Name: Xinrun Zhang

Title: ML Homework 1 Report

Instructor: Prof. Kaliappan Gopalan

Time: 11/02/2019

## 1. Files

a) main.m – Includes the main program of the project.

b) computeCost.m – This function is used to compute the cost between hypothesis and input y, and can be both used on linear regression and polynomial regression.

c) normalizing.m – This function is used in univariate linear regression to normalize the input x.

d) featureScaling.m – This function is used in polynomial regression to do the feature scaling.

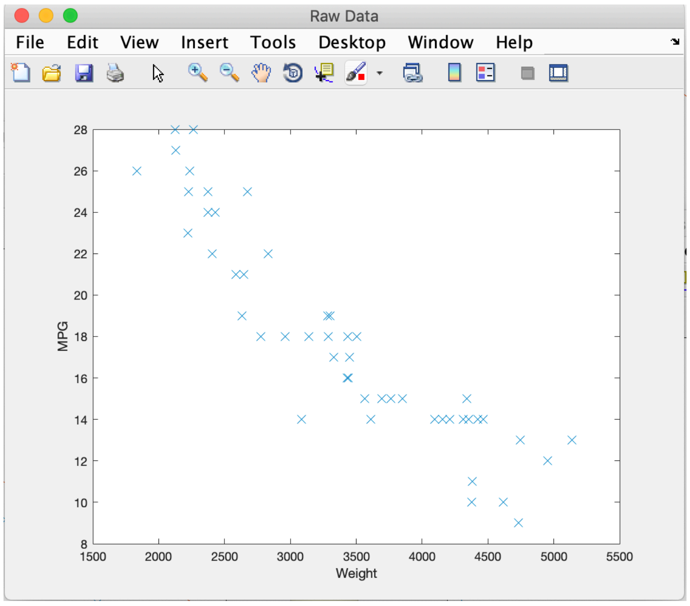
e) gradientDescent.m – This function is used in univariate linear regression to execute the gradient descent to find proper parameters.

f) gradientDescentPyn.m - This function is used in polynomial regression to execute the gradient descent to find proper parameters.

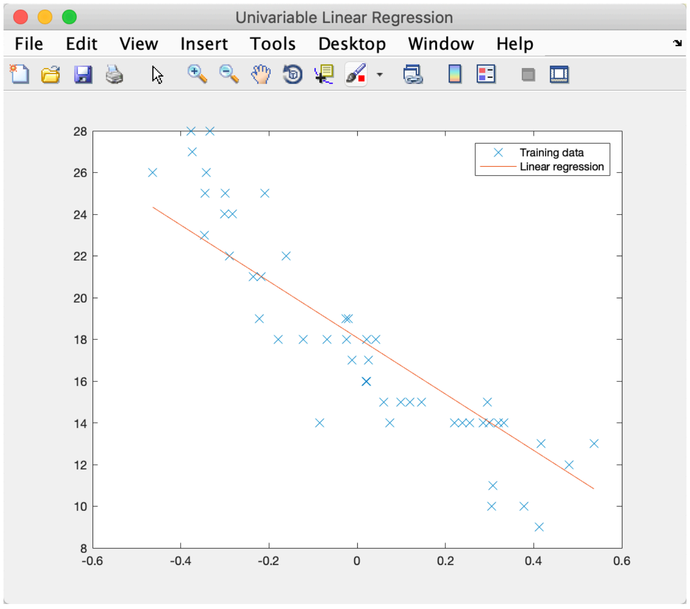
g) All the codes can be found in these files and all the details can be found in the comments.

## 2. Univariate Linear Regression

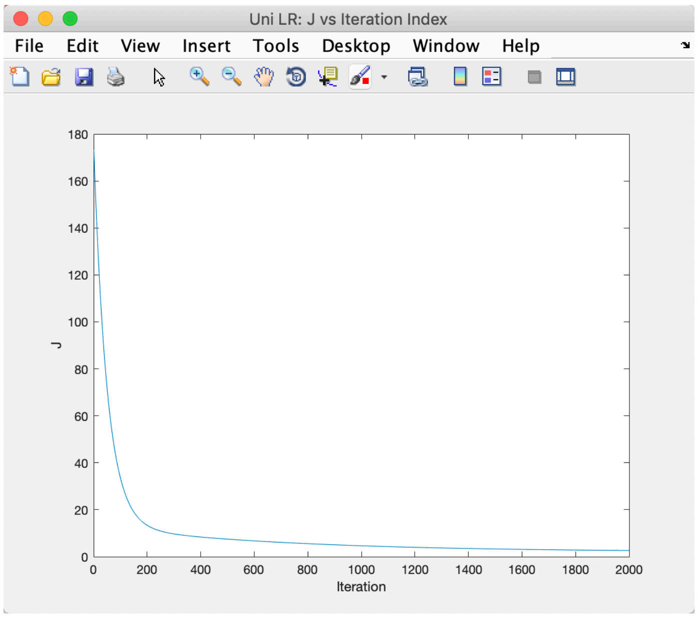
a) Scatter plot of y vs. x. label axes with 'weight' and 'MPG'



b) Linear hypothesis plot – the final straight line – on the scatter plot of y vs. x



c) Plot of J vs. Iteration index



d) Minimum J and the hypothesis parameters

```
With theta = [18.080000 ; -13.501455]
Cost computed = 2.641001
```

e) Predicted output for the weight x = 3100

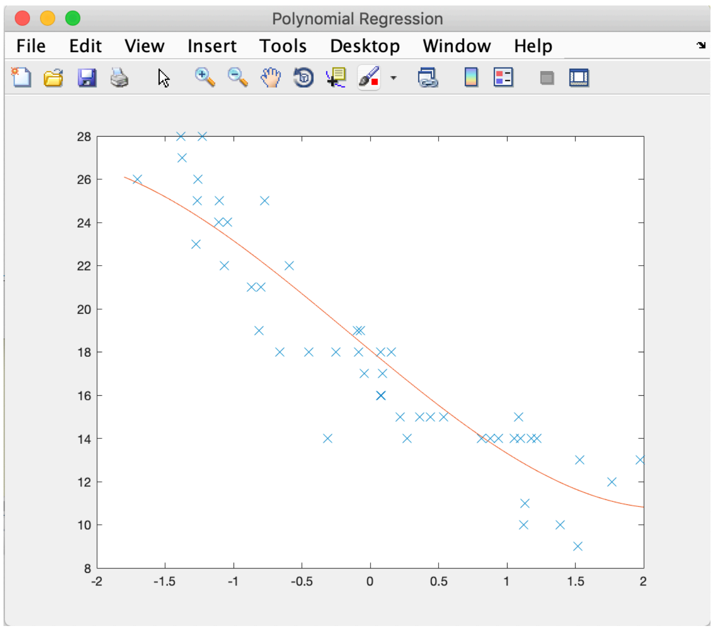
```
With x = 3100, the predict y = 17.534468
```

f) Situation that without normalizing. The gradient descent with $\alpha = 0.1$ and iteration = 2000 will not convert, which means we have to set $\alpha$ very small, like $\alpha = 0.00000001$.

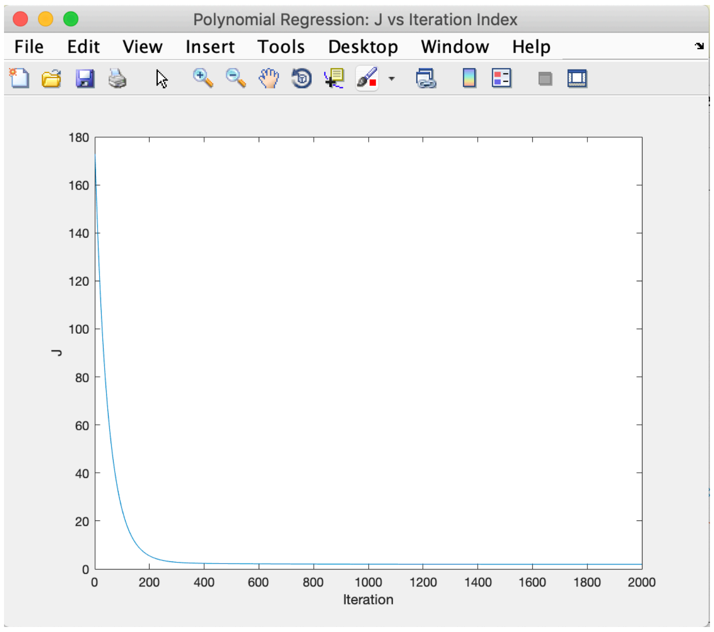
## 3. Polynomial Regression and Feature scaling

a) Cubic hypothesis plot on the scatter plot of y vs. x

I used regularization here to reduce the impact of theta(3) and theta(4) to get a better shape of curve. Below is what I get in lambda = 50.



b) Plot of J vs. Iteration index



c) Minimum J and theta, the hypothesis parameters

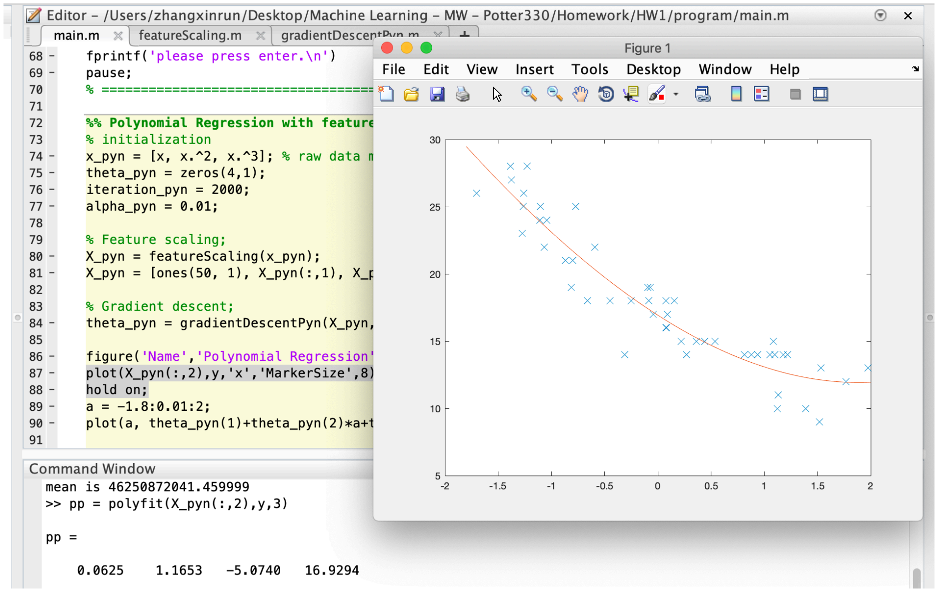
```
With theta = [18.080000 ; -5.250971; 0.151673; 0.329294]
Cost computed = 1.927384
```

d) Predicted output for x = 3100

```
With x = 3100, the predict y = 19.641106
```

e) Polynomial curve fitting with build-in function

I've also tried the build-in function in MATLAB to fit the dataset with polynomial curve. The function which I used is called polyfit() function, which returns the values of parameters theta. As you can see in the screenshot, the shape of curve is much better than the curve which is generated by the gradient descent.



Editor – /Users/zhangxinrun/Desktop/Machine Learning – MW – Potter330/Homework/HW1/program/main.m

main.m | featureScaling.m | gradientDescentPyn.m

```
fprintf('please press enter.\n')
pause;
% ==================================

%% Polynomial Regression with feature
% initialization
x_pyn = [x, x.^2, x.^3]; % raw data m
theta_pyn = zeros(4,1);
iteration_pyn = 2000;
alpha_pyn = 0.01;

% Feature scaling;
X_pyn = featureScaling(x_pyn);
X_pyn = [ones(50, 1), X_pyn(:,1), X_p

% Gradient descent;
theta_pyn = gradientDescentPyn(X_pyn,

figure('Name','Polynomial Regression'
plot(X_pyn(:,2),y,'x','MarkerSize',8)
hold on;
a = -1.8:0.01:2;
plot(a, theta_pyn(1)+theta_pyn(2)*a+t
```

Command Window

```
mean is 46250872041.459999
>> pp = polyfit(X_pyn(:,2),y,3)

pp =

    0.0625    1.1653   -5.0740   16.9294
```

I guess the reason that I can't get the proper fitting curve is the overfitting problem. By the observation of J, which is very small and high bias from two sides of the curve, the overfitting has occurred. Or, maybe the J function converted into local minimum with the specific alpha and the number of iteration.

## 4. Code

### a) main.m

```
     %% Machine Learning Homework 1

     % Author: Xinrun Zhang

     % Time: 02/09/2019 18:45

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


     %% Initialization

clear ; close all; clc


     % Import the data;

fprintf('Initializing...\n')

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

A = xlsread('AutoData_HW1.xlsx');


     % Extract the column 4 as input x and column 6 as input y;

x = A(:,4);

y = A(:,6);


     % Plot the data;

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

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

plot(x,y,'x','MarkerSize',8);

ylabel('MPG');

xlabel('Weight');


     % Initialize the theta vector;

theta = zeros(2, 1);


     % Initialize the gradient descent parameters
```

```
iteration = 2000;

alpha = 0.01;

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


     %% Data processing

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

     % choose normalizing the data or not;

choice = input('Do you want to normalize the data? 1.y / 0.n\n');

switch choice

    case 1

        x_nor = normalizing(x); % Call the normalizing function;

        X = [ones(50, 1), x_nor(:,1)]; % Add a column of ones to xnor;

    case 0

        X = [ones(50, 1), x(:,1)]; % Add a column of ones to x;

end

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


     %% Univariable linear regression

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

fprintf('\nWith theta = [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('\nTheta found by gradient descent:\n');

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

J = computeCost(X, y, theta);

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


figure('Name','Univariable Linear Regression','NumberTitle','off');

plot(x_nor,y,'x','MarkerSize',8);

     hold on; % keep previous plot visible

plot(X(:,2), X*theta, '-') % plot the hypothesis
```

```
legend('Training data', 'Linear regression');


x_predict = theta(1) + theta(2)*((3500 - mean(x))/(max(x) - min(x)));

fprintf('\nWith x = 3100, the predict y = %f\n',x_predict);

fprintf('Now, the program is paused.\n');

fprintf('If you want to go to polynomial regression,\n')

fprintf('please press enter.\n')

pause;

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


     %% Polynomial Regression with feature scaling

     % initialization

x_pyn = [x, x.^2, x.^3]; % raw data matrix;

theta_pyn = zeros(4, 1);

iteration_pyn = 2000;

alpha_pyn = 0.01;

x_predict = 3100;


     % Feature scaling;

[X_pyn, x_predict]= featureScaling(x_pyn, x_predict);

X_pyn = [ones(50, 1), X_pyn(:,1), X_pyn(:,2), X_pyn(:,3)];


     % Gradient descent;

theta_pyn = gradientDescentPyn(X_pyn, y, theta_pyn, iteration_pyn, alpha_pyn);


figure('Name','Polynomial Regression','NumberTitle','off');

plot(X_pyn(:,2),y,'x','MarkerSize',8);

hold on;

a = -1.8:0.01:2;

plot(a, theta_pyn(1)+theta_pyn(2)*a+theta_pyn(3)*(a.^2)+theta_pyn(4)*(a.^3), '-');


     % print the result theta;

fprintf('\nTheta found by gradient descent:\n');
fprintf('%f\n', theta_pyn);

J = computeCost(X_pyn, y, theta_pyn);

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

theta_pyn(1),theta_pyn(2),theta_pyn(3),theta_pyn(4), J);


     % compute the x_predict and print the result;

x_predict = theta_pyn(1) + theta_pyn(2)*(x_predict) + theta_pyn(3)*(x_predict.^2) +

theta_pyn(4)*(x_predict.^3);

fprintf('\nWith x = 3100, the predict y = %f\n',x_predict');


     % =====================================================================
```

## b) <u>computeCost.m</u>

```
function J = computeCost(X, y, theta)


m = length(y);

J = 0;


hypothesis = X * theta;

squareError = (hypothesis - y).^2;

J = 1/ (2*m) * sum(squareError);

end
```

## c) <u>normalizing.m</u>

```
function x_nor = normalizing(x)

x_nor = x;

x_max = max(x);

x_min = min(x);

x_mean = mean(x);


for i = 1:50

           x_nor(i) = (x(i) - x_mean)/(x_max - x_min);
```

```
end
```

## d) featureScaling.m

```
function [X_pyn, x_predict] = featureScaling(x_pyn, x_predict)

X_pyn = x_pyn;

Sd = std(x_pyn); % compute the standard deviation of each column;

me = mean(x_pyn); % compute the mean value of each column;


for i = 1:3

       X_pyn(:, i) = (x_pyn(:, i) - me(i)) / Sd(i);

end


x_predict = (x_predict - me(1)) / Sd(1);


end
```

## e) gradientDescent.m

```
function [theta, J_history ] = gradientDescent(X, y, theta, iteration, alpha)


m = length(y);

J_history = zeros(iteration, 1);

x = X(:,2);


for i = 1:iteration

    hypothesis = theta(1) + (theta(2)*x); %prevent theta(1) from affecting by theta(2);


    theta_zero = theta(1) - (alpha / m) * sum(hypothesis - y);

    theta_one = theta(2) - (alpha /m)* sum((hypothesis - y) .*x);


    theta = [theta_zero; theta_one];

    J_history(i) = computeCost(X, y, theta);
```

```
end


figure('Name','Uni LR: J vs Iteration Index','NumberTitle','off');

plot(J_history); % plot J vs iteration index

ylabel('J');

xlabel('Iteration');

end
```

## f) gradientDescentPyn.m

```
function [theta_pyn, j_pyn_history] = gradientDescentPyn(X_pyn, y, theta_pyn, iteration_pyn,

alpha_pyn)

m = 50;

j_pyn_history = zeros(iteration_pyn, 1);

lambda = 30; %* (1 - alpha_pyn * lambda / m)

% regularization is used here to get a better shape curve

x1 = X_pyn(:,2);

x2 = X_pyn(:,3);

x3 = X_pyn(:,4);


for i = 1:iteration_pyn

    hyp = theta_pyn(1) + theta_pyn(2)*x1 + theta_pyn(3)*x2 + theta_pyn(4)*x3;


    theta_zero = theta_pyn(1) - alpha_pyn * (1/m) * sum(hyp - y);

    theta_one = theta_pyn(2) - alpha_pyn * (1/m) * sum((hyp - y).*x1);

    theta_two = theta_pyn(3) * (1 - alpha_pyn * lambda / m) - alpha_pyn * (1/m) * sum((hyp -

y).*x2); % regularized theta_two

    theta_three = theta_pyn(4) * (1 - alpha_pyn * lambda / m) - alpha_pyn * (1/m) * sum((hyp

- y).*x3); % regularized theta_three


    theta_pyn = [theta_zero; theta_one; theta_two; theta_three];

    j_pyn_history(i) = computeCost(X_pyn, y, theta_pyn);

end

figure('Name','Polynomial Regression: J vs Iteration Index','NumberTitle','off');
```

```
plot(j_pyn_history); % plot J vs iteration index

ylabel('J');

xlabel('Iteration');

end
```