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:

```
octave:9> logisticregression

Program paused. Press enter to continue.
Sigmoid function test:
 0.000000
 0.500000
 1.000000

Program paused. Press enter to continue.
Cost at initial theta (zeros): 0.693147
Gradient at initial theta (zeros):
 0.287670
 5.707682
 7.101443
 -37.881006
 82.937671
 0.001006

Program paused. Press enter to continue.
Cost at theta found by fminunc: 0.058752
theta:
 19.969589
 -1.431460
 -0.039164
 0.020613
 0.006227
 -0.101407

Program paused. Press enter to continue.
Train Accuracy on test data set 1: 97.523452
Train Accuracy on test data set 2: 98.205496
octave:10>
```

2. <u>Linear Discriminant Analysis:</u>

<u>lda.m</u>

```
%% 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;
```

<u>ldac.m</u>

```
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
```

<u>Output:</u>

```
octave:17> lda
Plotting data with + indicating (y = 1) examples and o indicating (y = 0) examples.

Program paused. Press enter to continue.
The mean from training set for y=0:
   2.0335e+01   2.5350e+01   2.7776e+01   4.9032e+02   3.7296e-03
The mean from training set for y=1:
   2.1673e+01   2.7148e+01   4.5985e+02   1.0377e+03   4.3554e-03
The value of sigma from training set:
   7.3446e-01  -1.1997e+00   3.1998e+01   5.6427e+01  -8.5415e-06
  -1.1997e+00   3.0050e+01  -8.9203e+01   5.9855e+02   4.3144e-03
   3.1998e+01  -8.9203e+01   6.7018e+03   1.0879e+03  -6.9778e-03
   5.6427e+01   5.9855e+02   1.0879e+03   4.8674e+04   1.1055e-01
  -8.5415e-06   4.3144e-03  -6.9778e-03   1.1055e-01   6.6088e-07
Training Accuracy of LDA classifier for test data set 1 is: 97.898687
Training Accuracy of LDA classifier for test data set 2 is: 98.759229
octave:18>
```

3. <u>Quadratic Discriminant Analysis:</u>

<u>qda.m</u>

```
%% 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:

```
octave:18> qda
The mean from training set for y=0:
   2.0335e+01   2.5350e+01   2.7776e+01   4.9032e+02   3.7296e-03
The mean from training set for y=1:
   2.1673e+01   2.7148e+01   4.5985e+02   1.0377e+03   4.3554e-03
The sigma from training set for y=0:
   8.2792e-01  -1.5387e+00   3.8033e+01   4.9067e+01  -4.1515e-05
  -1.5387e+00   2.8031e+01  -9.8194e+01   2.5488e+02   3.8412e-03
   3.8033e+01  -9.8194e+01   8.0267e+03   1.4986e+03  -7.0431e-03
   4.9067e+01   2.5488e+02   1.4986e+03   2.3381e+04   5.2110e-02
  -4.1515e-05   3.8412e-03  -7.0431e-03   5.2110e-02   5.6660e-07
The sigma from training set for y=1:
   3.8777e-01   5.7820e-02   9.6117e+00   8.3730e+01   1.1378e-04
   5.7820e-02   3.7537e+01  -5.5848e+01   1.8735e+03   6.0700e-03
   9.6117e+00  -5.5848e+01   1.7871e+03  -4.3544e+02  -6.7356e-03
   8.3730e+01   1.8735e+03  -4.3544e+02   1.4250e+05   3.2733e-01
   1.1378e-04   6.0700e-03  -6.7356e-03   3.2733e-01   1.0106e-06
Training Accuracy of QDA classifier for test data set 1 is: 97.748593
Training Accuracy of QDA classifier for test data set 2 is: 98.677194
octave:19>
```

4. <u>Naive Bayes Classifier</u>

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
```

<u>myconfusionmat.m</u>

```
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
```

<u>Output:</u>

```
--- start ---
confusion matrix:
        1648          45
         107         865

accuracy = 94.2964%
confusion matrix:
        7643          60
         331        1718

accuracy = 95.9906%
Elapsed time is 15.085419 seconds.
IdleTimeout has been reached.
Parallel pool using the 'local' profile is shutting down.
>>
>>
```