Machine Learning - Exercise 2: Logistic Regression

Instructions

This file contains code that helps you get started on the logistic regression exercise. You will need to complete the following functions in this exericse:

% plotData.m

% sigmoid.m

% costFunction.m

% predict.m

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

%% Load Data
%  The first two columns contains the exam scores and the third column
%  contains the label.

data = load('ex2data1.txt');
X = data(:, [1, 2]); y = data(:, 3);

%% ==================== 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']);
```

```
Plotting data with + indicating (y = 1) examples and o indicating (y = 0) examples.
```

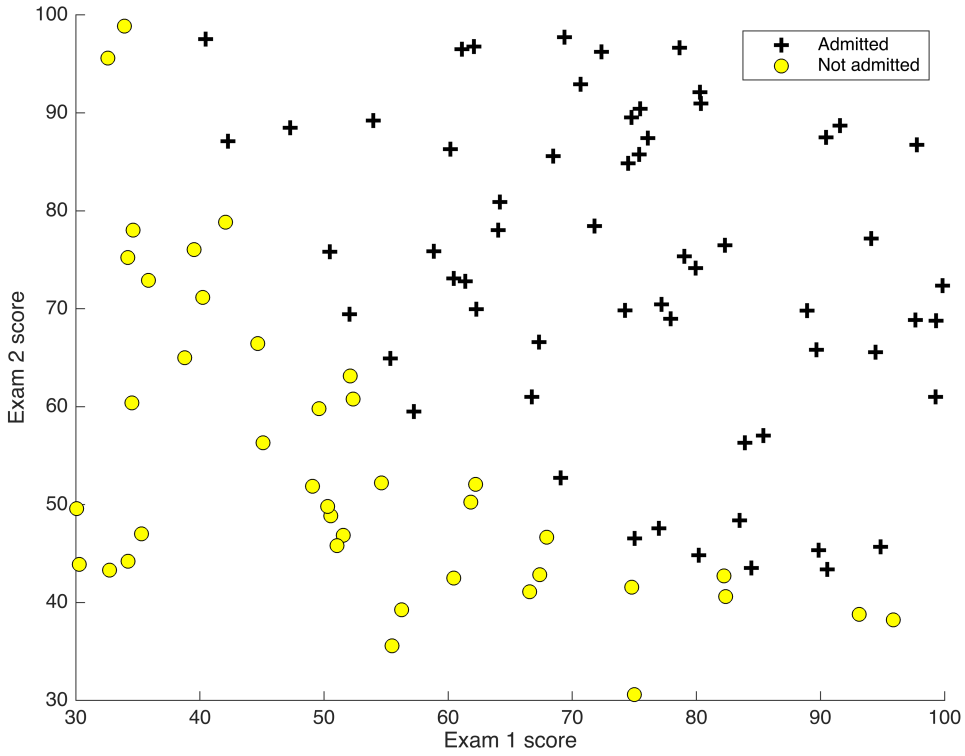
```

%======Finish plotData Below========%
plotData(X, y);

% Put some labels
hold on;
% Labels and Legend
xlabel('Exam 1 score')
ylabel('Exam 2 score')

% Specified in plot order
legend('Admitted', 'Not admitted')
```

```
hold off;
```




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

```
Program paused. Press enter to continue.
```

```
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

%  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);

% =======Finish costFunction to Compute and display initial cost and
% gradient=====
[cost, grad] = costFunction(initial_theta, X, y);
```

```
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
   -0.1000
         0
         0
grad = 3×1
   -0.1000
  -12.0092
         0
grad = 3×1
   -0.1000
  -12.0092
  -11.2628
```

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

```
Cost at initial theta (zeros): 0.693147
```

```
fprintf('Expected cost (approx): 0.693\n');
```

```
Expected cost (approx): 0.693
```

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

```
Gradient at initial theta (zeros):
```

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

```
 -0.100000
 -12.009217
 -11.262842
```

```
fprintf('Expected gradients (approx):\n -0.1000\n -12.0092\n -11.2628\n');
```

```
Expected gradients (approx):
 -0.1000
 -12.0092
 -11.2628
```

```
% Compute and display cost and gradient with non-zero theta
test_theta = [-24; 0.2; 0.2];
[cost, grad] = costFunction(test_theta, X, y);
```

```
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0429
         0
         0
grad = 3×1
    0.0429
    2.5662
         0
grad = 3×1
    0.0429
    2.5662
    2.6468
```

```
fprintf('\nCost at test theta: %f\n', cost);
```

```
Cost at test theta: 0.218330
```

```
fprintf('Expected cost (approx): 0.218\n');
```

```
Expected cost (approx): 0.218
```

```
fprintf('Gradient at test theta: \n');
```

```
Gradient at test theta:
```

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

```
 0.042903
 2.566234
 2.646797
```

```
fprintf('Expected gradients (approx):\n 0.043\n 2.566\n 2.647\n');
```

```
Expected gradients (approx):
 0.043
 2.566
 2.647
```

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

```
Program paused. Press enter to continue.
```

```
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);
```

```
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
   -0.1000
         0
         0
grad = 3×1
   -0.1000
  -12.0092
         0
grad = 3×1
   -0.1000
  -12.0092
  -11.2628
```

```
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.4000
         0
         0
grad = 3×1
    0.4000
   20.8129
         0
grad = 3×1
    0.4000
   20.8129
   21.8482
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.4000
         0
         0
grad = 3×1
    0.4000
   20.8129
         0
grad = 3×1
    0.4000
   20.8129
   21.8482
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.3757
         0
         0
grad = 3×1
    0.3757
   19.4850
         0
grad = 3×1
    0.3757
   19.4850
   20.4520
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0799
         0
         0
grad = 3×1
    0.0799
    0.2760
         0
grad = 3×1
    0.0799
    0.2760
    1.0541
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0712
         0
         0
grad = 3×1
    0.0712
   -0.2986
         0
grad = 3×1
    0.0712
```

```
   -0.2986
    0.4482
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0671
         0
         0
grad = 3×1
    0.0671
   -0.5505
         0
grad = 3×1
    0.0671
   -0.5505
    0.1437
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0613
         0
         0
grad = 3×1
    0.0613
   -0.8535
         0
grad = 3×1
    0.0613
   -0.8535
   -0.3370
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0594
         0
         0
grad = 3×1
    0.0594
   -0.8391
         0
grad = 3×1
    0.0594
   -0.8391
   -0.5966
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0636
         0
         0
grad = 3×1
    0.0636
   -0.4306
         0
grad = 3×1
    0.0636
   -0.4306
   -0.4168
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0684
         0
         0
grad = 3×1
    0.0684
   -0.0797
         0
```

```
grad = 3×1
    0.0684
   -0.0797
   -0.1143
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0698
         0
         0
grad = 3×1
    0.0698
    0.0119
         0
grad = 3×1
    0.0698
    0.0119
   -0.0090
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0703
         0
         0
grad = 3×1
    0.0703
    0.0352
         0
grad = 3×1
    0.0703
    0.0352
    0.0231
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0712
         0
         0
grad = 3×1
    0.0712
    0.0873
         0
grad = 3×1
    0.0712
    0.0873
    0.0975
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0725
         0
         0
grad = 3×1
    0.0725
    0.1583
         0
grad = 3×1
    0.0725
    0.1583
    0.1997
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0747
         0
         0
grad = 3×1
    0.0747
    0.2798
```

```
         0
grad = 3×1
    0.0747
    0.2798
    0.3749
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0781
         0
         0
grad = 3×1
    0.0781
    0.4708
         0
grad = 3×1
    0.0781
    0.4708
    0.6501
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0836
         0
         0
grad = 3×1
    0.0836
    0.7790
         0
grad = 3×1
    0.0836
    0.7790
    1.0945
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0921
         0
         0
grad = 3×1
    0.0921
    1.2650
         0
grad = 3×1
    0.0921
    1.2650
    1.7954
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.1049
         0
         0
grad = 3×1
    0.1049
    2.0159
         0
grad = 3×1
    0.1049
    2.0159
    2.8796
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.1229
         0
         0
grad = 3×1
    0.1229
```

```
    3.1076
         0
grad = 3×1
    0.1229
    3.1076
    4.4621
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.1438
         0
         0
grad = 3×1
    0.1438
    4.4753
         0
grad = 3×1
    0.1438
    4.4753
    6.4797
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.1565
         0
         0
grad = 3×1
    0.1565
    5.5891
         0
grad = 3×1
    0.1565
    5.5891
    8.2727
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.1433
         0
         0
grad = 3×1
    0.1433
    5.5020
         0
grad = 3×1
    0.1433
    5.5020
    8.5397
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.1110
         0
         0
grad = 3×1
    0.1110
    4.5069
         0
grad = 3×1
    0.1110
    4.5069
    7.1859
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0753
         0
         0
grad = 3×1
```

```
    0.0753
    3.3358
         0
grad = 3×1
    0.0753
    3.3358
    5.0410
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0426
         0
         0
grad = 3×1
    0.0426
    2.1096
         0
grad = 3×1
    0.0426
    2.1096
    2.8361
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0223
         0
         0
grad = 3×1
    0.0223
    1.2304
         0
grad = 3×1
    0.0223
    1.2304
    1.4509
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0106
         0
         0
grad = 3×1
    0.0106
    0.6568
         0
grad = 3×1
    0.0106
    0.6568
    0.6686
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0051
         0
         0
grad = 3×1
    0.0051
    0.3497
         0
grad = 3×1
    0.0051
    0.3497
    0.3190
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
    0.0025
         0
         0
```

```
grad = 3×1
    0.0025
    0.1837
         0
grad = 3×1
    0.0025
    0.1837
    0.1676
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
1.0e-03 *
    0.4029
         0
         0
grad = 3×1
    0.0004
    0.0315
         0
grad = 3×1
    0.0004
    0.0315
    0.0523
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
1.0e-03 *
   -0.4449
         0
         0
grad = 3×1
   -0.0004
   -0.0418
         0
grad = 3×1
   -0.0004
   -0.0418
    0.0076
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
1.0e-03 *
   -0.3904
         0
         0
grad = 3×1
   -0.0004
   -0.0464
         0
grad = 3×1
   -0.0004
   -0.0464
    0.0056
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
1.0e-04 *
   -0.4409
         0
         0
grad = 3×1
   -0.0000
   -0.0179
         0
grad = 3×1
   -0.0000
   -0.0179
    0.0123
```

```
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
1.0e-04 *
    0.8716
         0
         0
grad = 3×1
1.0e-03 *
    0.0872
    0.9216
         0
grad = 3×1
    0.0001
    0.0009
    0.0086
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
1.0e-04 *
    0.3714
         0
         0
grad = 3×1
    0.0000
    0.0019
         0
grad = 3×1
    0.0000
    0.0019
    0.0024
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
1.0e-05 *
    0.5702
         0
         0
grad = 3×1
1.0e-03 *
    0.0057
    0.3852
         0
grad = 3×1
1.0e-03 *
    0.0057
    0.3852
    0.3048
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
1.0e-06 *
    0.4374
         0
         0
grad = 3×1
1.0e-04 *
    0.0044
    0.3340
         0
grad = 3×1
1.0e-04 *
    0.0044
    0.3340
    0.2299
Warning: Colon operands must be real scalars. This warning will become an error in a future release.
grad = 3×1
1.0e-07 *
```

```
    0.1535
         0
         0
grad = 3×1
1.0e-05 *
    0.0015
    0.1203
         0
grad = 3×1
1.0e-05 *
    0.0015
    0.1203
    0.1050
Local minimum found.

Optimization completed because the size of the gradient is less than
the value of the optimality tolerance.

<stopping criteria details>
```

```
% fminunc finds costfunction's minimizer

% BONUS - use gradient descent you wrote from last HW to verify the theta
% results

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

```
Cost at theta found by fminunc: 0.203498
```

```
fprintf('Expected cost (approx): 0.203\n');
```

```
Expected cost (approx): 0.203
```

```
fprintf('theta: \n');
```

```
theta:
```

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

```
 -25.161343
 0.206232
 0.201472
```

```
fprintf('Expected theta (approx):\n');
```

```
Expected theta (approx):
```

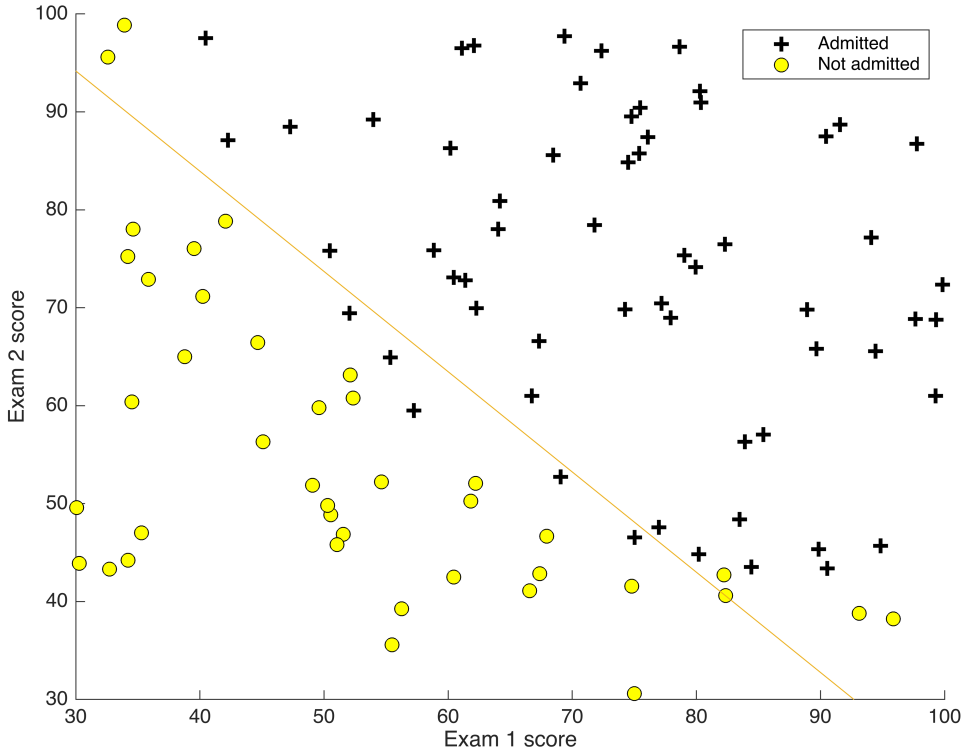
```
fprintf(' -25.161\n 0.206\n 0.201\n');
```

```
 -25.161
 0.206
 0.201
```

```
% Plot Boundary
plotDecisionBoundary(theta, X, y);
```

```
% Put some labels
hold on;
% Labels and Legend
xlabel('Exam 1 score')
ylabel('Exam 2 score')

% Specified in plot order
legend('Admitted', 'Not admitted')
hold off;
```




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

```
Program paused. Press enter to continue.
```

```
pause;

%% ============== Part 4: Predict and Accuracies ==============
%  After learning the parameters, you'll like to use it to predict the
outcomes
%  on unseen data. In this part, you will use the logistic regression model
%  to predict the probability that a student with score 45 on exam 1 and
%  score 85 on exam 2 will be admitted.
%
%  Furthermore, you will compute the training and test set accuracies of
%  our model.
%
```

```
%  Your task is to complete the code in predict.m

%  Predict probability for a student with score 45 on exam 1
%  and score 85 on exam 2

prob = sigmoid([1 45 85] * theta);
fprintf(['For a student with scores 45 and 85, we predict an admission ' ...
         'probability of %f\n'], prob);
```

```
For a student with scores 45 and 85, we predict an admission probability of 0.776291
```

```
fprintf('Expected value: 0.775 +/- 0.002\n\n');
```

```
Expected value: 0.775 +/- 0.002
```

```


% ======Finish predict to Compute accuracy on our training set=======%
p = predict(theta, X);

fprintf('Train Accuracy: %f\n', mean(double(p == y)) * 100);
```

```
Train Accuracy: 89.000000
```

```
fprintf('Expected accuracy (approx): 89.0\n');
```

```
Expected accuracy (approx): 89.0
```

```
fprintf('\n');




```

```
function plotData(X, y)
%PLOTDATA Plots the data points X and y into a new figure
%   PLOTDATA(x,y) plots the data points with + for the positive examples
%   and o for the negative examples. X is assumed to be a Mx2 matrix.

% Create New Figure
figure; hold on;

% ====================== YOUR CODE HERE ======================
% Instructions: Plot the positive and negative examples on a
%               2D plot, using the option 'k+' for the positive
%               examples and 'ko' for the negative examples.
%
% Find Indices of Positive and Negative Examples
pos = find(y==1); %pos = find(y)
neg = find(y==0); %neg = find(~y)

% pos = [];
% neg = [];
%
```

```
% for i = 1:length(y)
%     if y(i)==1
%         pos = [pos i];
%     elseif y(i) == 0
%         neg = [neg i];
%     end
% end

% Plot Examples
plot(X(pos, 1), X(pos, 2), 'k+','LineWidth', 2, ...
'MarkerSize', 7);
plot(X(neg, 1), X(neg, 2), 'ko', 'MarkerFaceColor', 'y', ...
'MarkerSize', 7);









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



hold off;

end
```

sigmoid/logistic function $g(z) = \frac{1}{1+e^{-z}}.$

```
function g = sigmoid(z)
%SIGMOID Compute sigmoid function
%   g = 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+exp(-z)); %g = 1./(1+e.^(-z))



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

end
```

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right],$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

h_theta(x) = g(X*theta), $g(z) = \frac{1}{1+e^{-z}}.$

```
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 = sigmoid(X*theta); %100 by 1 vector

J = (1/m)*(-y'*log(h)-(1-y)'*log(1-h));

for j = 1:size(theta)
    grad(j) = (1/m)*(h-y)'*X(:,j)
end

% bonus: can you compute grad in one line?


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

end
```

h_theta(x)>= 0.5, predict 1, otherwise, predict 0

```
function p = predict(theta, X)
%PREDICT Predict whether the label is 0 or 1 using learned logistic
%regression parameters theta

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
%

h = sigmoid(X*theta);

p(h>=0.5) = 1;

% p = round(h)

% if h_theta >= 0.5
%     p = 1;
% else
%     p = 0;

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


end
```

Appendix [DO NOT CHANGE THESE]

```
function plotDecisionBoundary(theta, X, y)
%PLOTDECISIONBOUNDARY Plots the data points X and y into a new figure with
%the decision boundary defined by theta
%   PLOTDECISIONBOUNDARY(theta, X,y) plots the data points with + for the
%   positive examples and o for the negative examples. X is assumed to be
%   a either
%   1) Mx3 matrix, where the first column is an all-ones column for the
%      intercept.
%   2) MxN, N>3 matrix, where the first column is all-ones

% Plot Data
plotData(X(:,2:3), y);
hold on
```

```
if size(X, 2) <= 3
    % Only need 2 points to define a line, so choose two endpoints
    plot_x = [min(X(:,2))-2,  max(X(:,2))+2];

    % Calculate the decision boundary line
    plot_y = (-1./theta(3)).*(theta(2).*plot_x + theta(1));

    % Plot, and adjust axes for better viewing
    plot(plot_x, plot_y)

    % Legend, specific for the exercise
    legend('Admitted', 'Not admitted', 'Decision Boundary')
    axis([30, 100, 30, 100])
else
    % Here is the grid range
    u = linspace(-1, 1.5, 50);
    v = linspace(-1, 1.5, 50);

    z = zeros(length(u), length(v));
    % Evaluate z = theta*x over the grid
    for i = 1:length(u)
        for j = 1:length(v)
            z(i,j) = mapFeature(u(i), v(j))*theta;
        end
    end
    z = z'; % important to transpose z before calling contour

    % Plot z = 0
    % Notice you need to specify the range [0, 0]
    contour(u, v, z, [0, 0], 'LineWidth', 2)
end
hold off

end


function out = mapFeature(X1, X2)
% MAPFEATURE Feature mapping function to polynomial features
%
%   MAPFEATURE(X1, X2) maps the two input features
%   to quadratic features used in the regularization exercise.
%
%   Returns a new feature array with more features, comprising of
%   X1, X2, X1.^2, X2.^2, X1*X2, X1*X2.^2, etc..
%
%   Inputs X1, X2 must be the same size
%

degree = 6;
```

```
out = ones(size(X1(:,1)));
for i = 1:degree
    for j = 0:i
        out(:, end+1) = (X1.^(i-j)).*(X2.^j);
    end
end
end
```