Silvia Ionescu
EC503: Learning from Data October 2, 2016

Assignment 3

Part 3.1a

Matlab helper functions were written for testing and training using Quadratic Discriminant Analysis(QDA) and Linear Discriminant Analysis (LDA) classifiers according to Eq. 1 and Eq.2 shown below.

$$h_{QDA}(x) = argmin_k \frac{1}{2}(x-u_k)^T\Sigma_k^{-1}(x-u_k) + \frac{1}{2}\ln\det(\Sigma_k) - \ln(\alpha_k) \qquad (1)$$

$$h_{LDA}(x) = argmax_k \, [(u_k^T\Sigma^{-1})x - \frac{1}{2}u_k^T\Sigma^{-1}u_k + \ln\alpha_k \qquad (2)$$

Where u_k is the mean vector of a specific class, Σ_k is the covariance matrix of a specific class; α_k is the prior class probability. For LDA all classes have the same convergence matrix $\Sigma_k = \Sigma$ for all k.

Part 3.1b

The data_iris.mat dataset was used. This set contained 150 classified samples with 4 features per sample. The training set was created by generating 100 uniformly random (without replacement) numbers between 1 and 150. The remaining samples were used for the testing set. This was repeated 10 times for 10 different training/test splits.
The performance evaluation is presented below.

Mean vectors for training samples for each class averaged over 10 splits – common for both LDA and QDA:

	Features			
Classes	5.01	3.44	1.46	0.24
	5.93	2.77	4.27	1.33
	6.60	2.99	5.57	2.04
	5.01	3.44	1.46	0.24

Table 1: QDA and LDA mean vectors for training samples for each class (LDA and QDA have the same mean vectors).

The variances of all the 4 dimensions for each class of training set averaged over 10 splits.

	Features			
Classes	0.12	0.14	0.03	0.01
	0.25	0.10	0.22	0.04
	0.38	0.10	0.28	0.08

Table2: Variances of all 4 features/dimensions for each class of the QDA training set averaged over 10 splits.

The variances of all the 4 dimensions in LDA averaged over 10 splits.

	Features			
Classes	0.25	0.12	0.18	0.04

Table 3: Variances for LDA averaged across 10 splits

The mean and standard deviation of all 10 test CCR's

	CCR mean	CCR Standard Deviation
QDA	0.982	0.0180
LDA	0.980	0.025

Table 4: Mean and standard deviation of all 10 CCR's

Confusion matrices with the best and worst CCR in LDA.

<u>Confusion matrix for best LDA CCR</u>

	Truth		
Decision	16	0	0
	0	19	0
	0	0	15

<u>Confusion matrix for worst LDA CCR</u>

	Truth		
Decision	16	0	0
	0	13	0
	0	4	17

Table 5: Confusion matrices for best CCR (left) and worst CCR (right).

<u>Part 3.1c</u>

The data_cancer.mat dataset was used. The dimension of data samples (4000) << dimension of features (216). As a result, the estimates of the covariance matrices in LDA/QDA will be singular. Regularized Discriminant Analysis (RDA) can be used in this case in combination with LDA or QDA. For this part the covariance matrix in LDA was replaced by:

$$\hat{\Sigma}_{reg} = \lambda\, diag\left(\hat{\Sigma}_{LDA}\right) + (1-\lambda)\hat{\Sigma}_{LDA}$$

<u>Part 3.1d</u>

The samples provided were split into 150 samples for the training set and 66 samples for the test set. Lambda, λ, was chosen in the range of [0.1:0.05:1]. The random seed was fixed. As shown in the graph below, λ between 0.7 and 0.9 gives the best CCR for the test set.

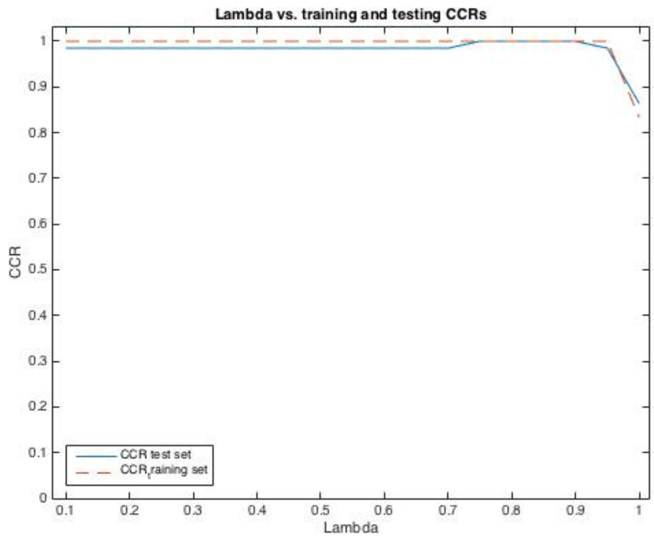

Figure 1: Training and test sets CCRs vs. lambda λ

Part 3.2

Applied k-Nearest Neighbor classifier based on the Euclidian distance to two datasets.

Part 3.2a

Scatter plot of all the training data in data_kkSimulation.mat is shown below:

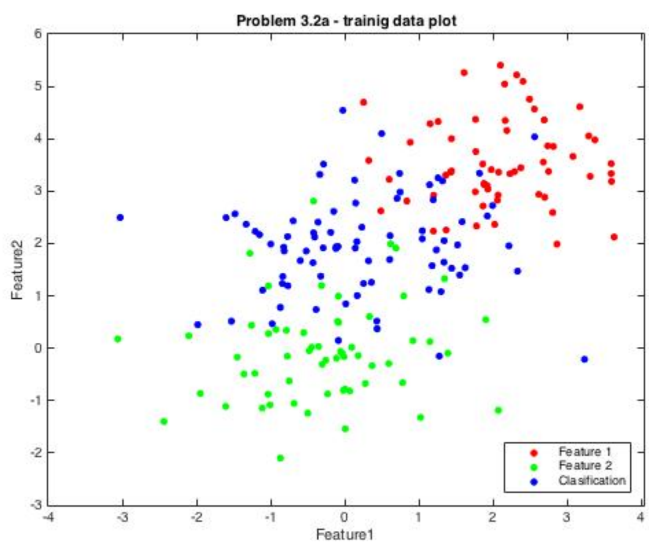

Figure 2: Scatter plot of all training data in data_kkSimulation.mat dataset.

Part 3.2b

Created a 2D grid [-3.5:0.1:6]x[-3:0.1:6.5] and calculated its probability of being class 2 using k = 10 nearest neighbors. As well as the probability of being class 3 using k = 10 nearest neighbors. The plots are shown below.

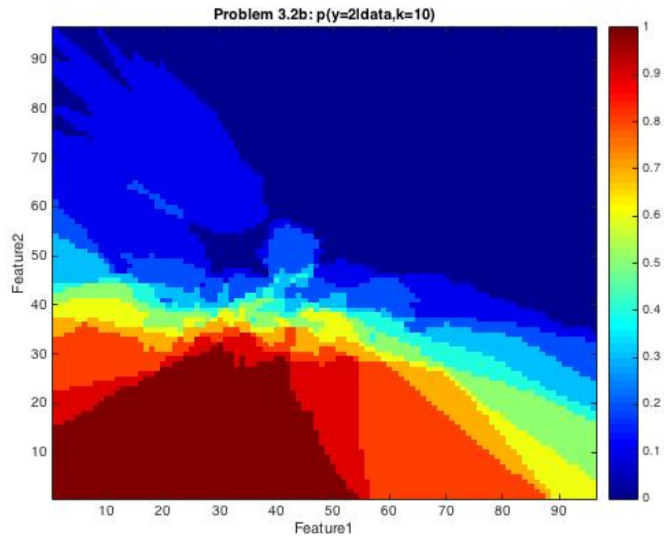

Figure 3: Probability of being class 2 using k = 10 nearest neighbors.

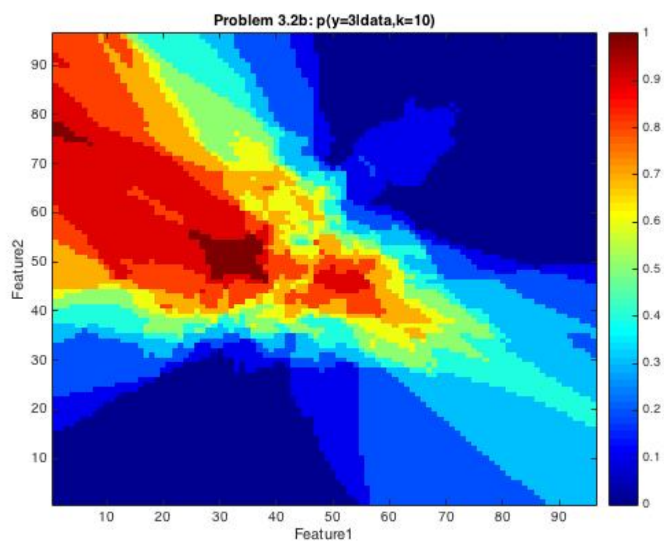

Figure 4: Probability of being class 3 using k = 10 nearest neighbors.

Part 3.2c

For the 2D grid, created in Part 3.2b, the classification for all its points was performed using k-NN classifier with k = 1 and k = 5 (plots shown below). For k = 1 we are looking only at the class of our nearest neighbor. For k = 5 we are looking at out 5 nearest neighbors, determine the class with the highest probability and assign this class to out test point. For k = 1 we are looking at a local area, while for k = 5 we are looking at out data a bit more globally. As the number of k nearest neighbors considered increases the predicted class of the sample converges to the true label of the sample.

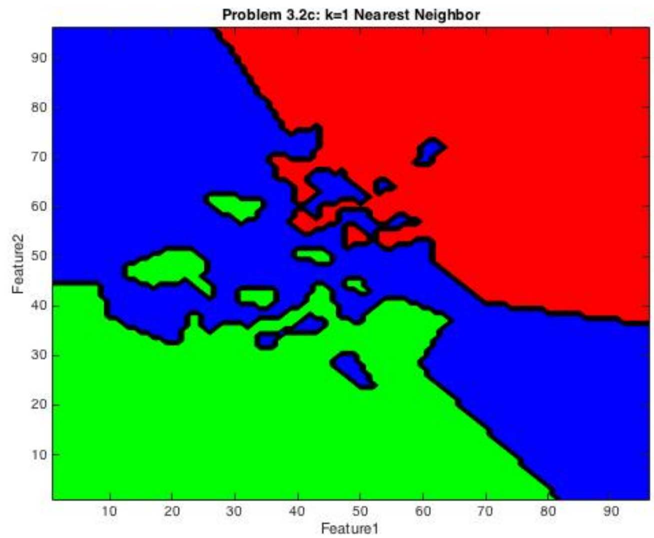

Fig. 4: Class prediction for 2D grid using k = 1 Nearest Neighbor Classifier

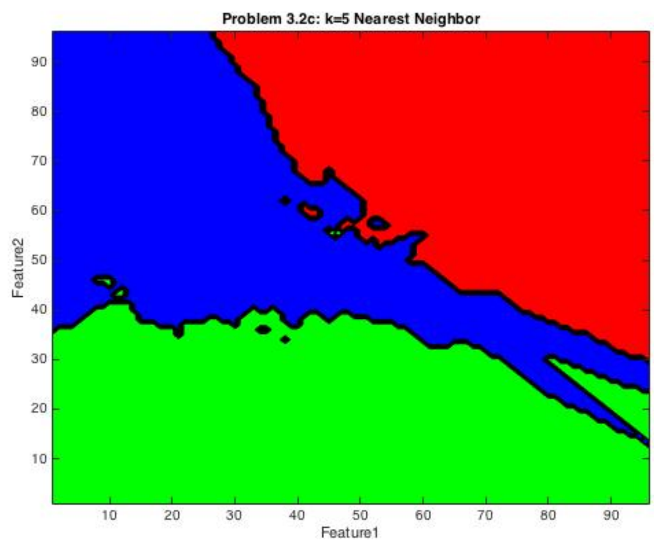

Fig. 5: Class prediction for 2D grid using k = 5 Nearest Neighbor Classifier.

Part 3.2d

Used data.mnist_train.mat and data_mnist_test.mat data sets. Calculated the 1-Nearest Neighbor classifier for this data, by using the formula shown below, where x are the test samples and x' are the training samples.

$$||x - x'||_2^2 = < x, x > -2 < x, x' > + < x', x' > \quad (4)$$

The **CCR is 0.9691** and the confusion matrix is presented below.

	Truth									
Decision	973	0	7	0	0	1	4	0	6	2
	1	1129	6	1	7	1	2	14	1	5
	1	3	992	2	0	0	0	6	3	1
	0	0	5	970	0	12	0	2	14	6
	0	1	1	1	944	2	3	4	5	10
	1	1	0	19	0	860	5	0	13	5
	3	1	2	0	3	5	944	0	3	1
	1	0	16	7	5	1	0	992	4	11
	0	0	3	7	1	6	0	0	920	1

Table 6: Confusion matrix for part 3.2d – data.mnist_train.mat and data_mnist_test.mat data sets.

```
function [QDAmodel]= sionescu_QDA_train(X_train, Y_train, numofClass)
%
% Training QDA
% EC 503 Learning from Data
```

```
% Gaussian Discriminant Analysis
%
% Assuming D = dimension of data
% Inputs :
% X_train : training data matrix, each row is a training data point
% Y_train : training labels for rows of X_train
% numofClass : number of classes
%
% Assuming that the classes are labeled  from 1 to numofClass
% Output :
% QDAmodel : the parameters of QDA classifier which has the following fields
% QDAmodel.Mu : numofClass * D matrix, i-th row = mean vector of class i
% QDAmodel.Sigma : D * D * numofClass array, Sigma(:,:,i) = covariance matrix
of class i
% QDAmodel.Pi : numofClass * 1 vector, Pi(i) = prior probability of class i
% Silvia Ionescu
% Date: 10-2-2016
% determine the length of the training set
X_train_size = size(X_train,1);

% pull out the features for class 1, 2, 3 and place it in a cell array
class1_X = X_train(find(Y_train == 1),:);
class2_X = X_train(find(Y_train == 2),:);
class3_X = X_train(find(Y_train == 3),:);
class = {class1_X, class2_X, class3_X};

% assign variable dimentions, to be used later in the code
Mu = zeros(numofClass, size(X_train,2));
Sigma = zeros(size(X_train,2),size(X_train,2),numofClass);
Pi = zeros(numofClass,1);

% calculate mean vector, covariance matrix, and prior class probabiliy
for c = 1: numofClass
    label = class{c};
    % iterate over features of the samples in a class
    for f = 1: size(label,2)
        % calculate the mean
        Mu(c,f) = sum(label(:,f))/size(label,1);
    end
    % calculate covariance per class
    Sigma(:,:,c) = cov(label);
    % determine prior class probability
    Pi(c) = size(label,1)/X_train_size;
end
% build the QDAmodel struct
Pi = Pi';
QDAmodel = struct('Mu', Mu, 'Sigma', Sigma, 'Pi', Pi);

end



function [Y_predict] = sionescu_QDA_test(X_test, QDAmodel, numofClass)
%
% Testing for QDA
%
% EC 503 Learning from Data
```

```
% Gaussian Discriminant Analysis
%
% Assuming D = dimension of data
% Inputs :
% X_test : test data matrix, each row is a test data point
% numofClass : number of classes
% QDAmodel: the parameters of QDA classifier which has the following fields
% QDAmodel.Mu : numofClass * D matrix, i-th row = mean vector of class i
% QDAmodel.Sigma : D * D * numofClass array, Sigma(:,:,i) = covariance
% matrix of class i
% QDAmodel.Pi : numofClass * 1 vector, Pi(i) = prior probability of class i
% Assuming that the classes are labeled  from 1 to numofClass
% Output :
% Y_predict predicted labels for all the testing data points in X_test
% Silvia Ionescu
% Date: 10-2-2016

% Input data mean, covariance matrix, and prior probability
Sigma = QDAmodel.Sigma;
Mu = QDAmodel.Mu;
Pi = QDAmodel.Pi;

% inverse of the covariance matrix
for i = 1: numofClass
    Sigma_inv(:,:,i) = inv(Sigma(:,:,i));
end

% assign variable dimentions, to be used later in the code
a = zeros(size(X_test,1), numofClass, 1);

% applying the QDA model
for j = 1:size(X_test,1)
    for s = 1: numofClass
        a(j,s,:) = (1/2)*(X_test(j,:) - Mu(s,:)) * Sigma_inv(:,:,s) *
(X_test(j,:) - Mu(s,:))' + (1/2)*log(det(Sigma(:,:,s))) - log(Pi(s));
    end
end

% finding the min value
min_val = min(a,[],2);
Y_predict = zeros(size(X_test,1),1);

% determine test sample classification
for z = 1:size(min_val,1)
   Y_predict(z,:) = find(a(z,:) == min_val(z,1));
end
end




function [LDAmodel] = sionescu_LDA_train(X_train, Y_train, numofClass)

% Training LDA
% EC 503 Learning from Data
% Gaussian Discriminant Analysis
```

```
%
% Assuming D = dimension of data
% Inputs :
% X_train : training data matrix, each row is a training data point
% Y_train : training labels for rows of X_train
% numofClass : number of classes
%
% Assuming that the classes are labeled  from 1 to numofClass
% Output :
% LDAmodel : the parameters of LDA classifier which has the following fields
% LDAmodel.Mu : numofClass * D matrix, i-th row = mean vector of class i
% LDAmodel.Sigmapooled : D * D  covariance matrix
% LDAmodel.Pi : numofClass * 1 vector, Pi(i) = prior probability of class i
% Silvia Ionescu
% Date: 10-2-2016

% determine the length of the training set
X_train_size = size(X_train,1);

% pull out the features for class 1, 2, 3 and place it in a cell array
class1_X = {X_train(find(Y_train == 1),:),find(Y_train == 1)};
class2_X = {X_train(find(Y_train == 2),:),find(Y_train == 2)};
class3_X = {X_train(find(Y_train == 3),:),find(Y_train == 3)};
class = {class1_X, class2_X, class3_X};

% assign variable dimentions, to be used later in the code
Mu = zeros(numofClass, size(X_train,2));
%Sigma = zeros(size(X_train,2),size(X_train,2),numofClass);
Pi = zeros(numofClass,1);

% calculate mean vector, covariance matrix, and prior class probabiliy
for c = 1: numofClass
    label = class{c};
    sublabel = label{1};
    sublabel_index = label{2};

    % iterate over features of the samples in a class
    for f = 1: size(sublabel, 2)
        % calculate the mean
        Mu(c,f) = sum(sublabel(:,f))/size(sublabel,1);

        % difference between the training samples and mean per class
        sublabel(:,f) = sublabel(:,f) - Mu(c,f);
    end

    class_diff(:,c) = {sublabel, sublabel_index};

    % determine prior class probability
    Pi(c) = size(sublabel,1)/X_train_size;
end

% recombine classes into a sample set of the original training set size
sample_set_diff = [];
for i = 1:size(class_diff, 2)
    first = class_diff(1,i);
    second = class_diff(2,i);
```

```
    sample_cat = [first{1}, second{1}];
    sample_set_diff = [sample_set_diff;sample_cat];
end

% sort the set according to the original indices
sample_diff_sort = sortrows(sample_set_diff,size(sample_set_diff,2));

% calculate the covariance matrix
sample_diff_for_cov = sample_diff_sort(:,1:(end-1));
Sigmapooled = (sample_diff_for_cov'*sample_diff_for_cov)/(length(X_train)-
numofClass);

% build the LDAmodel struct
LDAmodel = struct('Mu', Mu, 'Sigmapooled', Sigmapooled, 'Pi', Pi);

end
```

```
function [Y_predict] = sionescu_LDA_test(X_test, LDAmodel, numofClass)
%
% Testing for LDA
% EC 503 Learning from Data
% Gaussian Discriminant Analysis
%
```

```
% Assuming D = dimension of data
% Inputs :
% X_test : test data matrix, each row is a test data point
% numofClass : number of classes
% LDAmodel : the parameters of LDA classifier which has the follwoing fields
% LDAmodel.Mu : numofClass * D matrix, i-th row = mean vector of class i
% LDAmodel.Sigmapooled : D * D  covariance matrix
% LDAmodel.Pi : numofClass * 1 vector, Pi(i) = prior probability of class i
% Assuming that the classes are labeled  from 1 to numofClass
% Output :
% Y_predict predicted labels for all the testing data points in X_test

% Silvia Ionescu
% Date: 10-2-2016

Sigma = LDAmodel.Sigmapooled;
Mu = LDAmodel.Mu;
Pi = LDAmodel.Pi;

% inverse of the covariance matrix
Sigma_inv = inv(Sigma);

% calculating the LDADA model
for j = 1:size(X_test,1)
    for s = 1: numofClass
        a(j,s,:) = Mu(s,:) * Sigma_inv * X_test(j,:)' - (1/2)*Mu(s,:) *
Sigma_inv * Mu(s,:)' + log(Pi(s));
    end
end

% finding the min value of a
max_val = max(a,[],2);

% determine test sample classification
Y_predict = zeros(size(X_test,1),1);
for z = 1:size(max_val,1)
   Y_predict(z,:) = find(a(z,:) == max_val(z,1));
end

end
```

```
% Assignment3_1b.m
% Silvia Ionescu
% 10-2-2016

% Description:
% Data set data_iris.mat was used with X = 150 samples x 4 features
% and Y - classifications for this 150 samples.
```

```
% This set was divided into 100 training samples and 50 test samples.
% The numbers were picked uniformlu at random.
% For each split training, testing, and performance evaluation was
% performed.

clear; close all; clc;
load('data_iris.mat')

% calcualte the sample size and number of classes
samples = 1:size(X,1);
numofClass = Y(size(X,1),1);

% assign variable dimentions, to be used later in the code
split_mean = zeros(numofClass, size(X,2),10);

QDA_Mu_10avg = zeros(numofClass, size(X,2));
LDA_Mu_10avg = zeros(numofClass, size(X,2));

variance = zeros(numofClass, size(X,2),10);
QDA_var_avg = zeros(numofClass, size(X,2));
LDA_var_avg = zeros(1, size(X,2));

QDA_confusion = zeros(numofClass,numofClass,10);
LDA_confusion = zeros(numofClass,numofClass,10);

% generate 10 splits
for i=1:10
    % fixing the random seed
    s = RandStream('mt19937ar','Seed',i-1);

    % generate 100 random numbers between 1-150
    sample_train = randperm(s,size(X,1),100);

    % create the train set by using the 100 random numbers
    X_train = X(sample_train,:);
    Y_train = Y(sample_train,:);

    % create the test set by taking the remaining samles
    sample_test = setdiff(samples, sample_train);
    X_test = X(sample_test,:);
    Y_test = Y(sample_test,:);

    % QDA Model test and train
    QDAmodel = sionescu_QDA_train(X_train, Y_train, numofClass);
    QDA_Y_predict = sionescu_QDA_test(X_test, QDAmodel, numofClass);

    % LDA Model test and train
    LDAmodel = sionescu_LDA_train(X_train, Y_train, numofClass);
    LDA_Y_predict = sionescu_LDA_test(X_test, LDAmodel, numofClass);

    % add up the mean accross 10 splits for QDA and LDA
    QDA_Mu_10avg = QDA_Mu_10avg + QDAmodel.Mu;
    LDA_Mu_10avg = LDA_Mu_10avg + LDAmodel.Mu;

    % calcualte the QDA Sigma summed over 10 splits per class
```

```
    QDA_Sigma = QDAmodel.Sigma;
    for j = 1:numofClass
        variance(j,:,i) = diag(QDA_Sigma(:,:,j))';
    end
    QDA_var_avg = QDA_var_avg + variance(:,:,i);

    % calcualte the LDA Sigma summed over 10 splits
    LDA_Sigma = diag(LDAmodel.Sigmapooled)';
    LDA_var_avg = LDA_var_avg + LDA_Sigma;

    % QDA confusion matrix
    QDA_confusion(:,:,i) = confusionmat(Y_test,QDA_Y_predict);
    QDA_CCR(1,i) = sum(diag(QDA_confusion(:,:,i)))/
sum(sum(QDA_confusion(:,:,i)));

    % LDA confusion matrix
    LDA_confusion(:,:,i) = confusionmat(Y_test,LDA_Y_predict);
    LDA_CCR(1,i) = sum(diag(LDA_confusion(:,:,i)))/
sum(sum(LDA_confusion(:,:,i)));
end

% 10 splits average mean per class for QDA and LDA
QDA_Mu_10avg = QDA_Mu_10avg/10;
LDA_Mu_10avg = LDA_Mu_10avg/10;

% 10 splits average variance per class for QDA(3x4), LDA(1x4)
QDA_var_avg = QDA_var_avg/10;
LDA_var_avg = LDA_var_avg/10;

% mean of all 10 test CCR's
QDA_CCR_mean = sum(QDA_CCR)/10;
LDA_CCR_mean = sum(LDA_CCR)/10;

% standard deviation accross all 10 CCR's
QDA_CCR_sd = std(QDA_CCR);
LDA_CCR_sd = std(LDA_CCR);
```

```
function [RDAmodel]= sionescu_RDA_train(X_train, Y_train,gamma, numofClass)
%
% Training RDA
%
% EC 503 Learning from Data
% Gaussian Discriminant Analysis
%
% Assuming D = dimension of data
```

```
% Inputs :
% X_train : training data matrix, each row is a training data point
% Y_train : training labels for rows of X_train
% numofClass : number of classes
%
% Assuming that the classes are labeled  from 1 to numofClass
% Output :
% RDAmodel : the parameters of RDA classifier which has the following fields
% RDAmodel.Mu : numofClass * D matrix, i-th row = mean vector of class i
% RDAmodel.Sigmapooled : D * D  covariance matrix
% RDAmodel.Pi : numofClass * 1 vector, Pi(i) = prior probability of class i
% Silvia Ionescu
% 10-2-2016

% Finding Sigma LDA
X_train_size = size(X_train,1);

class1_X = {X_train(find(Y_train == 0),:),find(Y_train == 0)};
class2_X = {X_train(find(Y_train == 1),:),find(Y_train == 1)};
class = {class1_X, class2_X};

% calculate mean
Mu = zeros(numofClass, size(X_train,2));
%C1 = zeros(size(class1_X,1), size(class1_X,2));
%Sigma = zeros(size(X_train,2),size(X_train,2),numofClass);
Pi = zeros(numofClass,1);

for c = 1: numofClass
    label = class{c};
    sublabel = label{1};
    sublabel_index = label{2};
    for f = 1: size(sublabel, 2)
        % 3x4 matrix - mean
        Mu(c,f) = sum(sublabel(:,f))/size(sublabel,1);
        sublabel(:,f) = sublabel(:,f) - Mu(c,f);
    end

    class_diff(:,c) = {sublabel, sublabel_index};
    Pi(c) = size(sublabel,1)/X_train_size;
end

sample_set_diff = [];
for i = 1:size(class_diff, 2)
    first = class_diff(1,i);
    second = class_diff(2,i);
    sample_cat = [first{1}, second{1}];
    sample_set_diff = [sample_set_diff;sample_cat];
end

sample_diff_sort = sortrows(sample_set_diff,size(sample_set_diff,2));
sample_diff_for_conv = sample_diff_sort(:,1:(end-1));
Sigma = (sample_diff_for_conv'*sample_diff_for_conv)/(length(X_train)-
numofClass);

% ------- Have Sigma LDA -----
```

```
% apply regularized discriminant analysis (RDA)
for g = 1: size(gamma,2)
    Sigmapooled(:,:,g) = gamma(g)*diag(diag(Sigma)) + (1-gamma(g))*Sigma;
end
RDAmodel = struct('Mu', Mu, 'Sigmapooled', Sigmapooled, 'Pi', Pi);
end
```

```
function [Y_predict] = sionescu_RDA_test(X_test, RDAmodel, numofClass)
%
% Testing for RDA
%
% EC 503 Learning from Data
% Gaussian Discriminant Analysis
%
% Assuming D = dimension of data
% Inputs :
```

```
% X_test : test data matrix, each row is a test data point
% numofClass : number of classes
% RDAmodel : the parameters of RDA classifier which has the following fields
% RDAmodel.Mu : numofClass * D matrix, i-th row = mean vector of class i
% RDAmodel.Sigmapooled : D * D  covariance matrix
% RDAmodel.Pi : numofClass * 1 vector, Pi(i) = prior probability of class i
%
% Assuming that the classes are labeled  from 1 to numofClass
% Output :
% Y_predict predicted labels for all the testing data points in X_test

% Write your code here:



Sigma = RDAmodel.Sigmapooled;
Mu = RDAmodel.Mu;
Pi = RDAmodel.Pi;



for g = 1: size(Sigma, 3)
   for s = 1: numofClass
        Mu1_rep = repmat(Mu(s,:),size(X_test,1),1);
        a = Mu1_rep/Sigma(:,:,g) * X_test' - (1/2)*Mu1_rep/Sigma(:,:,g) *
Mu1_rep' + log(Pi(s));
        value_per_class(:,s) = diag(a);
   end


    max_val = max(value_per_class,[],2);

    %Y_predict = zeros(size(X_test,1),1);
    for z = 1:size(max_val,1)
       Y_predict(z,g) = find(value_per_class(z,:) == max_val(z,1))-1;
    end

end

end
```

```
% Assignment3_2abc.m
% Silvia Ionescu
% 10-2-2016

% Description: Nearest Neighbor Classifier
% Problem 3.2a
% Fig 1: Scatter plot of all the training data in data_knnSimulation.mat
%
```

```
% Problem 3.2b
% Fig.2 shows the probabilities of being class 2 using k = 10
% Fig.3 shows the probabilities of being class 3 using k = 10
%
% Problem 3.2c
% Fig. 4 prediction of class using k = 1 NN
% Fig. 5 prediction of class using k = 5 NN

clear; close all; clc;

load('data_knnSimulation.mat')

% plot of all the training data in data_knnSimulation.mat
figure(1);
gscatter(Xtrain(:,1), Xtrain(:,2), ytrain,[1 0 0; 0 1 0; 0 0 1]);
legend('Feature 1', 'Feature 2', 'Clasification','Location','southeast');
title('Problem 3.2a - trainig data plot');
xlabel('Feature1');
ylabel('Feature2');

% create a 2D 96x96 matrix of test points
x = -3.5:.1:6;
y = -3:.1:6.5;
[A B] = meshgrid(x,y);



class1 = 0;
class2 = 0;
class3 = 0;
c1 = 1;
c2 = 2;
c3 = 3;
for i = 1:length(A);
    for j = 1:length(B);
        for k = 1:size(Xtrain,1)
            % calculate Euclidian distance
            d = sqrt((A(i,j)-Xtrain(k,1))^2 + (B(i,j)-Xtrain(k,2))^2);
            class = ytrain(k,1);
            % contains distance and the according class of the neighbor
            dis_class(k,:) = [d class];
        end

        % sort the array for each point in the 96x96 grid
        euclid_dis = sortrows(dis_class,1);
        % keep only the shorthest 10 distances and their class
        euclid_dis = euclid_dis(1:10,:);

        % calculate the class for k = 10 NNC
        class1 = 0;
        class2 = 0;
        class3 = 0;
        for p = 1:10
            test = euclid_dis(p,2);
            if (euclid_dis(p,2) == c1)
                class1 = class1 + 1;
            elseif (euclid_dis(p,2) == c2)
                class2 = class2 + 1;
```

```
            else

                class3 = class3 + 1;
            end

        end

        % probabilities of the nearest neighbors in terms of classes
        prob_10nn = [class1 c1; class2 c2; class3 c3];
        class_sort = sortrows(prob_10nn,2);
        prob_class2(i,j) = class_sort(2,1)/10;
        prob_class3(i,j) = class_sort(3,1)/10;

        % determine k = 1 NN classifier
        classification_k1(i,j) = euclid_dis(1,2);

        % calculate the k = 5 majority of neighbors
        freq1_5nn = 0;
        freq2_5nn = 0;
        freq3_5nn = 0;

        for t = 1:5
            if (euclid_dis(t,2) == c1)
                freq1_5nn = freq1_5nn + 1;
            elseif (euclid_dis(t,2) == c2)
                freq2_5nn = freq2_5nn + 1;
            else
                freq3_5nn = freq3_5nn + 1;
            end
        end

        freq_5nn = [freq1_5nn c1; freq2_5nn c2; freq3_5nn c3];
        freq_5nn_sort = sortrows(freq_5nn, 1);
        classification_k5(i,j) = freq_5nn_sort(3,2);

    end
end

% plot the probabilities of being class 2 using k = 10
figure(2);
imagesc(prob_class2);
colormap jet;
axis('xy');
title('Problem 3.2b: p(y=2|data,k=10)');
xlabel('Feature1');
ylabel('Feature2');
colorbar;

% plot the probabilities of being class 3 using k = 10
figure(3);
imagesc(prob_class3);
colormap jet;
axis('xy');
title('Problem 3.2b: p(y=3|data,k=10)');
xlabel('Feature1');
ylabel('Feature2');
```

```
colorbar;

% plot the prediction of class using k = 1 NN
figure(4);
contourf(classification_k1);
map = [1 0 0; 0 1 0; 0 0 1];
colormap(map);
title('Problem 3.2c: k=1 Nearest Neighbor');
xlabel('Feature1');
ylabel('Feature2');

% plot the prediction of class using k = 5 NN
figure(5);
contourf(classification_k5);
map = [1 0 0; 0 1 0; 0 0 1];
colormap(map);
title('Problem 3.2c: k=5 Nearest Neighbor');
xlabel('Feature1');
ylabel('Feature2');
```

```
% Assignment3_2d.m
% Silvia Ionescu
% 10-2-2016

% Description: Nearest Neighbor Classifier
% Problem 3.2d
% Apply 1-NNC to the two loaded datasets and display the CCRs


clear; close all; clc;
```

```
% train and test datasets
load('data_mnist_train.mat');
load('data_mnist_test.mat');

% calculating <x,x> for the traing set
X_train_norm = sum(X_train.^2,2);
X_train_norm1 = X_train_norm*ones(1,1000);
X_train_norm1 = X_train_norm1';


c = 0;
for i = 1:1000:size(X_test,1)
    c = c + 1;
    X_test_part = X_test(i:(i+999),:);

    % calculating <x',x'> for the test set
    X_test_norm = sum(X_test_part.^2,2);
    X_test_norm1 = X_test_norm * ones(1,60000);

    % calculating distance
    distance =  X_test_norm1 -2*(X_test_part*X_train') + X_train_norm1;
    [M,I(:,c)] = min(distance,[],2);
    class(:,c) = Y_train(I(:,c),1);
end

% determine confusion matrix and CCR
sample_class = class(:);
confusion_matrix = confusionmat(sample_class, Y_test);
CCR = sum(diag(confusion_matrix))/sum(sum(confusion_matrix));
disp(['Problem3.2d CCR: ', num2str(CCR)])
```