2.

e. Confusion table:

True class	Classified class		
	Setosa	Versicolor	Virginica
Setosa	20	0	0
Versicolor	0	20	0
Virginica	0	0	20

f. Scatter plot for the PCA projected Iris data:

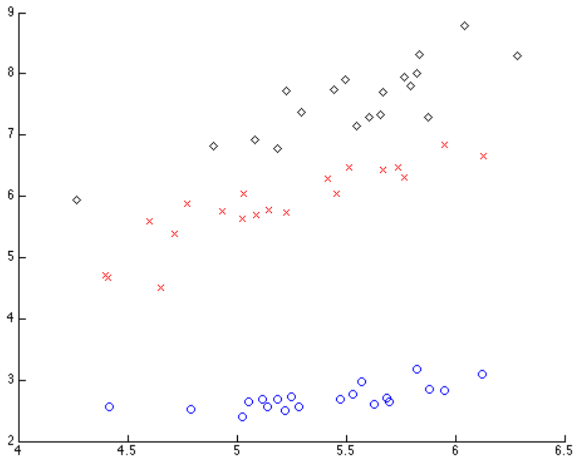

Figure 1: 2D projected test dataset

g. The original 4D Iris test dataset was projected into 2D data samples using PCA before classification task is performed and for the given set of test data we found that there is no classification error using MLE classifier which can be seen from the scatter plot and the confusion table.

3.

c. The classification task of the Iris data is a 3-class classification problem and for the Multiple discriminant analysis which is a generalization of Fisher's linear discriminant involves (3-1) discriminant functions hence the dimensionality of projection is not possible to get up to 3.

e. Confusion table (MDA):

True class	Classified class		
	Setosa	Versicolor	Virginica
Setosa	20	0	0
Versicolor	0	20	0
Virginica	0	0	20

g. Scatter plot for the PCA projected Iris data with MDA:

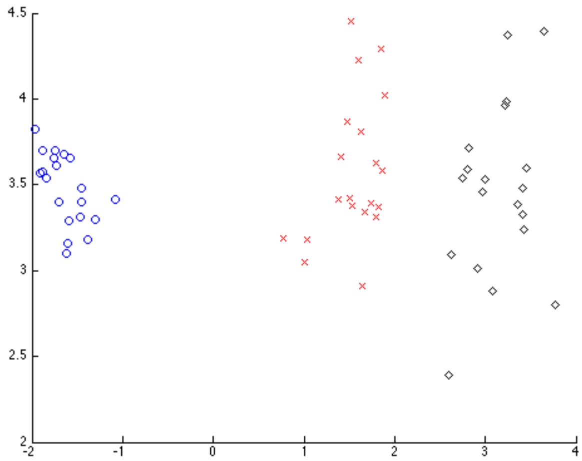

Figure 2: 2D projected test dataset

h. Same as the classification result of the PCA function, projected 2D dataset produced by MDA function also work really well with MLE classifier and we could also get zero classification error for the given dataset.

4.

b. Display original and approximated images:

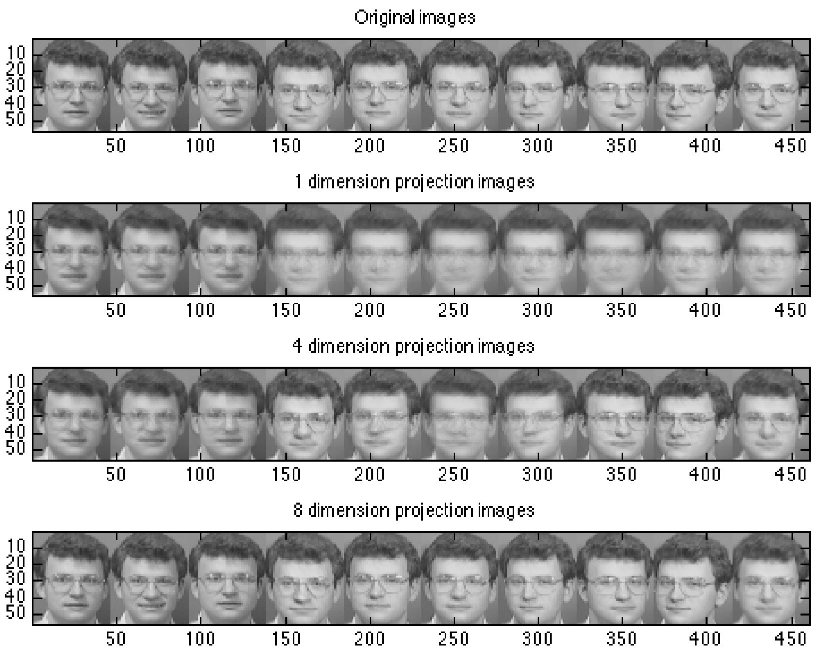

Figure 3: Original and approximated images

c. Proportion of variances(beta_k) and approximation error(e_square).

```
k:1 => beta_k = 0.40158, e_square = 631257.3661
k:4 => beta_k = 0.78715, e_square = 224526.7043
k:8 => beta_k = 0.98096, e_square = 20080.7289
```

5.

c. Classification error rate summary:

# of nearest neighbors	# of features		
	1	4	10
1	0.32	0	0
3	0.24	0	0
5	0.2	0	0

6.

a. Plot of PDFs with h=0.8:

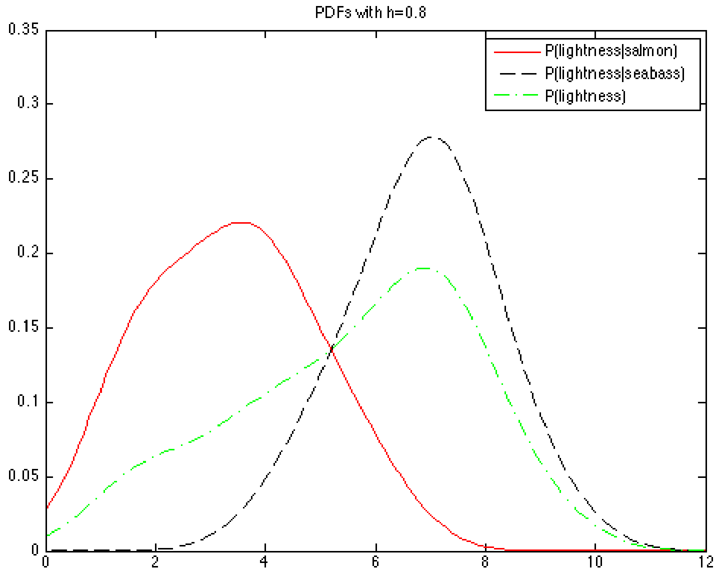

Figure 4: PDFs with h=0.8

b. Plot of PDFs with h=0.2:

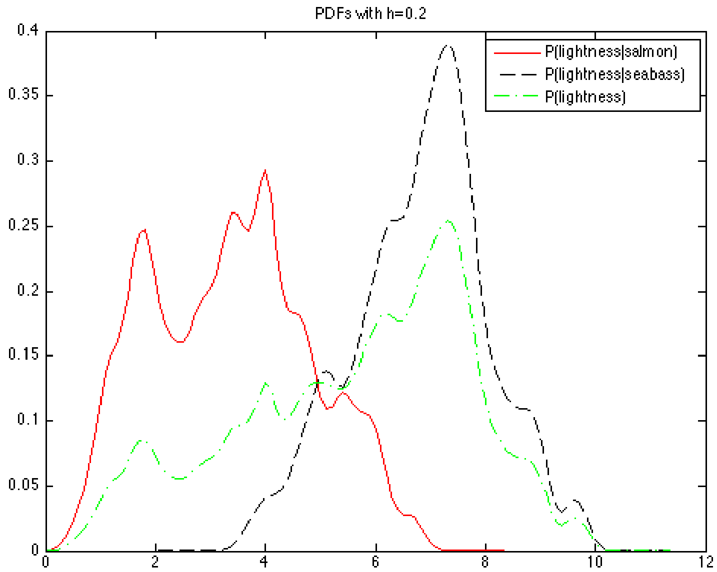

Figure 5: PDFs with h=0.2

c. The glance into the above plots in (a) and (b) clearly reveals the effect of the difference in the window size parameter **h** of the Parzen-window method to learn the underlying distribution of the data. When in the case of $h = 0.8$ the 3 PDFs namely $P(lightness|salmon)$, $P(lightness|seabass)$ and $P(lightness)$ is quite smooth comparing to when the $h = 0.2$ which gives rather spiky distributions.

Appendix:

assignment_4.m

```
%
% CS7720 Spring 2015
% Introduction to Machine Learning and Pattern Recognition
% University of Missouri-Columbia
%
% Author: Chanmann Lim
% email: cl9p8@mail.missouri.edu
%
% Homework Assignment 4
%
clc; clear; close all;

%% Load iris data
setosa     = 'setosa    ';
versicolor = 'versicolor';
virginica  = 'virginica ';

[x1, x2, x3, x4, y] = textread('iris.data', '%f,%f,%f,%f,Iris-%s');

X = [x1 x2 x3 x4];
y = char(y);

X_given_setosa = X(strcomp(y, setosa), :);
X_pca_given_versicolor = X(strcomp(y, versicolor), :);
X_given_virginica = X(strcomp(y, virginica), :);

[X_given_setosa_training, X_given_setosa_test] = split(X_given_setosa);
[X_given_versicolor_training, X_given_versicolor_test] = split(X_pca_given_versicolor);
[X_given_virginica_training, X_given_virginica_test] = split(X_given_virginica);

%% Problem 2
%
problem_2

%% Problem 3
%
problem_3

%% Problem 4
%
problem_4

%% Problem 5
%
problem_5

%% Problem 6
%
problem_6
```

problem_2.m

```
% (a) and (b) PCA
% dimensions to keep
m = 2;
[~,~,~,~,W] = PCA(...
    [X_given_setosa_training; X_given_versicolor_training; X_given_virginica_training]', [], m);

X_pca_given_setosa = (W * X_given_setosa_training')';
X_pca_given_versicolor = (W * X_given_versicolor_training')';
X_pca_given_virginica = (W * X_given_virginica_training')';

X_pca_given_setosa_test = (W * X_given_setosa_test')';
X_pca_given_versicolor_test = (W * X_given_versicolor_test')';
X_pca_given_virginica_test = (W * X_given_virginica_test')';

% (c) Estimate Mu and Sigma
[Mu_setosa, Sigma_setosa] = mle(X_pca_given_setosa);
[Mu_versicolor, Sigma_versicolor] = mle(X_pca_given_versicolor);
```

```
[Mu_virginica, Sigma_virginica] = mle(X_pca_given_virginica);

% (d) MLE classification
Theta_c1 = [Mu_setosa Sigma_setosa];
Theta_c2 = [Mu_versicolor Sigma_versicolor];
Theta_c3 = [Mu_virginica Sigma_virginica];

c_1 = classify(X_pca_given_setosa_test, Theta_c1, Theta_c2, Theta_c3);
c_2 = classify(X_pca_given_versicolor_test, Theta_c1, Theta_c2, Theta_c3);
c_3 = classify(X_pca_given_virginica_test, Theta_c1, Theta_c2, Theta_c3);

% (e) confusion table
confusion_table(1, :) = [sum(c_1==1) sum(c_1==2) sum(c_1==3)];
confusion_table(2, :) = [sum(c_2==1) sum(c_2==2) sum(c_2==3)];
confusion_table(3, :) = [sum(c_3==1) sum(c_3==2) sum(c_3==3)];

display(confusion_table);

% (f) plot PCA projected data
figure;
scatter(X_pca_given_setosa_test(:,1), X_pca_given_setosa_test(:,2), 'ob'); hold on;
scatter(X_pca_given_versicolor_test(:,1), X_pca_given_versicolor_test(:,2), 'xr');
scatter(X_pca_given_virginica_test(:,1), X_pca_given_virginica_test(:,2), 'dk'); hold off;
```

problem_3.m

```
% Y - input target vector for MDA
Y = [ones(1,30) ones(1,30)*2 ones(1,30)*3];
[~,~,W] = MultipleDiscriminantAnalysis(...
    [X_given_setosa_training; X_given_versicolor_training; X_given_virginica_training]', Y);

% (a) and (b) MDA projection
X_pca_given_setosa = (W' * X_given_setosa_training')';
X_pca_given_versicolor = (W' * X_given_versicolor_training')';
X_pca_given_virginica = (W' * X_given_virginica_training')';

X_pca_given_setosa_test = (W' * X_given_setosa_test')';
X_pca_given_versicolor_test = (W' * X_given_versicolor_test')';
X_pca_given_virginica_test = (W' * X_given_virginica_test')';

% (d) Estimate Mu and Sigma
[Mu_setosa, Sigma_setosa] = mle(X_pca_given_setosa);
[Mu_versicolor, Sigma_versicolor] = mle(X_pca_given_versicolor);
[Mu_virginica, Sigma_virginica] = mle(X_pca_given_virginica);

% (e) MLE classification
Theta_c1 = [Mu_setosa Sigma_setosa];
Theta_c2 = [Mu_versicolor Sigma_versicolor];
Theta_c3 = [Mu_virginica Sigma_virginica];

c_1 = classify(X_pca_given_setosa_test, Theta_c1, Theta_c2, Theta_c3);
c_2 = classify(X_pca_given_versicolor_test, Theta_c1, Theta_c2, Theta_c3);
c_3 = classify(X_pca_given_virginica_test, Theta_c1, Theta_c2, Theta_c3);

% (f) confusion table
confusion_table(1, :) = [sum(c_1==1) sum(c_1==2) sum(c_1==3)];
confusion_table(2, :) = [sum(c_2==1) sum(c_2==2) sum(c_2==3)];
confusion_table(3, :) = [sum(c_3==1) sum(c_3==2) sum(c_3==3)];

display(confusion_table);

% (g) plot MDA projected data
figure;
scatter(X_pca_given_setosa_test(:,1), X_pca_given_setosa_test(:,2), 'ob'); hold on;
scatter(X_pca_given_versicolor_test(:,1), X_pca_given_versicolor_test(:,2), 'xr');
scatter(X_pca_given_virginica_test(:,1), X_pca_given_virginica_test(:,2), 'dk'); hold off;
```

problem_4.m

```
% Initialization
clear all; close all; clc; warning off all;

% Setup
```

```
Nfolder = 1; % # of folders
Nfile = 10; % # of files

% Read images from input{i} folder
Im = readimages( 'input' , 1:Nfolder , 'pgm', 1:Nfile);
[R, ~]=size(Im{1,1}); % # of rows and colums per image

% Convert images to vectors
Imv = mat2vec(Im);

% PCA approximation
k = [1 4 8]; % dimensions to be preserved
x_bar = mean(Imv{1}, 2); % the sample mean
[D, N] = size(Imv{1}); % # original dimensions and training samples

% (a) and (b)
% Plot a 4-by-1 subplotting system
figure;
subplot(4,1,1);
image(uint8(cell2mat(Im))); % image printing (have to convert double precision numbers into 8-bit integ
axis image;
title('Original images');
subplot_counter = 2;

for m=k
    [~,~,~,~,W] = PCA(Imv{1}, [], m);
    W = W'; % Transpose to get column Eigenvector
    Im_approximation = zeros(D, N);
    for i=1:N
        x = Imv{1}(:, i);
        Im_approximation(:, i) = pca_approximation(x, W, x_bar);
    end

    % Image Reconstructing (from data points (vectors))
    Imr = vec2mat({Im_approximation}, R);
    subplot(4,1, subplot_counter);
    image(uint8(cell2mat(Imr)));
    axis image;
    title([num2str(m) ' dimension projection images']);
    subplot_counter = subplot_counter + 1;
end
colormap(gray(256));

% (c)
% Covariance S = E[(x-x_bar) * (x-x_bar)'];
S = cov(Imv{1}');
[~, D] = eig(S);
Lambda = sort(diag(D), 'descend'); % l1 > l2 > ...

for m=k
    beta_k = sum(Lambda(1:m)) / sum(Lambda);
    e_square = sum(Lambda(m+1:end));
    display(['k:', num2str(m), ' => beta_k = ', num2str(beta_k), ', e_square = ', num2str(e_square)]);
end
```

problem_5.m

```
% Initialization
clear all; close all; clc; warning off all;

% Setup
Nfolder = 5; % # of folders (one person per folder)
Nfile = 10; % # of files

% Read images from input{i} folder
Im = readimages( 'ImageFaceID/input' , 1:Nfolder , 'pgm', 1:Nfile);
[R, C]=size(Im{1,1}); % # of rows and colums per image

% Convert images to vectors
Imv = mat2vec(Im);
X_training = [Imv{1}(:, 1:5) Imv{2}(:, 1:5) Imv{3}(:, 1:5) Imv{4}(:, 1:5) Imv{5}(:, 1:5)];
X_test = [Imv{1}(:, 6:10) Imv{2}(:, 6:10) Imv{3}(:, 6:10) Imv{4}(:, 6:10) Imv{5}(:, 6:10)];

% PCA approximation
```

```
M = [1 4 10]; % dimensions to be preserved
K = [1 3 5]; % # of neighbors in Knn
% Both the training targets (labels) and test targets
% since our training and set are order in symmetry
targets = [ones(1,5) ones(1,5)*2 ones(1,5)*3 ones(1,5)*4 ones(1,5)*5];
targets_size = length(targets);
error_rate_table = ones(3, 3);

row = 1;
col = 1;
for m=M
    [~,~,~,~,W] = PCA(X_training, [], m);
    X_training_pca = W * X_training;
    X_test_pca = W * X_test;

    for k=K
        classification = Nearest_Neighbor(X_training_pca, targets, X_test_pca, k);
        error_rate_table(row, col) = 1 - (sum(classification == targets) / targets_size);
        row = row + 1;
    end
    col = col + 1;
    row = 1;
end

display(error_rate_table);
```

problem_6.m

```
% Initialization
clc; clear all; close all;

% Load data
SalmonLightness = load('SalmonLightness.dat');
SeabassLightness = load('SeabassLightness.dat');

Total = length(SalmonLightness) + length(SeabassLightness);
P_salmon = length(SalmonLightness) / Total;
P_seabass = length(SeabassLightness) / Total;

H = [0.8 0.2];
x_range = 0:0.1:12;

for h=H
    p_lightness_given_salmon = parzen_window(SalmonLightness, x_range, h);
    p_lightness_given_seabass = parzen_window(SeabassLightness, x_range, h);
    p_lightness = p_lightness_given_salmon * P_salmon + ...
        p_lightness_given_seabass * P_seabass;

    figure;
    plot(x_range, p_lightness_given_salmon, 'r'); hold on;
    plot(x_range, p_lightness_given_seabass, '--k');
    plot(x_range, p_lightness, '-.g'); hold off;
    legend('P(lightness|salmon)', 'P(lightness|seabass)', 'P(lightness)');
    title(['PDFs with h=', num2str(h)]);
end
```

classify.m

```
function [ c ] = classify( X, Theta_1, Theta_2, Theta_3 )
% classify - Classify X given Theta {1 2 and 3}
%   by comparing the value of discriminant function g(x)
%   for each parameter theta.
%
% Return:
%   c - classification vector
%
% where value of
%   c = 1 (Iris-setosa)
%   c = 2 (Iris-versicolor)
%   c = 3 (Iris-virginica)
%

    [r, ~] = size(X);
```

```
    c = zeros(r, 1);

    mu_column = 1;
    sigma_column = 2:3;

    for k=1:r
        x = X(k, :)';
        [~, c(k,:)] = max([
            g_mle(x, Theta_1(:, mu_column), Theta_1(:, sigma_column)) ...
            g_mle(x, Theta_2(:, mu_column), Theta_2(:, sigma_column)) ...
            g_mle(x, Theta_3(:, mu_column), Theta_3(:, sigma_column)) ]);
    end
end
```

g_mle.m

```
function [ g ] = g_mle( x, mu, Sigma )
% g_mle - Discriminant function 'g' for Maximum Likelihood Estimation
%   Compute log( P(x|w) )
%
% Input:
%   mu - mean of P(x|w)
%   Sigma - covariance matrix of P(x|w)

    d = length(x);
    x_tilde = x - mu;
    g = - d/2*log(2*pi) ...
        - 1/2*log(det(Sigma)) ...
        - 1/2*x_tilde'/Sigma*x_tilde;
end
```

mle.m

```
function [ m, P ] = mle( dataset )
% mle - Maximum likelihood estimator for mean and covariance
%       of 1-D and 2-D Gaussian dataset
%
%   m : the estimated mean (sample mean)
%   P : the estimated biased variance for 1-D dataset
%           and covariance matrix for 2-D dataset
%
% Note:
%   P = [var1 cov(1,2); cov(1,2) var2]
%
% where
%   var1        - biased variance of x1
%   cov(1, 2)   - E[(x1-mean_x1)(x2-mean_x2)]
%   var2        - biased variance of x2
%
    m = mean(dataset)';
    P = cov(dataset, 1);
end
```

parzen_window.m

```
function [ pdf ] = parzen_window( X, x_range, h )
% PARZEN_WINDOW density estimation

    pdf = zeros(length(x_range), 1);
    for i=1:length(x_range)
        x = x_range(i);
        delta = (1/sqrt(2*pi)) .* exp(-((x-X)/h).^2 / 2);
        pdf(i, 1) = sum(delta) / (length(X) * h);
    end
end
```

pca_approximation.m

```
function [ x_tilda ] = pca_approximation( x, W, x_bar )
% pca_approximation - PCA approximation of x given Eigenvector W

    [~, M] = size(W); % # of preserved dimensions
    distance = x - x_bar;
```

```
    x_tilda = zeros(size(x,1), 1);

    for i=1:M
        x_tilda = x_tilda + (W(:, i)' * distance) * W(:, i);
    end
    x_tilda = x_tilda + x_bar;
end
```

split.m

```
function [ training, test ] = split( dataset )
% split - Divide the dataset into trainset and testset
%   Training set : row 11 to 40
%   Test set     : row 1 to 10 and 41 to 50

    training = dataset(11:40, :);
    test = dataset([1:10 41:50], :);
end
```

strcomp.m

```
function [ y ] = strcomp( x, str )
% strcomp - Compare each row of 'x' with 'str'
%   Return 1 for the rows that equal to 'str' and 0 otherwise

    [r, c] = size(x);

    y = zeros(r, 1);
    for k=1:r
        y(k) = sum(x(k,:) == str);
    end
    y = y == c;
end
```