;
%
% imagesc(reshape(eyeIm(:,1),sizeIm));axis image;colormap(gray)
%  - where of course you would select any column

% We will first see how far we can get with classification
% on the original data using kNN. The task is to distinguish
% eyes from non-eyes. This is useful to gain insight about
% how hard this problem is, and how much we can improve
% or lose by doing dimensionality reduction.
% Generate training and testing sets with classes for kNN,
% we need eye images to be on ROWS, not COLUMNS, and we also
% need a vector with class labels for each

trainSet=[eyeIm'
          nonIm'];
trainClass=[zeros(size(eyeIm,2),1)
            ones(size(nonIm,2),1)];

testSet=[testEyeIm'
         testNonIm'];
testClass=[zeros(size(testEyeIm,2),1)
           ones(size(testNonIm,2),1)];

% Compute matrix of pairwise distances (this takes a while...)
d=som_eucdist2(testSet,trainSet);

% Compute kNN results, I simply chose a reasonable value
% for K but feel free to change it and play with it...
K=5;
[C,P]=knn(d,trainClass,K);

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



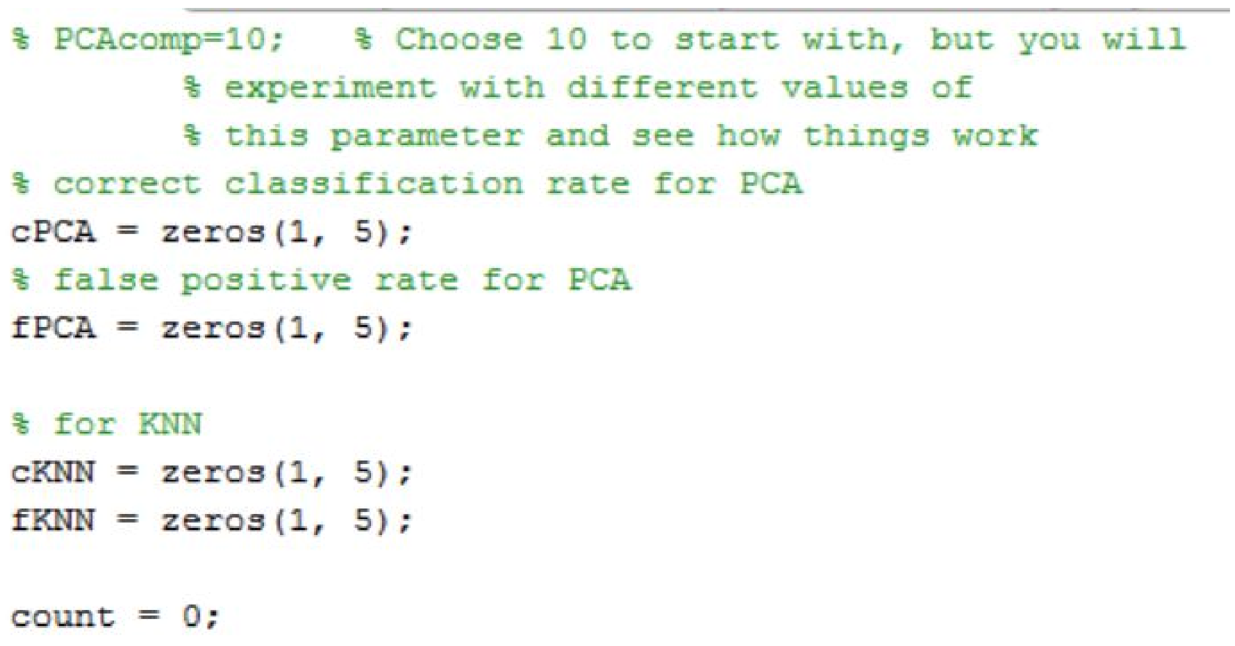
```
% Compute classification accuracy: We're interested in 2 numbers:
% Correct classification rate - how many eyes were classified as eyes
% False-positive rate: how many non-eyes were classified as eyes

fprintf(2,'Correct classification rate:\n');
correctEye_knn=length(find(class(1:size(testEyeIm,2))==0))/size(testEyeIm,2)
fprintf(2,'False positive rate:\n');
falseEye_knn=length(find(class(size(testEyeIm,2)+1:end)==0))/size(testNonIm,2)

% Keep in mind the above figures! (and the kNN process, you'll
% have to do it again on the dimension-reduced data later on.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% 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);
nonMean = mean(nonIm, 2);
```



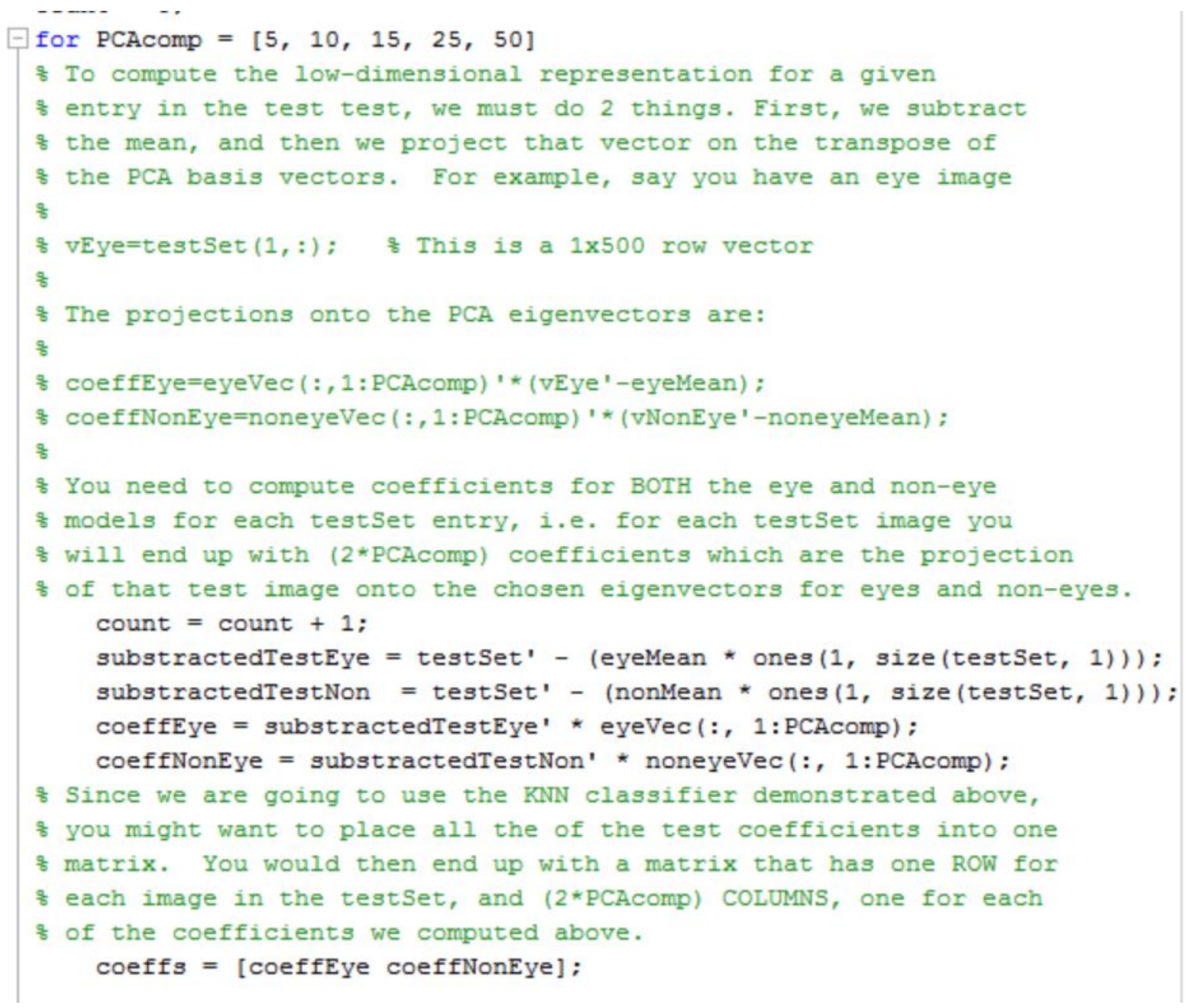
```
%%% TO PRINT:
% Plot the mean eye and mean non-eye images and hand the
% printouts in with your report.
figure(1);
imagesc(reshape(eyeMean, sizeIm));
colormap(gray);
title('Mean eye image');
figure(2);
imagesc(reshape(nonMean, sizeIm));
colormap(gray);
title('Mean non eye image');
%%% TO DO:
% Now,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

% substract mean from the training images
substractedEye = eyeIm - (eyeMean * ones(1, size(eyeIm, 2)));
substractedNon  = nonIm - (nonMean * ones(1, size(nonIm, 2)));
% calculate 2 covariance matrices
eyeCov = cov(substractedEye');
nonCov = cov(substractedNon');
% calculate the eigenvectors and eigenvalues
[eyeVec, eyeD] = eig(eyeCov);
[noneyeVec, nonD] = eig(nonCov);

[c1, ind] = sort(diag(eyeD), 'descend');
eyeVec = eyeVec(:, ind);
```



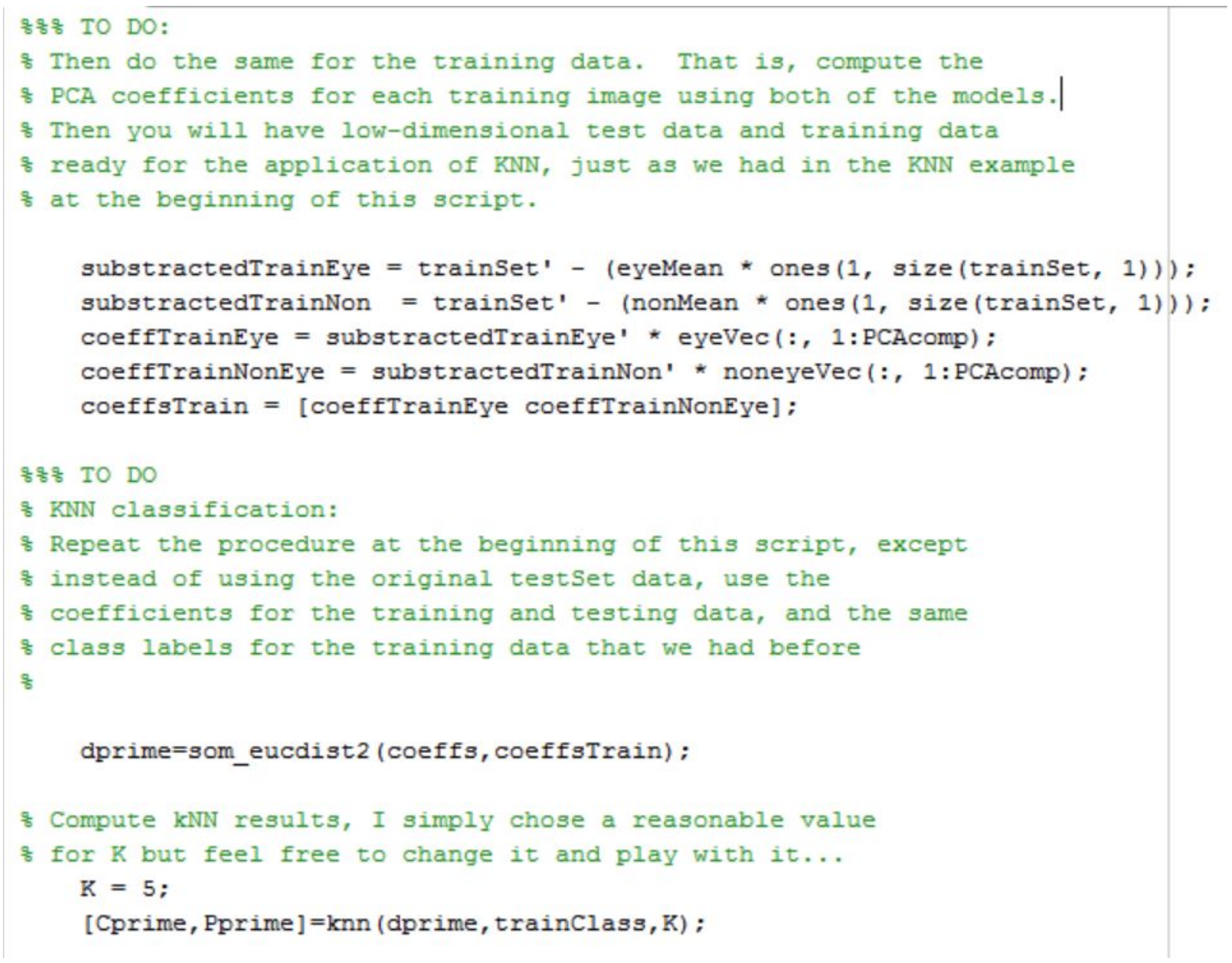
```
[c2, ind] = sort(diag(nonD), 'descend');
noneyeVec = noneyeVec(:, ind);
%%% TO PRINT:
% Display and print out the first 5 eigenvectors for eyes and non-eyes
% (i.e. the eigenvectors with LARGEST 5 eigenvalues, make sure you sort
% the eigenvectors by eigenvalues!)
figure(3);
for i = 1:5
    subplot(2, 5, i);
    imagesc(reshape(eyeVec(:, i), sizeIm));
    axis image;
    colormap(gray);
    hold on;

    subplot(2, 5, i+5);
    imagesc(reshape(noneyeVec(:, i), sizeIm));
    axis image;
    colormap(gray);
    hold on;
end
%%% 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:
```



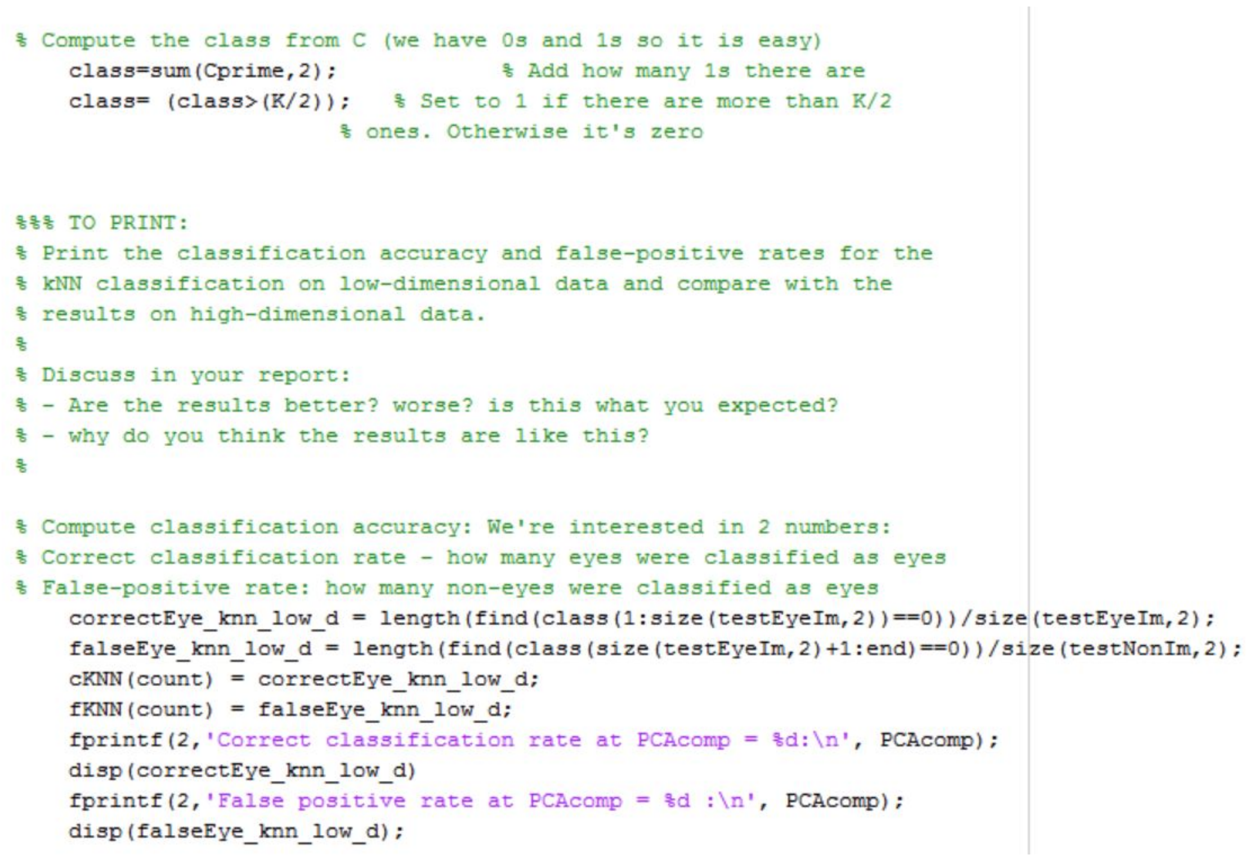
```
% PCAcomp=10;    % Choose 10 to start with, but you will
                 % experiment with different values of
                 % this parameter and see how things work
% correct classification rate for PCA
cPCA = zeros(1, 5);
% false positive rate for PCA
fPCA = zeros(1, 5);

% for KNN
cKNN = zeros(1, 5);
fKNN = zeros(1, 5);

count = 0;
```



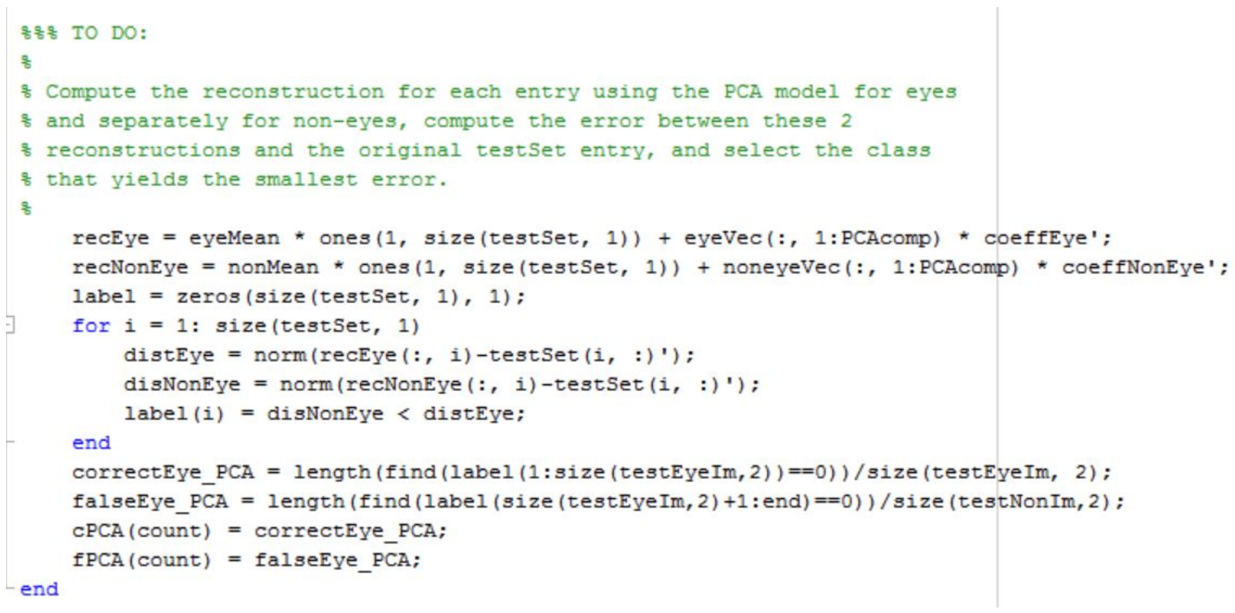
```matlab
for PCAcomp = [5, 10, 15, 25, 50]
% To compute the low-dimensional representation for a given
% entry in the test test, we must do 2 things. First, we subtract
% the mean, and then we project that vector on the transpose of
% the PCA basis vectors.  For example, say you have an eye image
%
% vEye=testSet(1,:);   % This is a 1x500 row vector
%
% The projections onto the PCA eigenvectors are:
%
% coeffEye=eyeVec(:,1:PCAcomp)'*(vEye'-eyeMean);
% coeffNonEye=noneyeVec(:,1:PCAcomp)'*(vNonEye'-noneyeMean);
%
% 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.
    count = count + 1;
    substractedTestEye = testSet' - (eyeMean * ones(1, size(testSet, 1)));
    substractedTestNon  = testSet' - (nonMean * ones(1, size(testSet, 1)));
    coeffEye = substractedTestEye' * eyeVec(:, 1:PCAcomp);
    coeffNonEye = substractedTestNon' * noneyeVec(:, 1:PCAcomp);
% Since we are going to use the KNN classifier demonstrated above,
% you might want to place all the of the test coefficients into one
% matrix.  You would then end up with a matrix that has one ROW for
% each image in the testSet, and (2*PCAcomp) COLUMNS, one for each
% of the coefficients we computed above.
    coeffs = [coeffEye coeffNonEye];
```



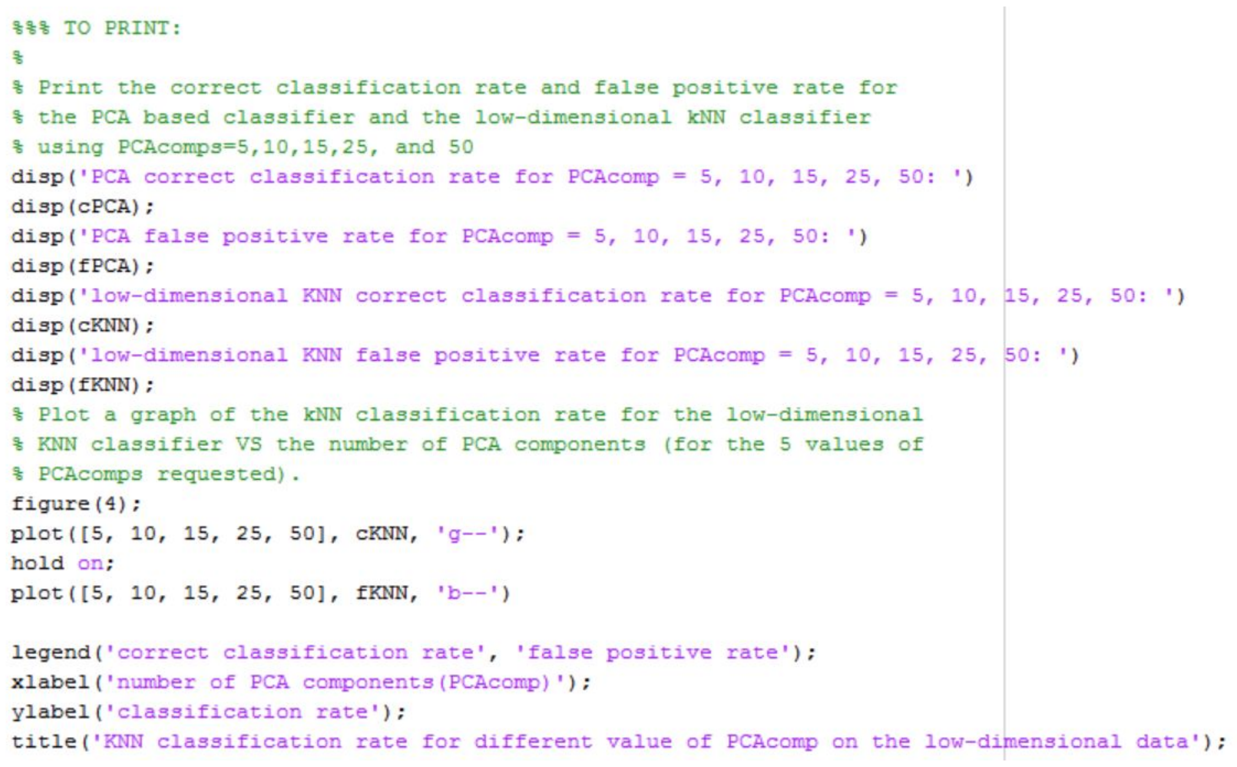
```
%%% TO DO:
% Then do the same for the training data.  That is, compute the
% PCA coefficients for each training image using both of the models.
% Then you will have low-dimensional test data and training data
% ready for the application of KNN, just as we had in the KNN example
% at the beginning of this script.

    substractedTrainEye = trainSet' - (eyeMean * ones(1, size(trainSet, 1)));
    substractedTrainNon  = trainSet' - (nonMean * ones(1, size(trainSet, 1)));
    coeffTrainEye = substractedTrainEye' * eyeVec(:, 1:PCAcomp);
    coeffTrainNonEye = substractedTrainNon' * noneyeVec(:, 1:PCAcomp);
    coeffsTrain = [coeffTrainEye coeffTrainNonEye];

%%% 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
%

    dprime=som_eucdist2(coeffs,coeffsTrain);

% Compute kNN results, I simply chose a reasonable value
% for K but feel free to change it and play with it...
    K = 5;
    [Cprime,Pprime]=knn(dprime,trainClass,K);
```



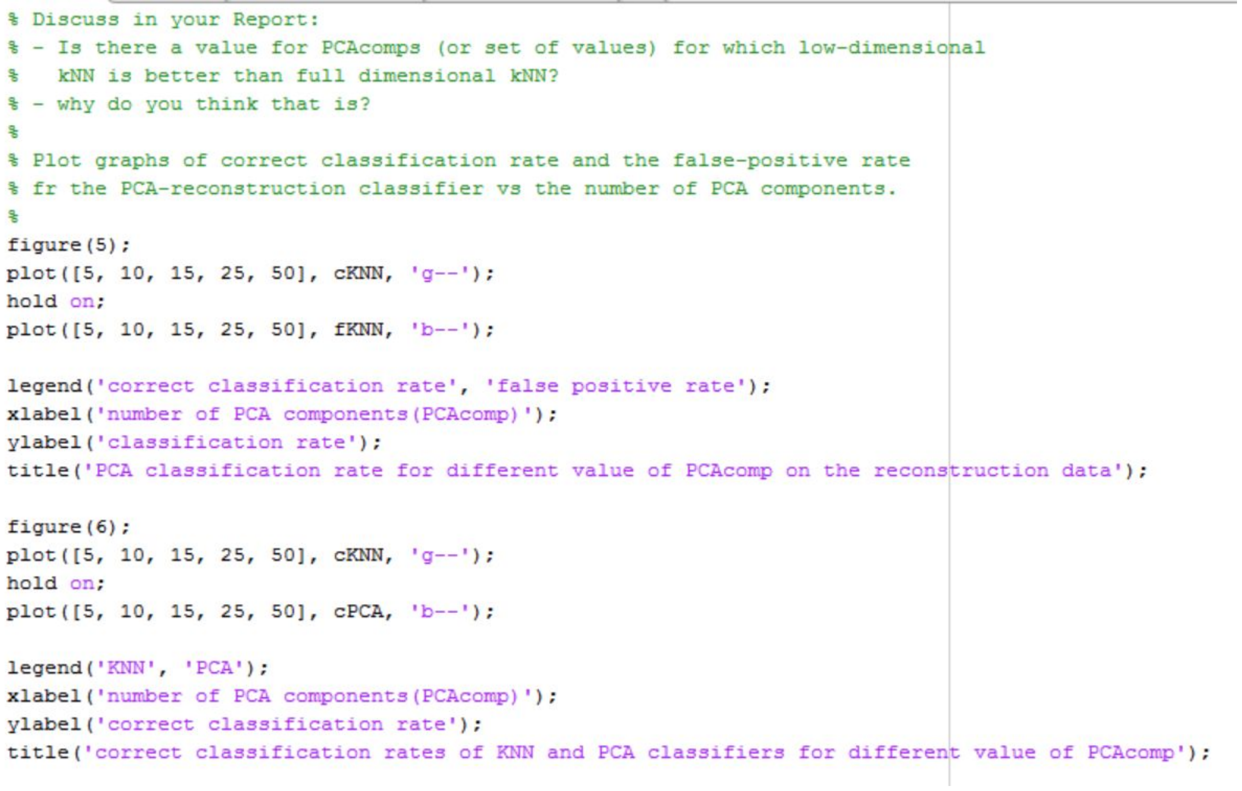
```
% Compute the class from C (we have 0s and 1s so it is easy)
    class=sum(Cprime,2);             % Add how many 1s there are
    class= (class>(K/2));   % Set to 1 if there are more than K/2
                        % ones. Otherwise it's zero


%%% 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.
%
% Discuss in your report:
% - Are the results better? worse? is this what you expected?
% - why do you think the results are like this?
%

% 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
    correctEye_knn_low_d = length(find(class(1:size(testEyeIm,2))==0))/size(testEyeIm,2);
    falseEye_knn_low_d = length(find(class(size(testEyeIm,2)+1:end)==0))/size(testNonIm,2);
    cKNN(count) = correctEye_knn_low_d;
    fKNN(count) = falseEye_knn_low_d;
    fprintf(2,'Correct classification rate at PCAcomp = %d:\n', PCAcomp);
    disp(correctEye_knn_low_d)
    fprintf(2,'False positive rate at PCAcomp = %d :\n', PCAcomp);
    disp(falseEye_knn_low_d);
```



```
%%% 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.
%
    recEye = eyeMean * ones(1, size(testSet, 1)) + eyeVec(:, 1:PCAcomp) * coeffEye';
    recNonEye = nonMean * ones(1, size(testSet, 1)) + noneyeVec(:, 1:PCAcomp) * coeffNonEye';
    label = zeros(size(testSet, 1), 1);
    for i = 1: size(testSet, 1)
        distEye = norm(recEye(:, i)-testSet(i, :)');
        disNonEye = norm(recNonEye(:, i)-testSet(i, :)');
        label(i) = disNonEye < distEye;
    end
    correctEye_PCA = length(find(label(1:size(testEyeIm,2))==0))/size(testEyeIm, 2);
    falseEye_PCA = length(find(label(size(testEyeIm,2)+1:end)==0))/size(testNonIm,2);
    cPCA(count) = correctEye_PCA;
    fPCA(count) = falseEye_PCA;
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
disp('PCA correct classification rate for PCAcomp = 5, 10, 15, 25, 50: ')
disp(cPCA);
disp('PCA false positive rate for PCAcomp = 5, 10, 15, 25, 50: ')
disp(fPCA);
disp('low-dimensional KNN correct classification rate for PCAcomp = 5, 10, 15, 25, 50: ')
disp(cKNN);
disp('low-dimensional KNN false positive rate for PCAcomp = 5, 10, 15, 25, 50: ')
disp(fKNN);
% 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(4);
plot([5, 10, 15, 25, 50], cKNN, 'g--');
hold on;
plot([5, 10, 15, 25, 50], fKNN, 'b--')

legend('correct classification rate', 'false positive rate');
xlabel('number of PCA components(PCAcomp)');
ylabel('classification rate');
title('KNN classification rate for different value of PCAcomp on the low-dimensional data');
```



```
% 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(5);
plot([5, 10, 15, 25, 50], cKNN, 'g--');
hold on;
plot([5, 10, 15, 25, 50], fKNN, 'b--');

legend('correct classification rate', 'false positive rate');
xlabel('number of PCA components(PCAcomp)');
ylabel('classification rate');
title('PCA classification rate for different value of PCAcomp on the reconstruction data');

figure(6);
plot([5, 10, 15, 25, 50], cKNN, 'g--');
hold on;
plot([5, 10, 15, 25, 50], cPCA, 'b--');

legend('KNN', 'PCA');
xlabel('number of PCA components(PCAcomp)');
ylabel('correct classification rate');
title('correct classification rates of KNN and PCA classifiers for different value of PCAcomp');
```

```
figure(7);
plot([5, 10, 15, 25, 50], fKNN, 'g--');
hold on;
plot([5, 10, 15, 25, 50], fPCA, 'b--');

legend('KNN', 'PCA');
xlabel('number of PCA components(PCAcomp)');
ylabel('false positive rate');
title('false positive rates of KNN and PCA classifiers for different value of PCAcomp');
% Discuss in your Report:
% - Which classifier gives the overall best performance?
% - What conclusions can you draw about the usefulness of dimensionality
%   reduction?
% - Which classifier would you use on a large training set
%   consisting of high-dimensional data?
% - Which classifier would you use on a large training set
%   of low-dimensional (e.g. 3-D) data?
% - why?
% - Summarize the advantages/disadvantages of each classifier!
```