Name: Xinrun Zhang

ML Homework 4 Report

Instructor: Prof. Kaliappan Gopalan

Time: 20/03/2019

# Files

1. main\_part\_1.m – Includes the main program of the homework 4 part 1.
2. main\_part\_2.m – Includes the main program of the homework 4 part 2.
3. computeCost.m – This function is used to compute the cost between hypothesis and input y for part 1.
4. computeCost\_new.m – This function is used to compute the cost between hypothesis and input y for part 1.
5. computeCost\_part2.m – This function is used to compute the cost between hypothesis and input y for part 2.
6. normalization.m – This function is used in part 2 to normalize the input x.
7. gradientDescent.m – This function is used in part 1 to execute the gradient descent to find proper parameters.
8. gradientDescent\_part2.m – This function is used in part 2 to execute the gradient descent to find proper parameters.
9. mapFeatures.m – This function is used in part 2 to map the x into polynomial form.

# Logistic Regression

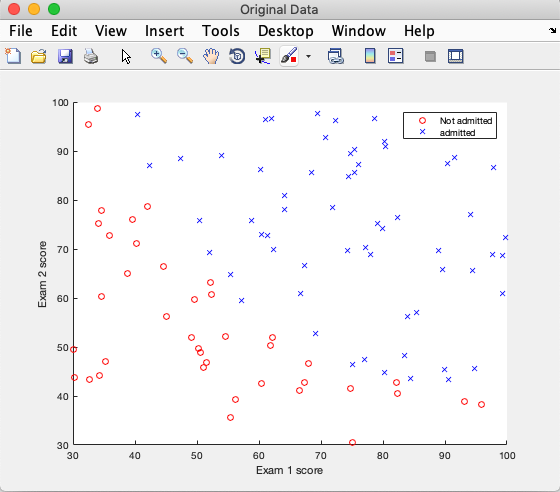
In this case, I found that without normalization, the proper theta could not be found by gradient descent. Therefore, I made two versions of logistic regression, which the first one is logistic regression with gradient descent and the second one is logistic regression by using the build-in function *optimset* and *fminunc*.

The second version referenced some materials online:

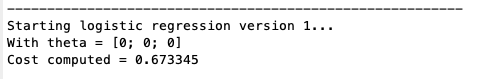
https://www.youtube.com/watch?v=bQqtZyav6K8

https://github.com/j-pandeirada/mlcourse/blob/master/Class%203/ex2.m

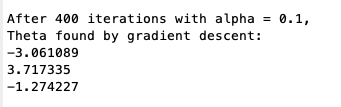
1. Scatter plot of the two features with y = 1 and y = 0 distinguished (similar to Fig. 1).

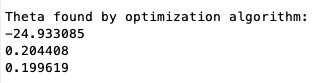


1. Cost at initial theta of all zeros.



1. Parameter values for theta.



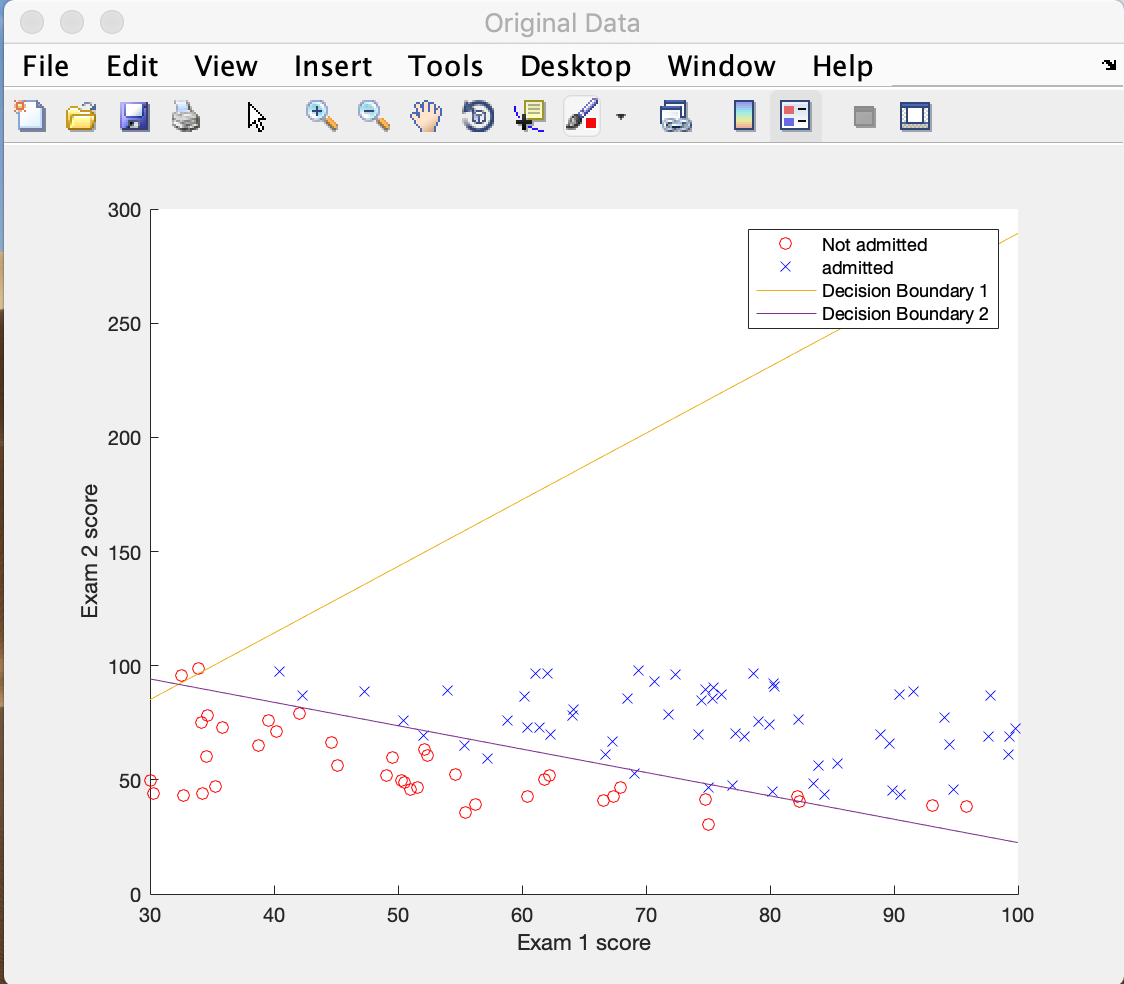


1. Cost after 400 iterations of logistic regression algorithm with .





1. Decision boundary for logistic regression superimposed on the scatter plot.



1. Training accuracy – No. of correct decisions/Total No. of students, in percentage.



1. Probability of getting admitted for a student with Exam 1 score of 45 and Exam 2 score of 85.



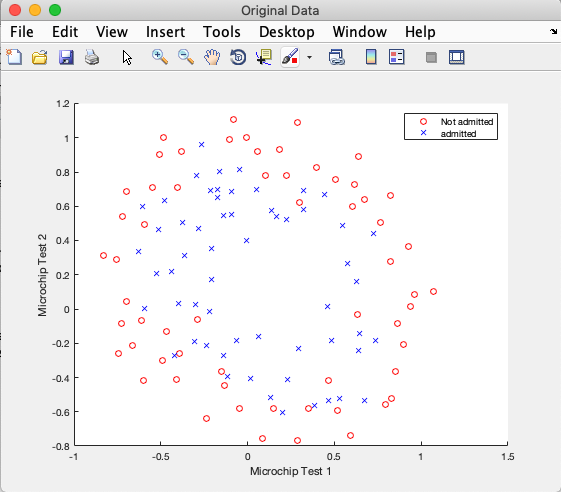
# Polynomial logistic regression

In this case, I got the theta through gradient descent.

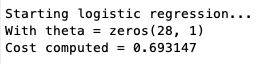
The plotting the decision boundary part conferenced the material:

https://github.com/j- pandeirada/mlcourse/blob/master/Class%203/plotDecisionBoundary.m

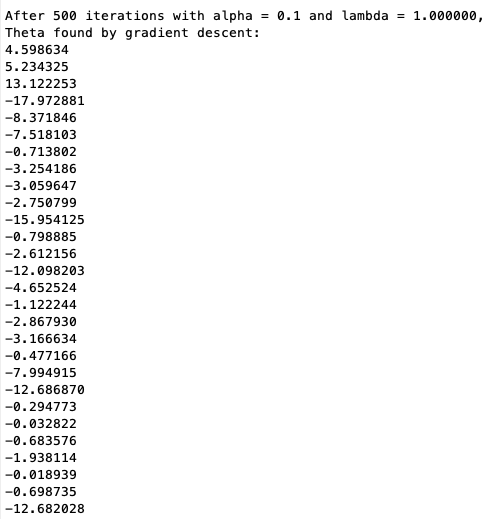
1. Scatter plot of the two features with y = 1 and y = 0 distinguished.



1. Cost at initial theta of all zeros.



1. Parameter values for theta.



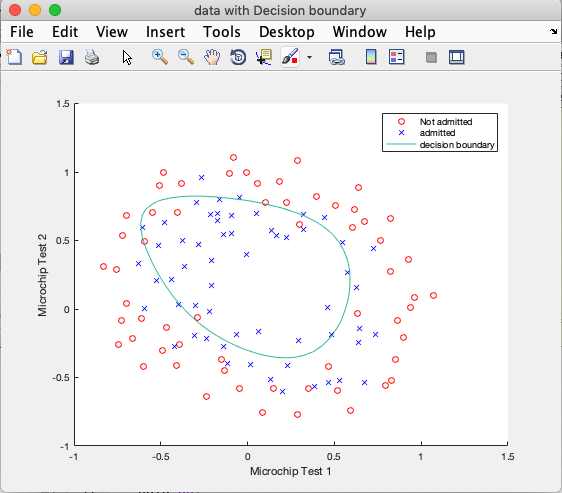
1. Cost after 500 iterations of regularized logistic regression algorithm.



1. Training accuracy.

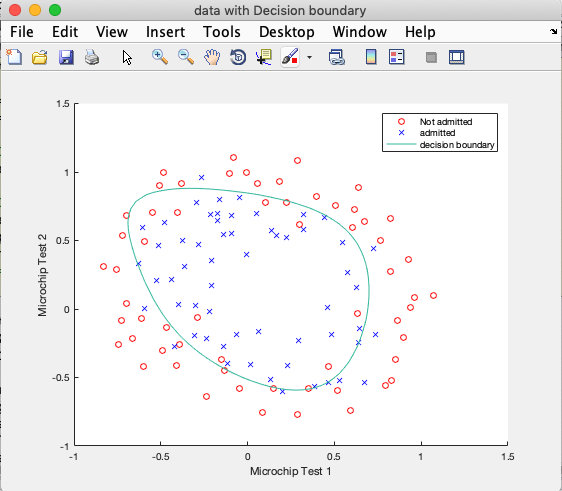


1. Plots of data along with decision boundary for at least three cases of .
2. With lambda = 1:



The accuracy has shown in section 3.5

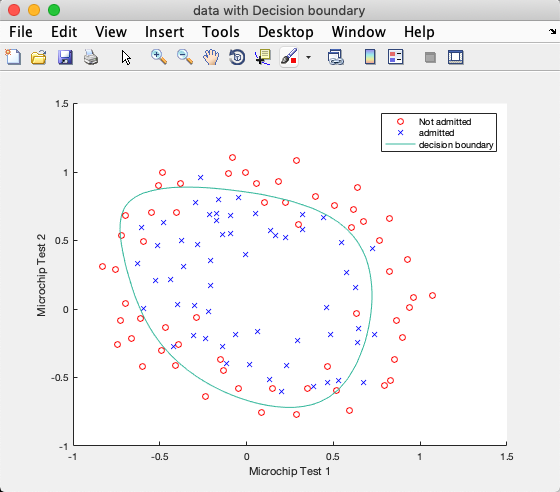
1. With lambda = 0.5



The cost and the accuracy is:



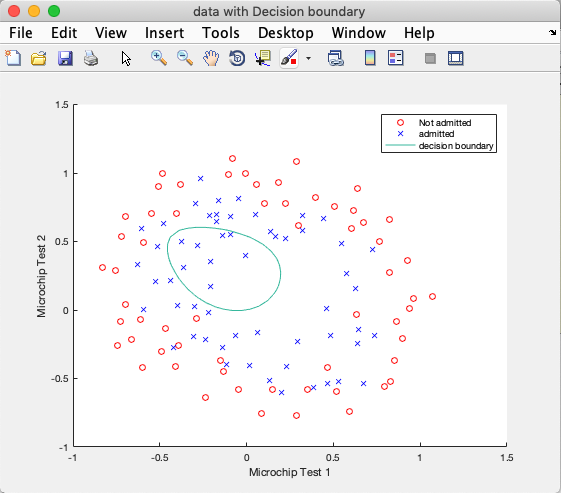
1. With lambda = 0



The cost and the accuracy is:



1. With lambda = 5



The cost and the accuracy is:



Analysis:

|  |  |  |
| --- | --- | --- |
| lambda | cost | accuracy |
| 0 | 0.497538 | 82.203390 |
| 0.5 | 0.392187 | 80.508475 |
| 1 | 0.815212 | 72.881356 |
| 5 | 2.062943 | 55.084746 |

We can see as the lambda decreasing, the accuracy is increasing. However, I got least cost at lambda = 0.5.

# Codes

1. main\_part\_1.m

%% Machine Learning Homework 4 part 1

% Author: Xinrun Zhang

% Time: 03/15/2019 14:39

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

%% Initialization

clear ; close all; clc

% Import the data;

fprintf('Initializing...\n')

fprintf('Reading the data...\n');

data = load('ex2data1.txt');

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

y = data(:, 3);  % y has values of 0 and 1.

% Initialize the theta vector;

theta = zeros(3, 1);

% Initialize the gradient descent parameters

iteration = 400;

alpha = 0.1;

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

%% Plot the original data;

data\_sorted = sortrows(data,3);

X\_0 = data\_sorted(1:40, [1, 2]);

X\_1 = data\_sorted(41:100, [1, 2]);

fprintf('Visualizing the original data...\n\n');

figure('Name','Original Data','NumberTitle','off');

scatter(X\_0(:, 1), X\_0(:, 2), 'o', 'r');

hold on;

scatter(X\_1(:, 1), X\_1(:, 2), 'x', 'b');

hold off;

xlabel('Exam 1 score')

ylabel('Exam 2 score')

legend('Not admitted', 'admitted');

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

%% Data processing

fprintf('Data processing...\n\n')

% Add a column of ones to x;

x = [ones(100, 1), X(:,1:2)];

fprintf('---------------------------------------------------------\n');

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

%% Logistic regression version 1

fprintf('Starting logistic regression version 1...');

J = computeCost(x, y, theta); %compute the cost

fprintf('\nWith theta = [0; 0; 0]\nCost computed = %f\n', J);

% Running gradient descent

theta = gradientDescent(x, y, theta, iteration, alpha);

% Print the output, including new theta and J;

fprintf('\nAfter 400 iterations with alpha = 0.1, ')

fprintf('\nTheta found by gradient descent:\n');

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

J = computeCost(x, y, theta);

fprintf('\nWith theta = [%f ; %f; %f]\nCost computed = %f\n', theta(1),theta(2),theta(3), J);

fprintf('---------------------------------------------------------\n');

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

%% Analysis

% I can't get proper desicion boundary from gradient descent algorithm

% which I wrote in the function gradientDescent.m.

% Therefore, I searched online and found another way to get the optimal

% The computeCost\_new.m is created to generate gradient.

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

%% Logistic regression version 2

fprintf('Starting logistic regression version 2...');

theta\_new = zeros(3, 1);

[~, grad] = computeCost\_new(theta\_new, x, y);

% run the function optimization algorithm

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

theta\_new= fminunc(@(t)computeCost\_new(t, x, y), theta\_new, options);

fprintf('\nTheta found by optimization algorithm:\n');

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

J = computeCost(x, y, theta\_new);

fprintf('\nWith theta\_v2 = [%f ; %f; %f]\nCost computed = %f\n', theta\_new(1),theta\_new(2),theta\_new(3), J);

fprintf('---------------------------------------------------------\n');

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

%% Plot the decision boundary

a = 30:0.1:100;

db\_1 = (-1./theta(3)).\*(theta(2).\*a + theta(1));

db\_2 = (-1./theta\_new(3)).\*(theta\_new(2).\*a + theta\_new(1));

figure('Name','Original Data','NumberTitle','off');

scatter(X\_0(:, 1), X\_0(:, 2), 'o', 'r');

hold on;

scatter(X\_1(:, 1), X\_1(:, 2), 'x', 'b');

hold on;

plot(a, db\_1);

hold on;

plot(a,db\_2);

xlabel('Exam 1 score')

ylabel('Exam 2 score')

legend('Not admitted', 'admitted', 'Decision Boundary 1', 'Decision Boundary 2');

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

%% Compute the accuracy

predict\_v1 = round(logsig(x \* theta));

predict\_v2 = round(logsig(x \* theta\_new));

accuracy\_v1 = mean( double(predict\_v1 == y) \* 100);

accuracy\_v2 = mean( double(predict\_v2 == y) \* 100);

fprintf('For version 1, the accuracy is %f\n', accuracy\_v1);

fprintf('For version 2, the accuracy is %f\n', accuracy\_v2);

fprintf('---------------------------------------------------------\n');

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

%% Compute the probability

prob = logsig(theta\_new(1) + theta\_new(2)\*45 + theta\_new(3)\*85);

fprintf('The probability of this student getting admitted is %f\n', prob);

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

1. main\_part\_2.m

%% Machine Learning Homework 4 part 2

% Author: Xinrun Zhang

% Time: 03/19/2019 17:28

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

%% Initialization

clear ; close all; clc

% Import the data;

fprintf('Initializing...\n')

fprintf('Reading the data...\n');

data = load('ex2data2.txt');

x = data(:, [1, 2]);

y = data(:, 3);  % y has values of 0 and 1.

% Initialize the theta vector;

theta = zeros(28, 1);

% Initialize the gradient descent parameters

iteration = 500;

alpha = 0.1;

lambda = 0.5;

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

%% Plot the original data;

data\_sorted = sortrows(data,3);

X\_0 = data\_sorted(1:60, [1, 2]);

X\_1 = data\_sorted(61:118, [1, 2]);

fprintf('Visualizing the original data...\n\n');

figure('Name','Original Data','NumberTitle','off');

scatter(X\_0(:, 1), X\_0(:, 2), 'o', 'r');

hold on;

scatter(X\_1(:, 1), X\_1(:, 2), 'x', 'b');

hold off;

xlabel('Microchip Test 1')

ylabel('Microchip Test 2')

legend('Not admitted', 'admitted');

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

%% Data processing

fprintf('Data processing...\n\n')

% rebuild the x

% 1 1

% 2 11 2

% 3 21 12 3

% 4 32 23 4

% 5 43 42 24 34 5

% 6 54 53 52 25 35 45 6

x\_2 = [x(:,1).^2, x(:,1).\*x(:,2), x(:,2).^2];

x\_3 = [x(:,1).^3, (x(:,1).^2).\*x(:,2), x(:,1).\*(x(:,2).^2), x(:,2).^3 ];

x\_4 = [x(:,1).^4, (x(:,1).^3).\*(x(:,2).^2), (x(:,1).^2).\*(x(:,2).^3), x(:,2).^4];

x\_5 = [x(:,1).^5, (x(:,1).^4).\*(x(:,2).^3), (x(:,1).^4).\*(x(:,2).^2), (x(:,1).^2).\*(x(:,2).^4), (x(:,1).^3).\*(x(:,2).^4), x(:,2).^5];

x\_6 = [x(:,1).^6, (x(:,1).^5).\*(x(:,2).^4), (x(:,1).^5).\*(x(:,2).^3), (x(:,1).^5).\*(x(:,2).^2), (x(:,1).^2).\*(x(:,2).^5), (x(:,1).^3).\*(x(:,2).^5), (x(:,1).^4).\*(x(:,2).^5), x(:,2).^6];

x = [x(:, 1:2), x\_2(:, 1:3), x\_3(:, 1:4), x\_4(:, 1:4), x\_5(:, 1:6), x\_6(:, 1:8)];

% normalization

%x = normalization(x);

% Add a column of ones to x;

X = [ones(118, 1), x(:,1:27)];

fprintf('---------------------------------------------------------\n');

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

%% Logistic regression

fprintf('Starting logistic regression...');

J = computeCost\_part2(X, y, theta, lambda); %compute the cost

fprintf('\nWith theta = zeros(28, 1)\nCost computed = %f\n', J);

% Running gradient descent

theta = gradientDescent\_part2(X, y, theta, iteration, alpha, lambda);

% Print the output, including new theta and J;

fprintf('\nAfter 500 iterations with alpha = 0.1 and lambda = %f, ', lambda);

fprintf('\nTheta found by gradient descent:\n');

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

J = computeCost(X, y, theta);

fprintf('\nWith this theta, \nCost computed = %f\n',J);

fprintf('---------------------------------------------------------\n');

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

%% Plot the decision boundary

% define two arrays

u = linspace(-1, 1.5, 50);

v = linspace(-1, 1.5, 50);

% g\_2 = [u.^2, u.\*v, v.^2];

% g\_3 = [u.^3, (u.^2).\*v, u.\*(v.^2), v.^3 ];

% g\_4 = [u.^4, (u.^3).\*(v.^2), (u.^2).\*(v.^3), v.^4];

% g\_5 = [u.^5, (u.^4).\*(v.^3), (u.^4).\*(v.^2), (u.^2).\*(v.^4), (u.^3).\*(v.^4), v.^5];

% g\_6 = [u.^6, (u.^5).\*(v.^4), (u.^5).\*(v.^3), (u.^5).\*(v.^2), (u.^2).\*(v.^5), (u.^3).\*(v.^5), (u.^4).\*(v.^5), v.^6];

% g = [u, v, g\_2(:, 1:3), g\_3(:, 1:4), g\_4(:, 1:4), g\_5(:, 1:6), g\_6(:, 1:8)];

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

% need to specify the range [0, 0]

figure('Name','data with Decision boundary','NumberTitle','off');

scatter(X\_0(:, 1), X\_0(:, 2), 'o', 'r');

hold on;

scatter(X\_1(:, 1), X\_1(:, 2), 'x', 'b');

hold on;

contour(u, v, z, [0, 0])

hold off;

xlabel('Microchip Test 1')

ylabel('Microchip Test 2')

legend('Not admitted', 'admitted', 'decision boundary');

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

%% compute the accuracy

predict = round(logsig(X \* theta));

accuracy = mean( double(predict == y) \* 100);

fprintf('The accuracy is %f\n', accuracy);

fprintf('---------------------------------------------------------\n');

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

1. computeCost.m

function J = computeCost(x, y, theta)

m = length(y);

h = logsig(x \* theta);

J = (-1/m) \* (y' \* log(h + 0.01) + (1 - y)' \* log(1 - h + 0.01));

end

1. computeCost\_new.m

function [ J, grad ] = computeCost\_new(theta, x, y)

m = length(y);

h = logsig(x \* theta);

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

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

end

1. computeCost\_part2.m

function J = computeCost\_part2(x, y, theta, lambda)

m = length(y);

h = logsig(x \* theta);

J = (-1/m) \* (y' \* log(h) + (1 - y)' \* log(1 - h)) + (lambda / (2 \* m) ) \* (theta' \* theta);

end

1. normalization.m

function x = normalization(x)

x\_mean = mean(x);

x = (x - x\_mean)./ (std(x));

end

1. gradientDescent.m

function theta = gradientDescent(x, y, theta, iteration, alpha)

m = length(y);

for i = 1:iteration

    h = logsig(x \* theta);

    theta = theta - (alpha / m) \* (x' \* (h - y));

end

end

1. gradientDescent\_part2.m

function theta = gradientDescent\_part2(x, y, theta, iteration, alpha, lambda)

m = length(y);

for i = 1:iteration

    h = logsig(x \* theta);

    foo = [0; theta(2:28, 1)];

    theta = theta - (alpha / m) \* (x' \* (h - y)) + (lambda / m) .\* foo ;

end

end

1. mapFeatures.m

function out = mapFeature(X1, X2)

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