**Occupancy Detection**

Jinto Jose (01677777)

**1. Logistic Regression:**

logisticregression.m

%% Initialization

clear ; close all;

%% Load Data

data = load('datatraining.txt');

X = data(:, [1, 2, 3, 4, 5]);

y = data(:, 6);

%% ==================== Part 1: Plotting ====================

% We start the exercise by first plotting the data to understand the

% the problem we are working with.

%fprintf(['Plotting data with + indicating (y = 1) examples and o indicating (y = 0) examples.\n']);

%plotMultiData(X, y);

fprintf('\nProgram paused. Press enter to continue.\n');

pause;

%% ============ Part 2: Compute Cost and Gradient ============

% In this part of the exercise, you will implement the cost and gradient

% for logistic regression. You neeed to complete the code in

% costFunction.m

% Sigmoid function test

fprintf('Sigmoid function test: \n');

fprintf(' %f \n', sigmoid([-1000; 0; 100]));

fprintf('\nProgram paused. Press enter to continue.\n');

pause;

% Setup the data matrix appropriately, and add ones for the intercept term

[m, n] = size(X);

% Add intercept term to x and X\_test

X = [ones(m, 1) X];

% Initialize fitting parameters

initial\_theta = zeros(n + 1, 1);

% Compute and display initial cost and gradient

[cost, grad] = costFunction(initial\_theta, X, y);

fprintf('Cost at initial theta (zeros): %f\n', cost);

fprintf('Gradient at initial theta (zeros): \n');

fprintf(' %f \n', grad);

fprintf('\nProgram paused. Press enter to continue.\n');

pause;

%% ============= Part 3: Optimizing using fminunc =============

% In this exercise, you will use a built-in function (fminunc) to find the

% optimal parameters theta.

% Set options for fminunc

options = optimset('GradObj', 'on', 'MaxIter', 400);

% Run fminunc to obtain the optimal theta

% This function will return theta and the cost

[theta, cost] = ...

fminunc(@(t)(costFunction(t, X, y)), initial\_theta, options);

% Print theta to screen

fprintf('Cost at theta found by fminunc: %f\n', cost);

fprintf('theta: \n');

fprintf(' %f \n', theta);

fprintf('\nProgram paused. Press enter to continue.\n');

pause;

%% ============== Part 4: Predict and Accuracies ==============

% After learning the parameters, you'll like to use it to predict the outcomes

% on unseen data.

%

% Furthermore, you will compute the training and test set accuracies of

% our model.

%% ============== Test data set 1 ==============

testdata = load('datatest.txt');

testX = testdata(:, [1, 2, 3, 4, 5]);

testX = [ones(size(testX, 1), 1) testX];

testY = testdata(:, 6);

prob = sigmoid(testX \* theta);

% Compute accuracy on our training set

p = predict(theta, testX);

fprintf('Train Accuracy on test data set 1: %f\n', mean(double(p == testY)) \* 100);

%% ============== Test data set 1 ==============

testdata = load('datatest2.txt');

testX = testdata(:, [1, 2, 3, 4, 5]);

testX = [ones(size(testX, 1), 1) testX];

testY = testdata(:, 6);

prob = sigmoid(testX \* theta);

%fprintf(['For a test set, we predict an occupancy probability of %f\n\n'], prob);

% Compute accuracy on our training set

p = predict(theta, testX);

fprintf('Train Accuracy on test data set 2: %f\n', mean(double(p == testY)) \* 100);

costfunction.m

function [J, grad] = costFunction(theta, X, y)

%COSTFUNCTION Compute cost and gradient for logistic regression

% J = COSTFUNCTION(theta, X, y) computes the cost of using theta as the

% parameter for logistic regression and the gradient of the cost

% w.r.t. to the parameters.

% Initialize some useful values

m = length(y); % number of training examples

% You need to return the following variables correctly

J = 0;

grad = zeros(size(theta));

% ====================== YOUR CODE HERE ======================

% Instructions: Compute the cost of a particular choice of theta.

% You should set J to the cost.

% Compute the partial derivatives and set grad to the partial

% derivatives of the cost w.r.t. each parameter in theta

%

% Note: grad should have the same dimensions as theta

%

h = 1./(1+e.^(-X \* theta));

J = 1/m \* sum((-y.\*log(h)) - ((1-y).\*log(1-h)));

grad = 1/m \* ((h - y)'\*X);

% =============================================================

end

predict.m

function p = predict(theta, X)

%PREDICT Predict whether the label is 0 or 1 using learned logistic

%regression parameters theta

% p = PREDICT(theta, X) computes the predictions for X using a

% threshold at 0.5 (i.e., if sigmoid(theta'\*x) >= 0.5, predict 1)

m = size(X, 1); % Number of training examples

% You need to return the following variables correctly

p = zeros(m, 1);

% ====================== YOUR CODE HERE ======================

% Instructions: Complete the following code to make predictions using

% your learned logistic regression parameters.

% You should set p to a vector of 0's and 1's

%

%if (sigmoid(X\*theta) >= 0.5)

% p = 1;

%else

% p = 0;

%endif

p = sigmoid(X\*theta)>=0.5;

% =========================================================================

end

sigmoid.m

function g = sigmoid(z)

%SIGMOID Compute sigmoid functoon

% J = SIGMOID(z) computes the sigmoid of z.

% You need to return the following variables correctly

g = zeros(size(z));

% ====================== YOUR CODE HERE ======================

% Instructions: Compute the sigmoid of each value of z. (z can be a matrix,

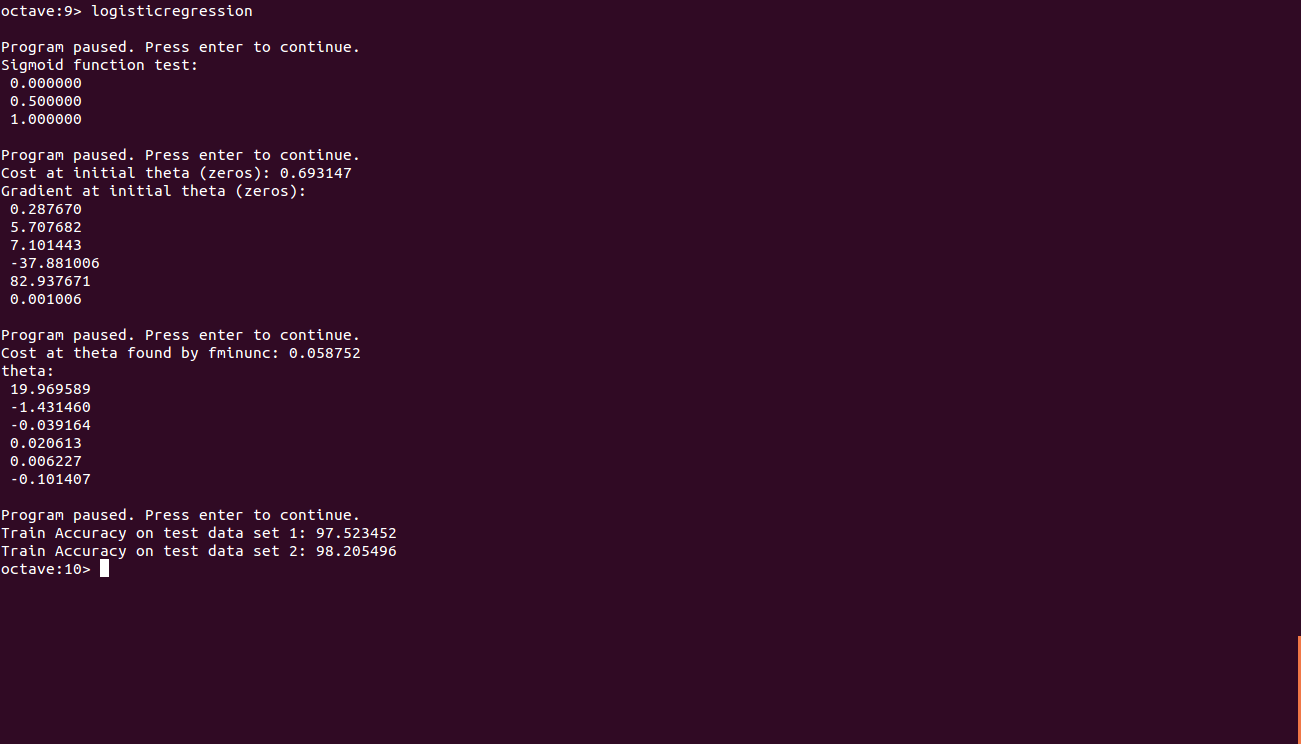
% vector or scalar).

g = 1./(1+e.^(-z));

% =============================================================

end

Output:



**2. Linear Discriminant Analysis:**

lda.m

%% Initialization

clear ; close all;

%% Load Data

data = load('datatraining.txt');

X = data(:, [1, 2, 3, 4, 5]);

y = data(:, 6);

%% ==================== Part 1: Plotting ====================

% We start the exercise by first plotting the data to understand the

% the problem we are working with.

fprintf(['Plotting data with + indicating (y = 1) examples and o ' ...

'indicating (y = 0) examples.\n']);

%plotData(X, y);

%plotMultiData(X, y);

fprintf('\nProgram paused. Press enter to continue.\n');

pause;

%% ============ Linear Discriminant Analysis ============

[mean\_0, mean\_1, sigma, priori\_0, priori\_1 ] = ldac(X, y);

printf('The mean from training set for y=0:\n')

disp(mean\_0);

printf('The mean from training set for y=1:\n')

disp(mean\_1);

printf('The value of sigma from training set:\n');

disp(sigma);

sigmaInv = inv(sigma);

%% ============ LDA Test on data set 1 ============

testdata = load('datatest.txt');

testX = testdata(:, [1, 2, 3, 4, 5]);

testY = testdata(:, 6);

discriminant\_1 = testX \* sigmaInv \* mean\_1' - 0.5 \* mean\_1 \* sigmaInv \* mean\_1' + log(priori\_1);

discriminant\_0 = testX \* sigmaInv \* mean\_0' - 0.5 \* mean\_0 \* sigmaInv \* mean\_0' + log(priori\_0);

ldac = (discriminant\_1 > discriminant\_0);

accuracy = mean(double(ldac == testY)) \* 100;

fprintf('Training Accuracy of LDA classifier for test data set 1 is: %f \n', accuracy);

%% ============ LDA Test on data set 2 ============

testdata = load('datatest2.txt');

testX = testdata(:, [1, 2, 3, 4, 5]);

testY = testdata(:, 6);

discriminant\_1 = testX \* sigmaInv \* mean\_1' - 0.5 \* mean\_1 \* sigmaInv \* mean\_1' + log(priori\_1);

discriminant\_0 = testX \* sigmaInv \* mean\_0' - 0.5 \* mean\_0 \* sigmaInv \* mean\_0' + log(priori\_0);

ldac = (discriminant\_1 > discriminant\_0);

accuracy = mean(double(ldac == testY)) \* 100;

fprintf('Training Accuracy of LDA classifier for test data set 2 is: %f \n', accuracy);

%hold off;

ldac.m

function [mean\_0, mean\_1, sigma, priori\_0, priori\_1 ] = ldac(X, y)

% Linear Discriminant Analysis

first\_class = find(y == 1);

second\_class = find(y == 0);

N = size(X, 1);

% priori probabilities of two classes

priori\_1 = size(X(first\_class, :), 1) / N;

priori\_0 = size(X(second\_class, :), 1) / N;

% centroids of two classes

mean\_1 = mean(X(first\_class, :), 1);

mean\_0 = mean(X(second\_class, :), 1);

sigma = zeros(size(X, 2));

% Covariance Matrix

for i = 1:size(first\_class, 1)

Xi = X(first\_class(i), :);

sigma = sigma + (Xi - mean\_1)' \* (Xi - mean\_1);

end

for i = 1:size(second\_class, 1)

Xi = X(second\_class(i), :);

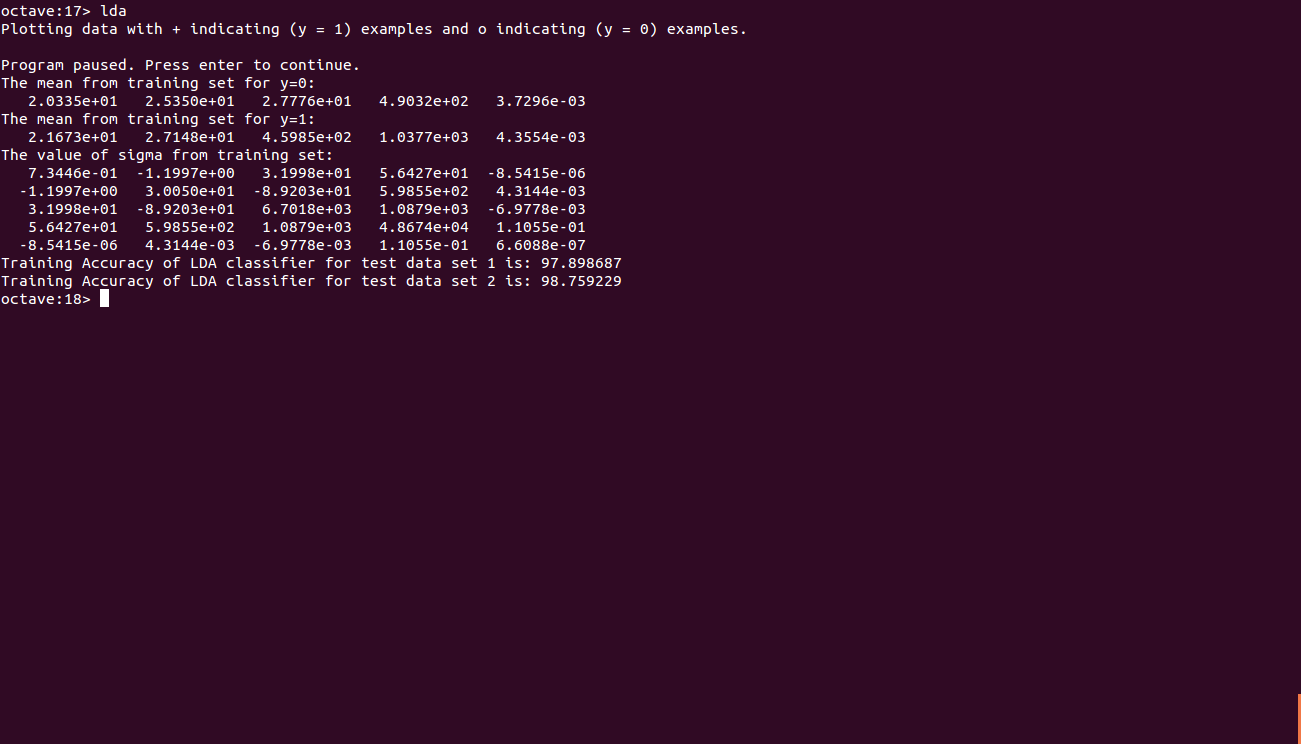
sigma = sigma + (Xi - mean\_0)' \* (Xi - mean\_0);

end

sigma = sigma / N;

end

Output:



**3. Quadratic Discriminant Analysis:**

qda.m

%% Initialization

clear ; close all;

%% Load Data

data = load('datatraining.txt');

X = data(:, [1, 2, 3, 4, 5]);

y = data(:, 6);

[mean\_0, mean\_1, sigma\_0, sigma\_1, priori\_0, priori\_1 ] = qdac(X, y);

printf('The mean from training set for y=0:\n')

disp(mean\_0);

printf('The mean from training set for y=1:\n')

disp(mean\_1);

printf('The sigma from training set for y=0:\n');

disp(sigma\_0);

printf('The sigma from training set for y=1:\n');

disp(sigma\_1);

sigma1\_inv = inv(sigma\_1);

sigma0\_inv = inv(sigma\_0);

testdata = load('datatest.txt');

testX = testdata(:, [1, 2, 3, 4, 5]);

testY = testdata(:, 6);

discriminant\_1 = zeros(size(testX, 1), 1);

discriminant\_0 = zeros(size(testX, 1), 1);

for i=1:size(testX, 1)

x = testX(i,:);

discriminant\_1(i) = -0.5 \* log(det(sigma\_1)) - 0.5 \* (x - mean\_1) \* sigma1\_inv \* (x - mean\_1)' + log(priori\_1);

end

for i=1:size(testX, 1)

x = testX(i,:);

discriminant\_0(i) = -0.5 \* log(det(sigma\_0)) - 0.5 \* (x - mean\_0) \* sigma0\_inv \* (x - mean\_0)' + log(priori\_0);

end

qdac = (discriminant\_1 > discriminant\_0);

accuracy = mean(double(qdac == testY)) \* 100;

fprintf('Training Accuracy of QDA classifier for test data set 1 is: %f \n', accuracy);

testdata = load('datatest2.txt');

testX = testdata(:, [1, 2, 3, 4, 5]);

testY = testdata(:, 6);

discriminant\_1 = zeros(size(testX, 1), 1);

discriminant\_0 = zeros(size(testX, 1), 1);

for i=1:size(testX, 1)

x = testX(i,:);

discriminant\_1(i) = -0.5 \* log(det(sigma\_1)) - 0.5 \* (x - mean\_1) \* sigma1\_inv \* (x - mean\_1)' + log(priori\_1);

end

for i=1:size(testX, 1)

x = testX(i,:);

discriminant\_0(i) = -0.5 \* log(det(sigma\_0)) - 0.5 \* (x - mean\_0) \* sigma0\_inv \* (x - mean\_0)' + log(priori\_0);

end

qdac = (discriminant\_1 > discriminant\_0);

accuracy = mean(double(qdac == testY)) \* 100;

fprintf('Training Accuracy of QDA classifier for test data set 2 is: %f \n', accuracy);

qdac.m

function [mean\_0, mean\_1, sigma\_0, sigma\_1, priori\_0, priori\_1 ] = qdac(X, y)

first\_class = find(y == 1);

second\_class = find(y == 0);

N = size(X, 1);

priori\_1 = size(X(first\_class, :), 1) / N;

priori\_0 = size(X(second\_class, :), 1) / N;

mean\_1 = mean(X(first\_class, :), 1);

mean\_0 = mean(X(second\_class, :), 1);

sigma\_1 = zeros(size(X, 2));

for i = 1:size(first\_class, 1)

Xi = X(first\_class(i), :);

sigma\_1 = sigma\_1 + (Xi - mean\_1)' \* (Xi - mean\_1);

end

sigma\_1 = sigma\_1 / (size(first\_class, 1));

sigma\_0 = zeros(size(X, 2));

for i = 1:size(second\_class, 1)

Xi = X(second\_class(i), :);

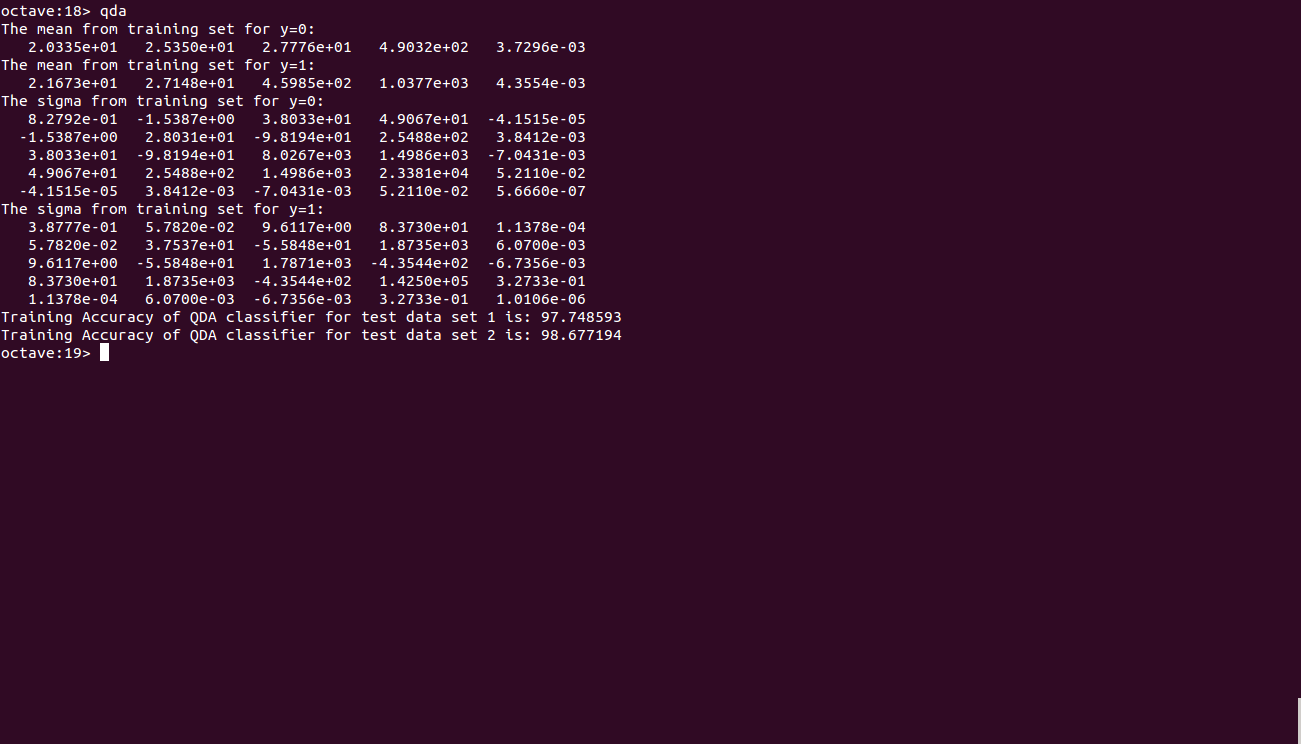
sigma\_0 = sigma\_0 + (Xi - mean\_0)' \* (Xi - mean\_0);

end

sigma\_0 = sigma\_0 / (size(second\_class, 1));

end

Output:



**4. Naive Bayes Classifier**

nbc.m

% NAIVE BAYES CLASSIFIER

clear ; close all; clc

tic

disp('--- start ---')

distr='kernel';

% read data

data = load('datatraining.txt');

X = data(:, [1, 2, 3, 4, 5]);

y = data(:, 6);

% test set

testdata = load('datatest.txt');

testX = testdata(:, [1, 2, 3, 4, 5]);

testY = testdata(:, 6);

yu=unique(y);

nc=length(yu); % number of classes

ni=size(X,2); % independent variables

ns=length(testY); % test set

% compute class probability

for i=1:nc

fy(i)=sum(double(y==yu(i)))/length(y);

end

% kernel distribution

% probability of test set estimated from training set

for i=1:nc

for k=1:ni

xi=X(y==yu(i),k);

ui=testX(:,k);

fuStruct(i,k).f=ksdensity(xi,ui);

end

end

% re-structure

for i=1:ns

for j=1:nc

for k=1:ni

fu(j,k)=fuStruct(j,k).f(i);

end

end

P(i,:)=fy.\*prod(fu,2)';

end

% get predicted output for test set

[pv0,id]=max(P,[],2);

for i=1:length(id)

pv(i,1)=yu(id(i));

end

% compare predicted output with actual output from test data

confMat=myconfusionmat(testY,pv);

disp('confusion matrix:')

disp(confMat)

conf=sum(pv==testY)/length(pv);

disp(['accuracy = ',num2str(conf\*100),'%'])

testdata = load('datatest2.txt');

testX = testdata(:, [1, 2, 3, 4, 5]);

testY = testdata(:, 6);

ns=length(testY); % test set

% probability of test set estimated from training set

for i=1:nc

for k=1:ni

xi=X(y==yu(i),k);

ui=testX(:,k);

fuStruct(i,k).f=ksdensity(xi,ui);

end

end

% re-structure

for i=1:ns

for j=1:nc

for k=1:ni

fu(j,k)=fuStruct(j,k).f(i);

end

end

P(i,:)=fy.\*prod(fu,2)';

end

% get predicted output for test set

[pv0,id]=max(P,[],2);

for i=1:length(id)

pv(i,1)=yu(id(i));

end

% compare predicted output with actual output from test data

confMat=myconfusionmat(testY,pv);

disp('confusion matrix:')

disp(confMat)

conf=sum(pv==testY)/length(pv);

disp(['accuracy = ',num2str(conf\*100),'%'])

toc

myconfusionmat.m

function confMat=myconfusionmat(v,pv)

yu=unique(v);

confMat=zeros(length(yu));

for i=1:length(yu)

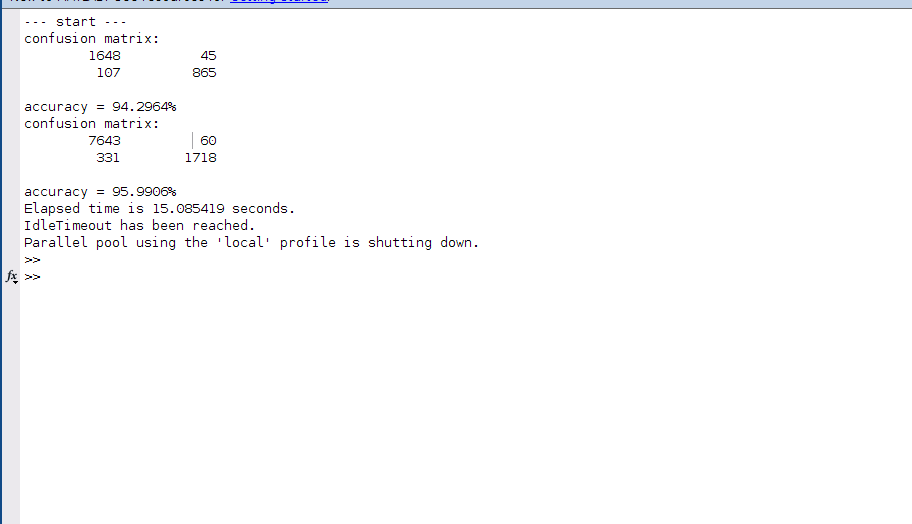
for j=1:length(yu)

confMat(i,j)=sum(v==yu(i) & pv==yu(j));

end

end

Output:

****