Programming Exercise 2: Logistic Regression
September 2020

Machine Learning Course (Stanford University)

In this exercise, logistic regression was applied to two different datasets. In the first dataset, we will predict whether a student gets admitted into a university. In the second dataset, we explore regularization and also predict whether microchips from a fabrication plant pass the quality assurance (QA).

1 Logistic regression

1.1 Challenge

Build a logistic regression model to **predict whether a student gets admitted into a university.**

Suppose that you are the administrator of a university department and you want to determine each applicant's chance of admission based on their results on two exams. You have historical data from previous applicants that you can use as a training set for logistic regression. For each training example, you have the applicant's scores on two exams and the admissions decision.

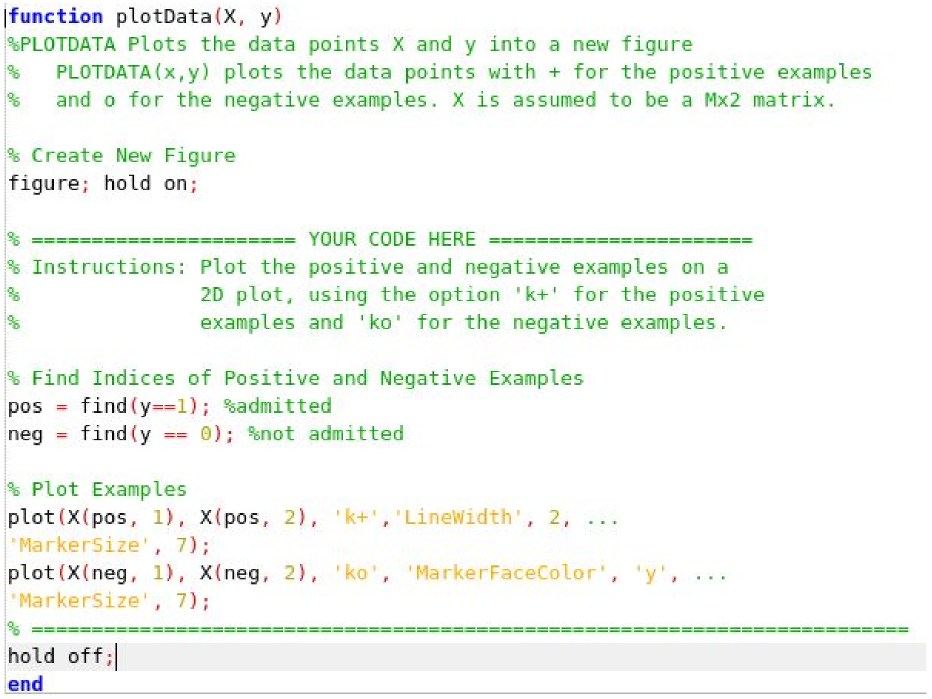
Your task is to **build a classification model that estimates an applicant's probability of admission based on the scores from those two exams.**

1.2 Visualizing the data

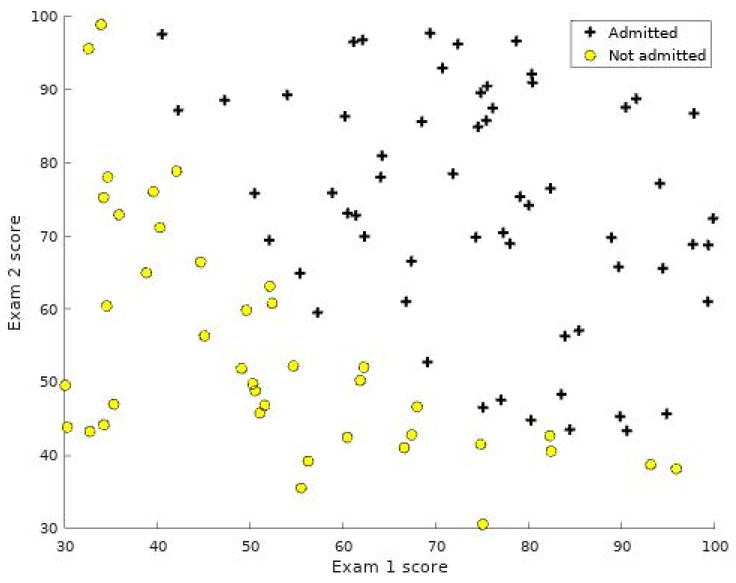
Advice:

Before starting to implement any learning algorithm, it is always good to visualize the data if possible.

plotData.m



```
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); %admitted
neg = find(y == 0); %not admitted

% 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
```



As we can see in the visualization, a linear boundary may suffice. Let’s see in the results later if this is a correct guess.

1.3 Implementation

We will approach the implementation bottom-up: hypothesis inside the sigmoid function, cost and gradient descent then adding learning parameters with `fminunc`.

1.3.1 Hypothesis and Sigmoid Function

Hypothesis:
**We actually implement this in 1.3.2*

$$h_\theta(x) = g(\theta^T x),$$

Sigmoid Function:

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

Notes:

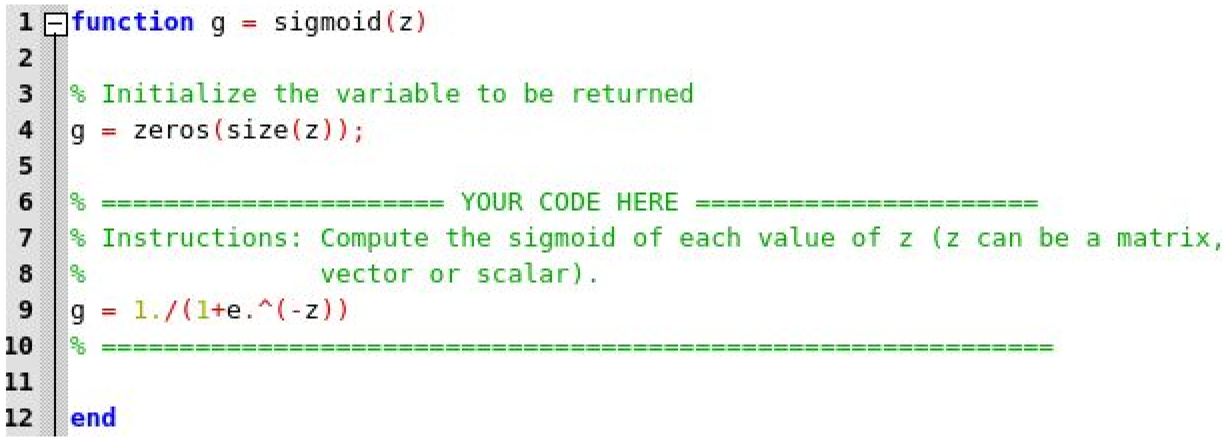
- For large positive values of x, the sigmoid should be close to 1,
- while for large negative values, the sigmoid should be close to 0.
- Evaluating sigmoid(0) should give you exactly 0.5.

Implementation Note:

- The code should also work with vectors and matrices.
- For a matrix, your function should perform the sigmoid function on every element.

So this means that we should work on the vectorized form and do **element-wise** operations.

sigmoid.m



```
function g = sigmoid(z)

% Initialize the variable to be returned
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
```

Here, the exponentiation and the division were done in an element-wise manner.

1.3.2 Cost Function and Gradient of the Cost

Cost Function for Logistic Regression

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

Notes:

- J or the cost is a single number
    - So if we operate some vectorization, the summation is still needed
- Beware of the dimensions

<u>Vectorization (as implemented in Octave)</u>

hypothesis

$$h_\theta(x) = g(x\theta)$$

cost function

$$J(\theta) = \frac{1}{m}\sum[-y'(log(h)) - (1-y)'(log(1-h)]$$

Gradient of the Cost

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

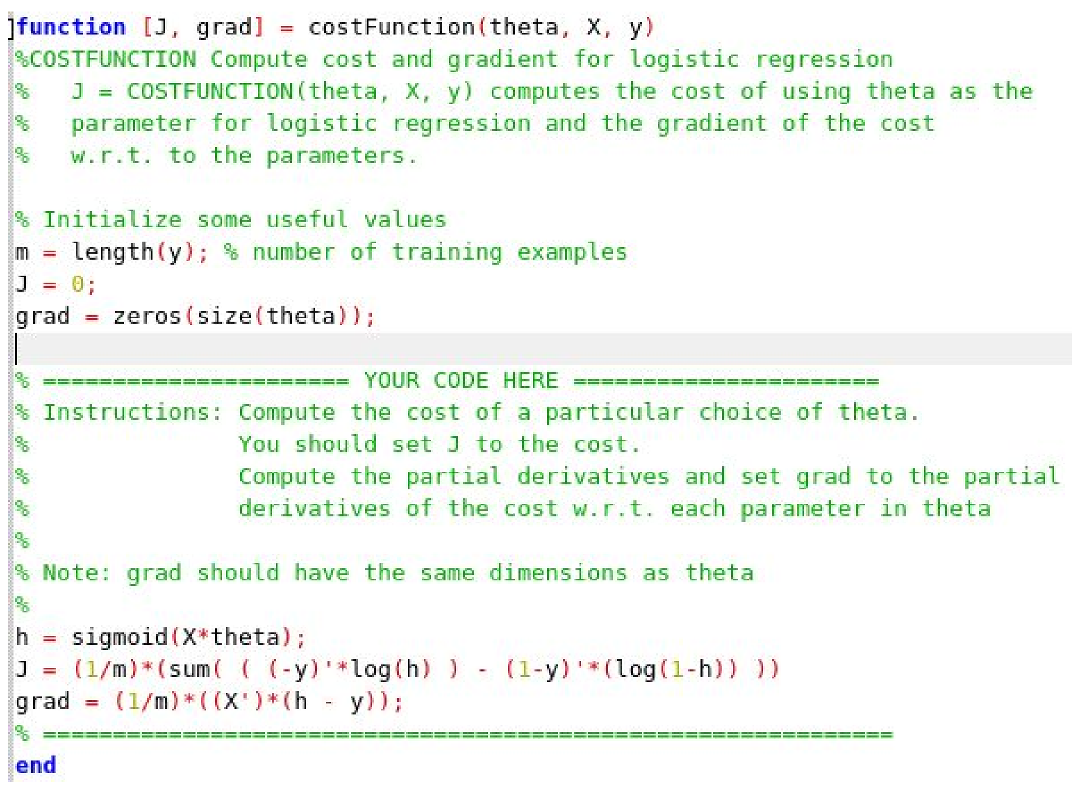
Note(s):

- In this case, we do not need the summation because it’s job here is to go over each observation in each matrix.
    - We can instead do vectorization

<u>Vectorization (as implemented in Octave)</u>

$$grad = \frac{1}{m}(X')(h-y)$$

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
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);
J = (1/m)*(sum( ( (-y)'*log(h) ) - (1-y)'*(log(1-h)) ))
grad = (1/m)*((X')*(h - y));
% =============================================================
end
```

Remark:

- *Note that while this gradient looks identical to the linear regression gradient, the formula is actually different because linear and logistic regression have different definitions of h θ (x).*

1.3.3 Learning parameters using fminunc

Instead of taking gradient descent steps, you will use an Octave/MATLAB built-in function called **fminunc**.

`fminunc`

- an optimization solver that finds the minimum of an unconstrained function
- For logistic regression, you want to optimize the cost function J(θ) with parameters θ.

- Usage:
  - use `fminunc` to find the best parameters θ for the logistic regression cost function, given a fixed dataset (of X and y values)
  - You will pass to fminunc the following inputs:
    - The initial values of the parameters we are trying to optimize.
    - A function that, when given the training set and a particular θ, computes the logistic regression cost and gradient with respect to θ for the dataset (X, y)

**This is given.*

```
%  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);
```

`'GradObj', 'on'`

- Tells `fminunc` that function returns both the cost and the gradient
- Allows `fminunc` to use gradient when minimizing the function

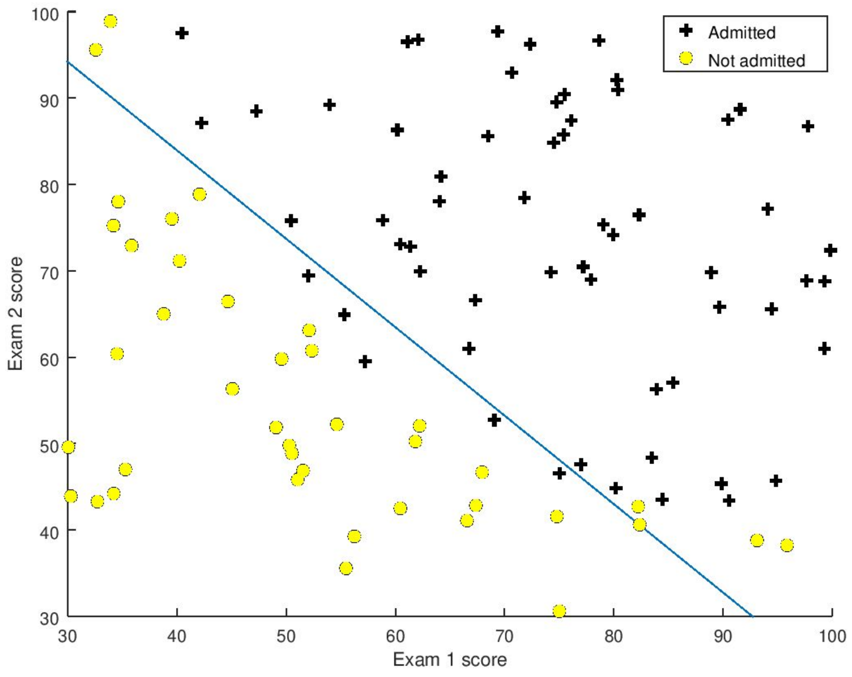
`'MaxIter', 400`

- `fminunc` will run for at most 400 steps before it terminates

Note:

- using `fminunc`, we did not have to write any loops ourselves or set a learning rate like we did for gradient descent
    - This is all done by `fminunc`: we only needed to provide a function calculating the cost and the gradient.

Resulting Decision Boundary



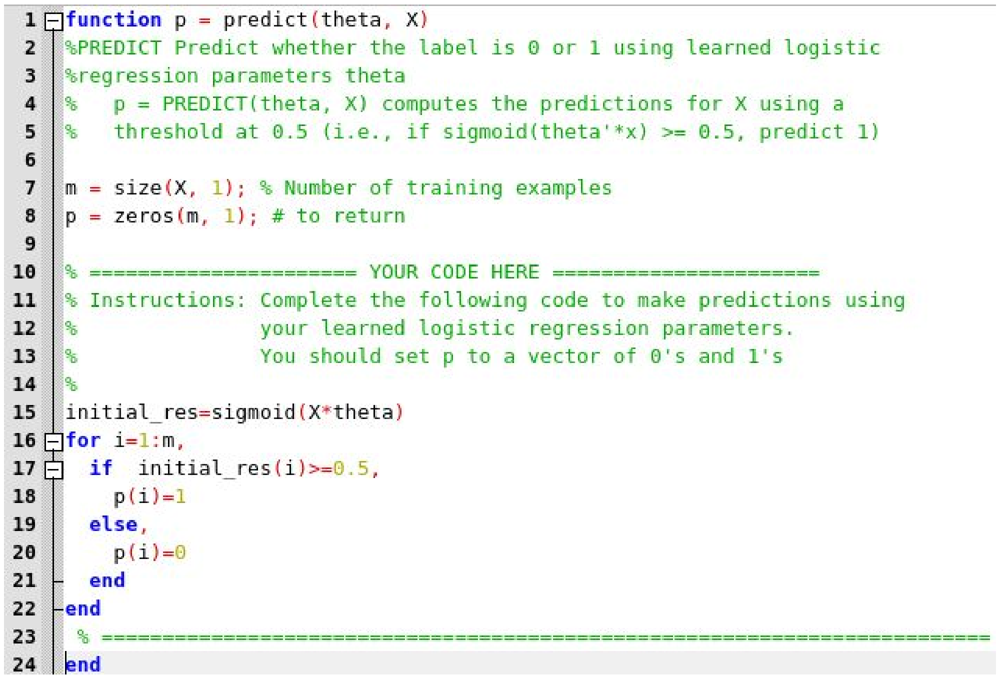
1.3.4 Evaluating logistic regression

There are two ways:

- Inference
    - After learning the parameters, you can use the model to predict whether a particular student will be admitted.
- Training set performance
    - Another way to evaluate the quality of the parameters we have found is to see how well the learned model predicts on our training set.

We do prediction by:

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
p = zeros(m, 1); # to return

% ====================== 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
%
initial_res=sigmoid(X*theta)
for i=1:m,
  if  initial_res(i)>=0.5,
    p(i)=1
  else,
    p(i)=0
  end
end
 % =========================================================================
end
```

Since our training examples have the size of **m**, we will also return the **p** or prediction with the size of **m**.

The input to make the prediction is theta and X.
As before,

- Theta - are the learned/trained parameters
- X - inputs to check/predict

As we know, the hypothesis or the model is

$$h_\theta(x) = g(x\theta)$$

In this exercise, since X has the dimension of 100x3 and theta has 3x1, the resulting hypothesis will have the dimension of 100x1--100 predictions for 100 row inputs from x.

We call that the initial result since what we really want is the discretized value.
So given that 100x1 predictions, if the result is 0.5 and above, we consider it as 1--which means the student gets admitted. The other case, if the result is below 0.5, then we consider it as 0 -- which means the student is rejected.

The resulting train accuracy is 89%.

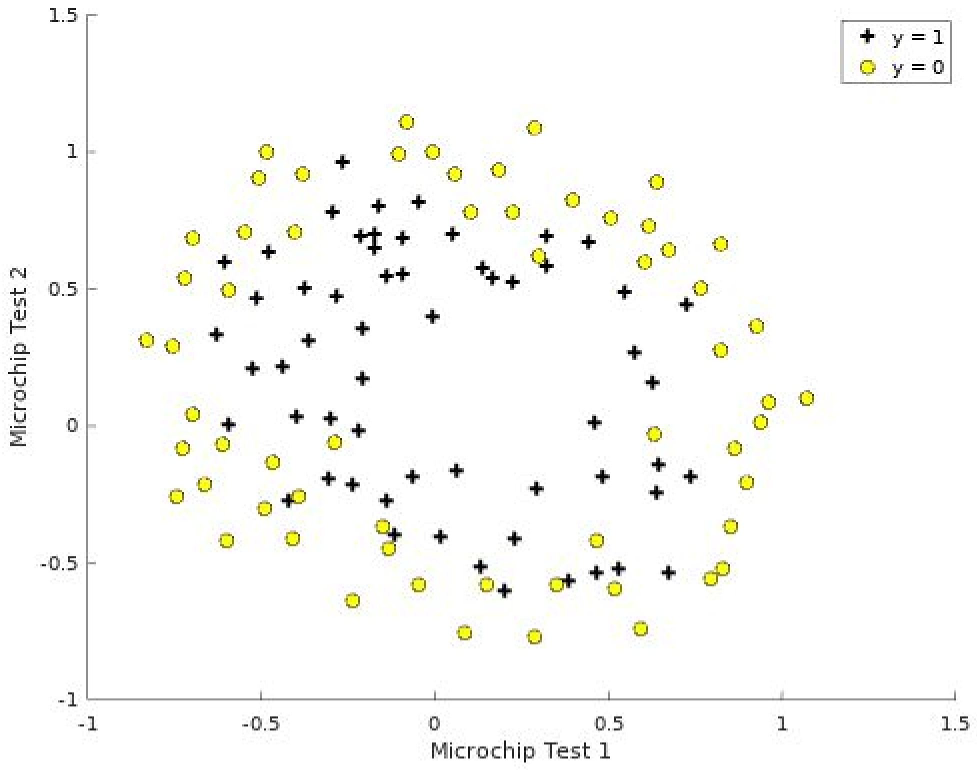
2 Regularized logistic regression

2.1 Challenge

Implement regularized logistic regression to predict whether microchips from a fabrication plant passes quality assurance (QA). During QA, each microchip goes through various tests to ensure it is functioning correctly.

Suppose you are the product manager of the factory and you have the test results for some microchips on two different tests. From these two tests, you would like to determine whether the microchips should be accepted or rejected. To help you make the decision, you have a dataset of test results on past microchips, from which you can build a logistic regression model.

2.2 Visualizing the data



Similar to the previous parts of this exercise, plotData is used to generate a figure like Figure 3, where the axes are the two test scores, and the positive (y = 1, accepted) and negative (y = 0, rejected) examples are shown with different markers.

Advice:

The figure shows that our dataset cannot be separated into positive and negative examples by a straight-line through the plot.

Therefore, a straight-forward application of logistic regression will not perform well on this dataset since logistic regression will only be able to find a linear decision boundary.

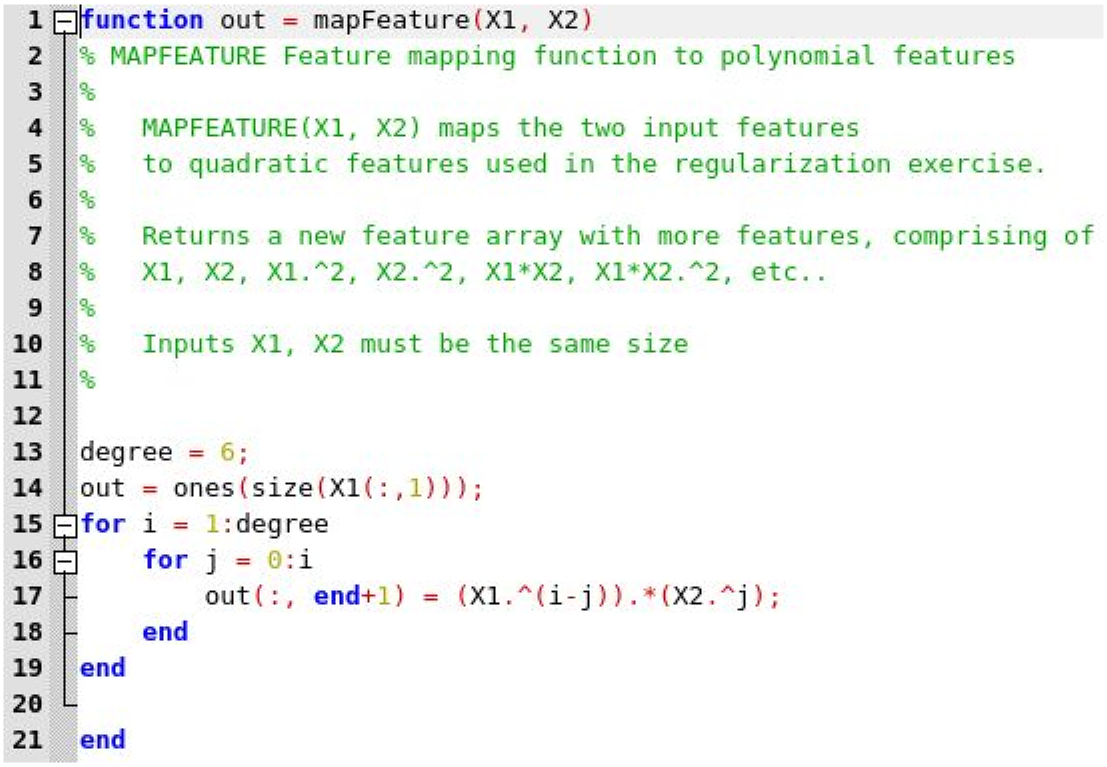
But what shall we do if our feature are only in first degree?

2.3 Feature mapping

Advice:

One way to fit the data better is to create more features from each data point.

In this exercise,
mapFeature.m was given



```
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
```

It maps the features into all polynomial terms of x_1 and x_2 up to the sixth power. Thus, the two features (the scores on two QA tests) have been transformed into a 28-dimensional vector.

A logistic regression classifier trained on this higher-dimension feature vector will have a more complex decision boundary and will appear nonlinear when drawn in our 2-dimensional plot.

Caution:
While the feature mapping allows us to build a more expressive classifier, it also more susceptible to overfitting.

That's why in the next parts, we will implement regularized logistic regression to fit the data and also see how regularization can help combat the overfitting problem.

2.4 Cost function and gradient

Cost Function for Logistic Regression with Regularization

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2.$$

Caution: do not regularize the parameter θ_0 as discussed in the lectures. In this implementation, parameter θ_0 is theta(1).

Vectorization (as implemented in Octave)

$$J(\theta) \;=\; \frac{1}{m}\sum\,[(-y)'\,log(h)\;-(1-y)'\,log(1-h)]\;+\frac{\lambda}{2m}\sum_{k=2}^{size(\theta)}\theta_k^2$$

Gradient of the Cost with Regularization

$$\frac{\partial J(\theta)}{\partial\theta_j}=\left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}\right)+\frac{\lambda}{m}\theta_j\qquad\text{for } j\geq 1$$

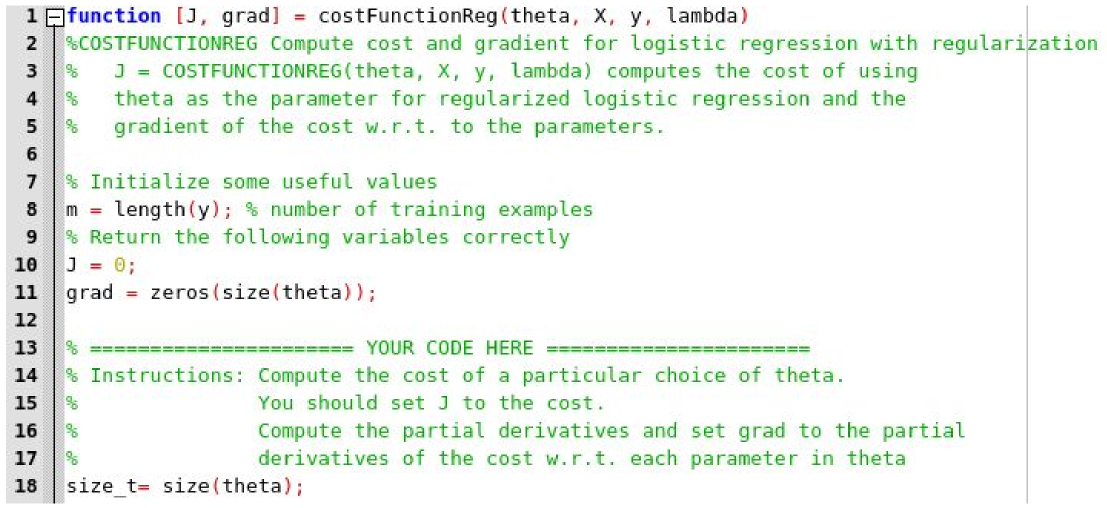
Vectorization (as implemented in Octave)

$$grad\;=\;\frac{1}{m}(X')(h-y)\;+\;gradtheta$$

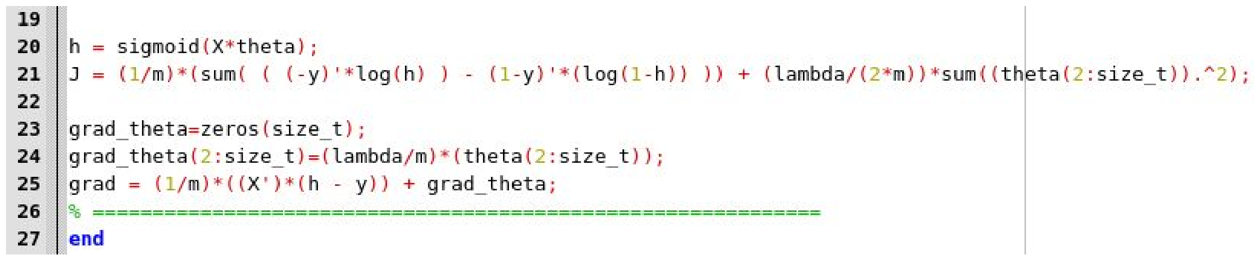
where

$$gradtheta_{2\ to\ size(\theta)}\;=\;\frac{\lambda}{m}\,\theta_{2\ to\ size(\theta)}$$

costFunctionReg.m



```
function [J, grad] = costFunctionReg(theta, X, y, lambda)
%COSTFUNCTIONREG Compute cost and gradient for logistic regression with regularization
%   J = COSTFUNCTIONREG(theta, X, y, lambda) computes the cost of using
%   theta as the parameter for regularized 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
% 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
size_t= size(theta);
```

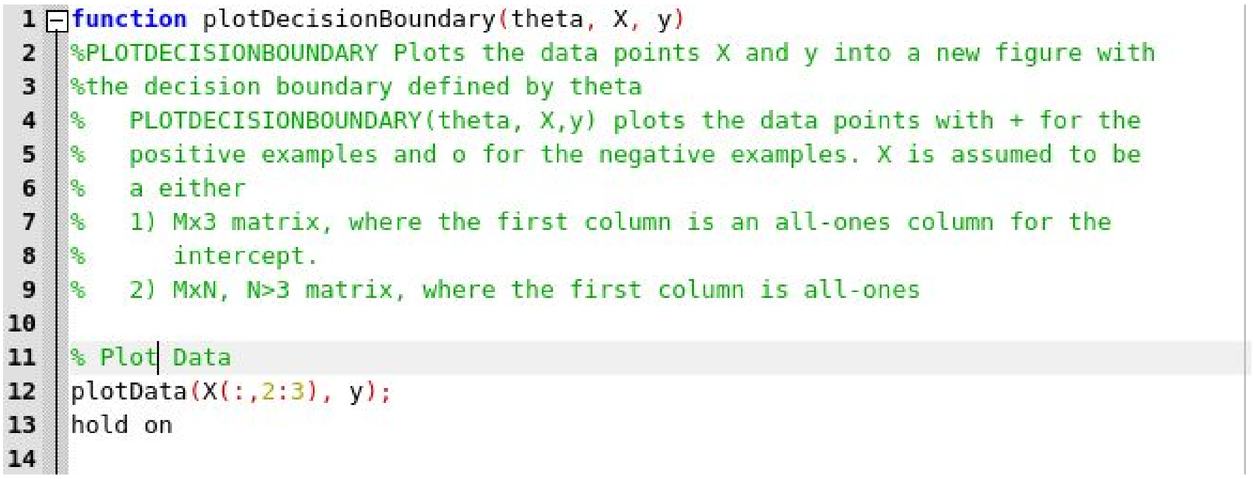


```

h = sigmoid(X*theta);
J = (1/m)*(sum( ( (-y)'*log(h) ) - (1-y)'*(log(1-h)) )) + (lambda/(2*m))*sum((theta(2:size_t)).^2);

grad_theta=zeros(size_t);
grad_theta(2:size_t)=(lambda/m)*(theta(2:size_t));
grad = (1/m)*((X')*(h - y)) + grad_theta;
% =============================================================
end
```

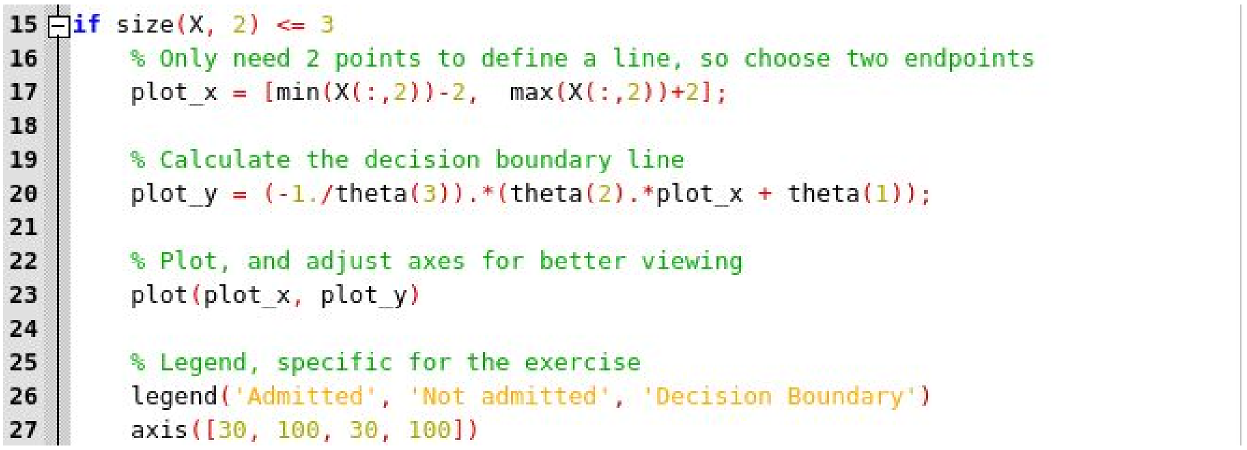
The first cost is 0.693.

2.5 Plotting the decision boundary

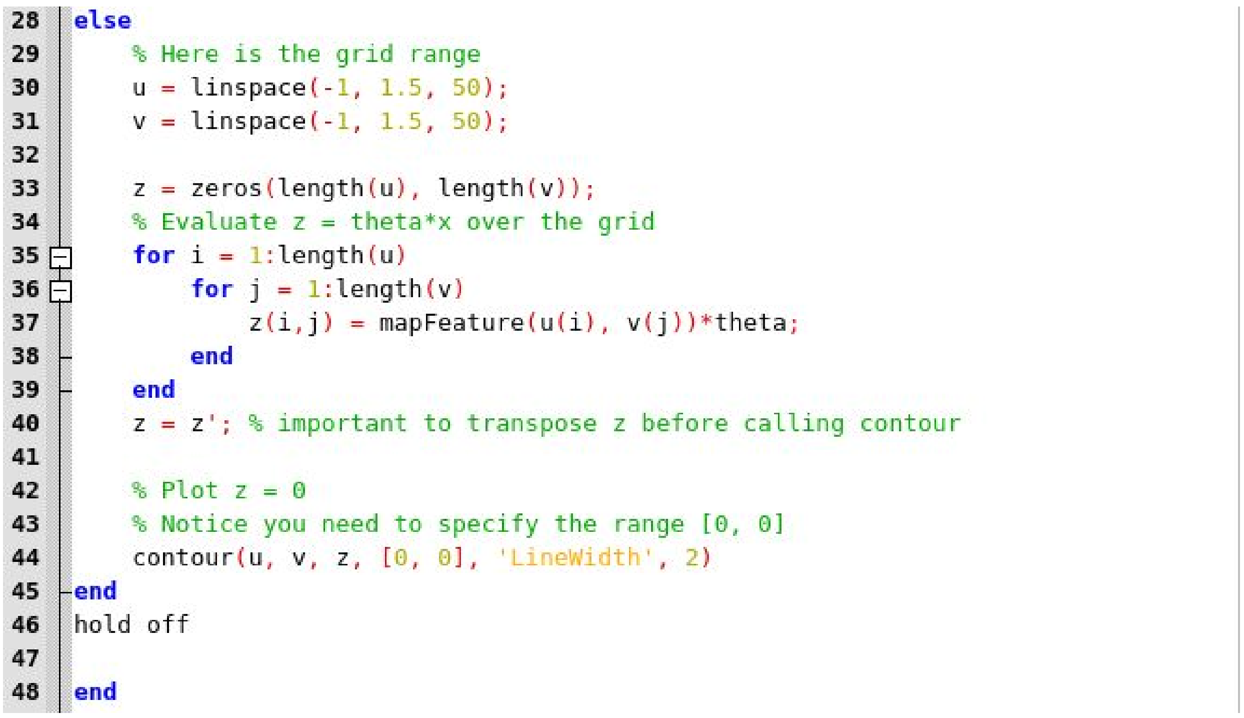
plotDecisionBoundary.m
*This was given.



```
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

```

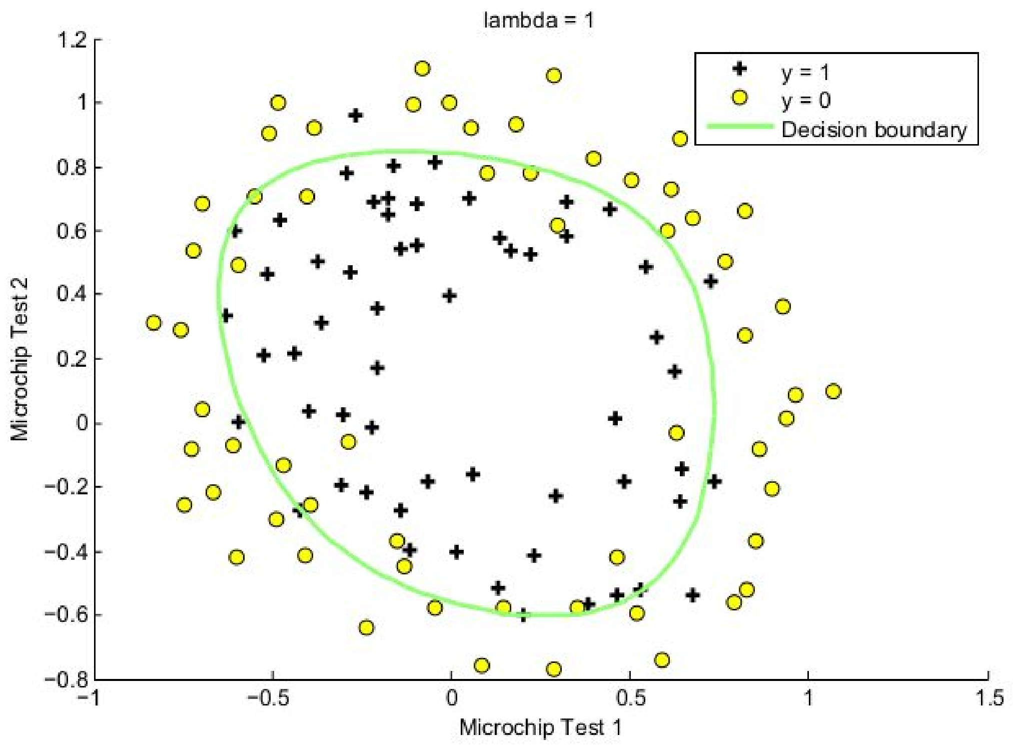


```
if size(X, 2) <= 3
    % Only need 2 points to define a line, so choose two endpoints
    plot_x = [min(X(:,2))-2,  max(X(:,2))+2];

    % 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])
```

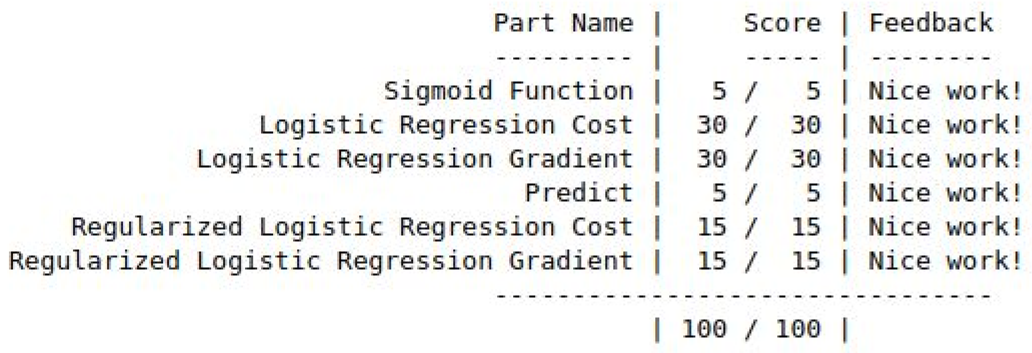


```
else
    % Here is the grid range
    u = linspace(-1, 1.5, 50);
    v = linspace(-1, 1.5, 50);

    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
```

With λ=1, resulting training accuracy is 83.050847.



3 Scores



```
                              Part Name |     Score | Feedback
                              --------- |     ----- | --------
                       Sigmoid Function |   5 /   5 | Nice work!
               Logistic Regression Cost |  30 /  30 | Nice work!
           Logistic Regression Gradient |  30 /  30 | Nice work!
                                Predict |   5 /   5 | Nice work!
   Regularized Logistic Regression Cost |  15 /  15 | Nice work!
Regularized Logistic Regression Gradient |  15 /  15 | Nice work!
                              --------------------------------
                                        | 100 / 100 |
```