Name: Xinrun Zhang

ML Homework 2 Report

Instructor: Prof. Kaliappan Gopalan

Time: 21/02/2019

1. Files

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

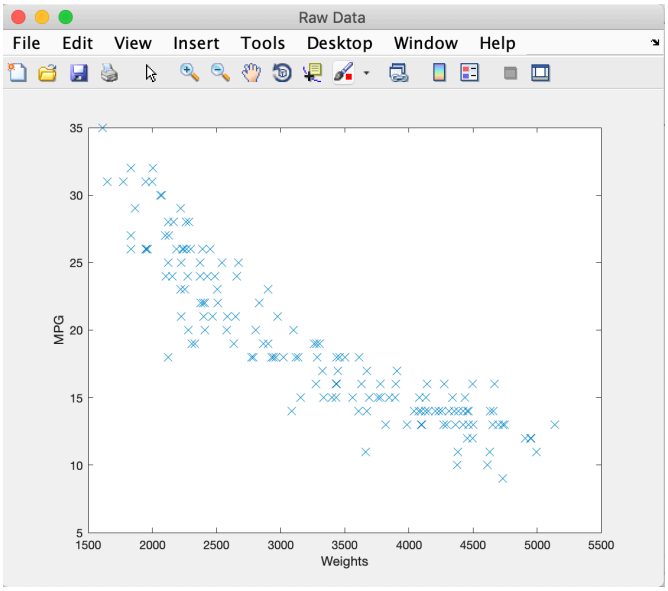
b) computeCost.m – This function is used to compute the cost between hypothesis and input y.

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

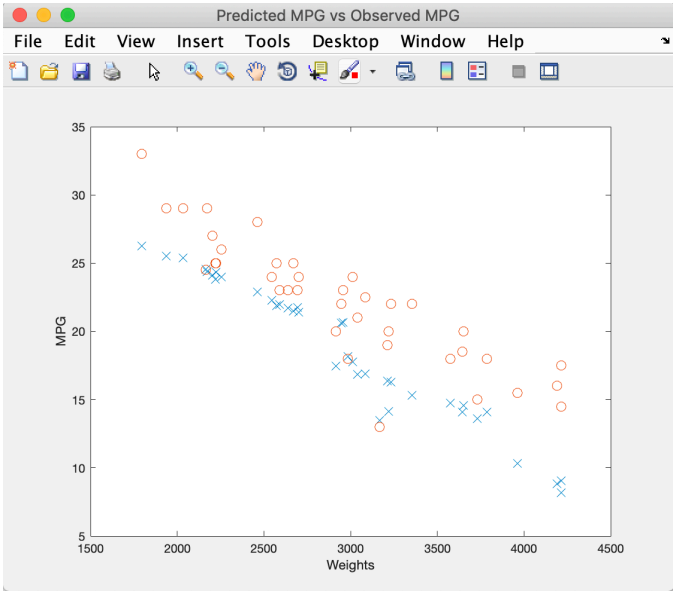
d) gradientDescent.m – This function is used in univariate linear regression to execute the gradient descent to find proper parameters.

2. Multivariate Linear Regression with Gradient Descent

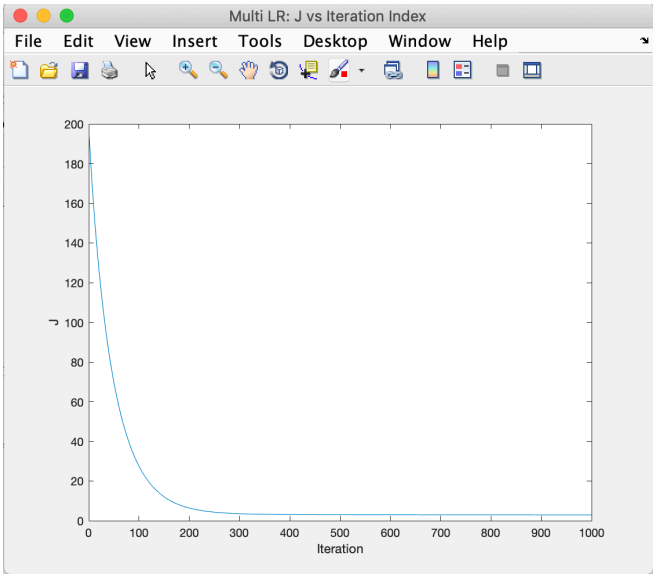
1. Scatter plot of *y* vs. *x*, where *x* is still the weight (as in HW 1)



2. Plot of predicted MPG against observed MPG for the validation data



3. Plot of J vs. Iteration index for the training data



4. Minimum J and the hypothesis parameters from gradient descent code

```
Theta found by gradient descent:
19.024179
-1.095249
-1.274129
-0.275286
-2.802676

With theta = [19.024179 ; -1.095249; -1.274129; -0.275286; -2.802676]
Cost computed = 2.925580
```

5. Mean squared error err_cv

```
The mean suqare error err_cv is: 17.800543
```

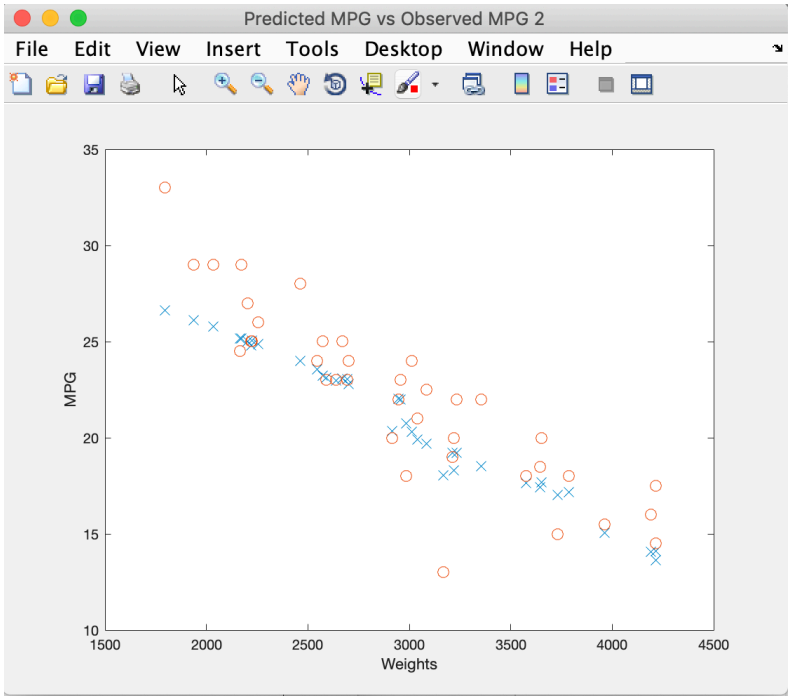
3. Multivariate Linear Regression with Normal Equation

1. Hypothesis parameters (theta) for the closed form

```
Theta found by normal equation:
35.486442
-0.296053
-0.010934
0.000843
-0.003744

With theta = [35.486442 ; -0.296053; -0.010934; 0.000843; -0.003744]
Cost computed = 2.876415
```

2. Plot of predicted MPG against observed MPG for the validation data



3. I didn't apply feature scaling (normalization) because when I searched this question on Google I found:

> Well, in the second article there is a sentence:
>
> > Note that before conducting linear regression, you should normalize the data. One way is $\frac{x_i - mean(x)}{Range(x)}$, and some use $sd(x)$ as the denominator. Both work.
>
> But is not said that it applies specifically to normal equations. And in gradient descent section there is nothing said about normalization. So I suppose it was a small mistake to include that sentence in Normal Equation section instead of Gradient Descent.
>
> Anyway Andrew Ng is pretty authoritative on machine learning topic, so you can rely on his words:
>
> > Using this formula does not require any feature scaling, and you will get an exact solution in one calculation: there is no 'loop until convergence' like in gradient descent.

The result of normal equation is not as good as I expected, there are maybe some method to get a better result.

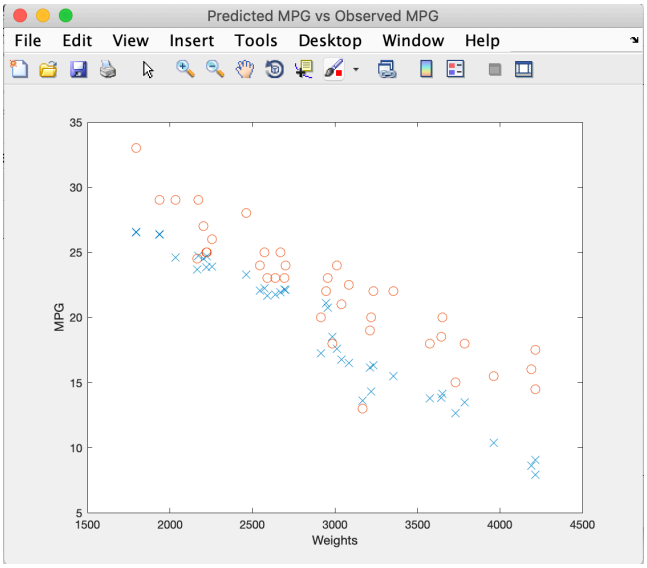
4. Optional Exercise

1. Minimum J and the hypothesis parameters from gradient descent code

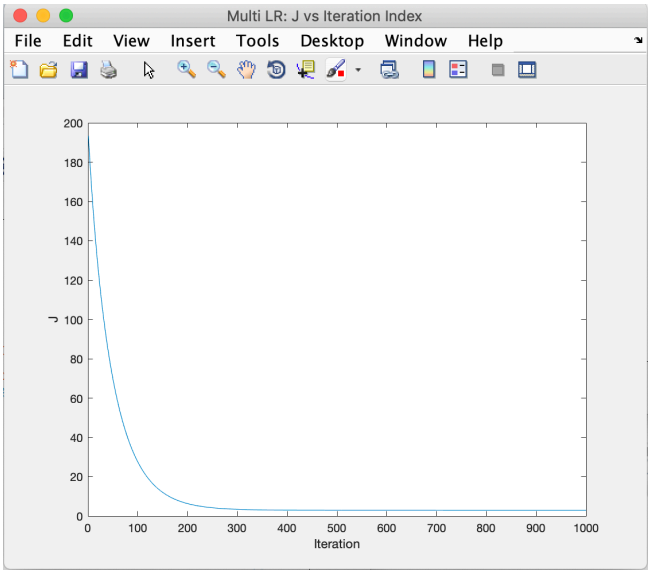
```
Optional exercise Theta found by gradient descent:
19.024179
-1.166674
-1.426639
-0.665490
-2.627616
-0.597876

With theta = [19.024179 ; -1.166674; -1.426639; -0.665490; -2.627616; -0.597876]
Cost computed = 2.886151
```

2. Plot of predicted MPG against observed MPG for the validation data



3. Plot of J vs. Iteration index for the training data



4. The result of 5 features LR is pretty closed to the situation that with 4 features LR. I believe that we can improve the result by adding more data to the training set instead of adding new features.

5. Codes

a) main.m

```
%% Machine Learning Homework 2

% Author: Xinrun Zhang

% Time: 02/20/2019 14?39

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


%% Initialization

clear ; close all; clc


% Import the data;

fprintf('Initializing...\n')

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

A = xlsread('AutoData_HW2.xlsx');


% Extract the data, generate training data and validation data;
```

```
x = A(1:160,2:5);

y = A(1:160,1);

x_val = A(161:200,2:5);

y_val = A(161:200,1);


% Plot the data;

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

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

plot(x(:,4),y,'x','MarkerSize',8);

ylabel('MPG');

xlabel('Weights');


% Initialize the theta vector;

theta = zeros(5, 1);


% Initialize the gradient descent parameters

iteration = 1000;

alpha = 0.01;

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


%% Data processing

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


% Call the normalizing function;

x_nor = normalizing(x);

x_val_nor = normalizing(x_val);


% Add a column of ones to xnor;

X = [ones(160, 1), x_nor(:,1:4)];

X_val = [ones(40, 1), x_val_nor(:,1:4)];


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

```
%% Multivariate 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; %f; %f; %f]\nCost computed = %f\n',

theta(1),theta(2),theta(3),theta(4),theta(5), J);

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


%% Validation

% Generate predict data

y_prd = X_val*theta;


% Calculate the mean squared error err_cv

err_cv = immse(y_prd,y_val);

fprintf('The mean suqare error err_cv is: %f\n',err_cv);


% Plot of predicted MPG against observed MPG for the validation data

figure('Name','Predicted MPG vs Observed MPG','NumberTitle','off');

plot(x_val(:,4),y_prd,'x','MarkerSize',8);

ylabel('MPG');

xlabel('Weights');

hold on;

plot(x_val(:,4),y_val,'o','MarkerSize',8);

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


%% Normal equation
```

```
% Data_processing

X_ne = [ones(160,1), x(:,1:4)];

X_ne_val = [ones(40,1), x_val(:,1:4)];

% Normal equation

theta_ne = pinv(X_ne)* y;

%theta_ne = (X'*X)^(-1)* X' * y;

y_prd_ne = X_ne_val*theta_ne;


% Print the output, including new theta and J;

fprintf('\nTheta found by normal equation:\n');

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


J = computeCost(X_ne, y, theta_ne);

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

theta_ne(1),theta_ne(2),theta_ne(3),theta_ne(4),theta_ne(5), J);


% Plot of predicted MPG against observed MPG for the validation data

figure('Name','Predicted MPG vs Observed MPG 2','NumberTitle','off');

plot(x_val(:,4),y_prd_ne,'x','MarkerSize',8);

ylabel('MPG');

xlabel('Weights');

hold on;

plot(x_val(:,4),y_val,'o','MarkerSize',8);

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


%% Optional exercise

% Extract the data, generate training data and validation data;

x_oe = A(1:160,2:6);

y_oe = A(1:160,1);

x_oe_val = A(161:200,2:6);

y_oe_val = A(161:200,1);


% Initialize the theta vector;
```

```
theta_oe = zeros(6, 1);


% Initialize the gradient descent parameters

iteration = 1000;

alpha = 0.01;

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


% Data processing

x_oe_nor = normalizing(x_oe);

X_oe = [ones(160, 1), x_oe_nor(:,1:5)];


x_oe_val_nor = normalizing(x_oe_val);

X_oe_val = [ones(40, 1), x_oe_val_nor(:,1:5)];


% Gradient Descent

theta_oe = gradientDescent(X_oe, y, theta_oe, iteration, alpha);


% Print the output, including new theta and J;

fprintf('\nOptional exercise Theta found by gradient descent:\n');

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


J = computeCost(X_oe, y_oe, theta_oe);

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

theta_oe(1),theta_oe(2),theta_oe(3),theta_oe(4),theta_oe(5), theta_oe(6), J);

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

% Validation

% Generate predict data

y_prd = X_oe_val*theta_oe;


% Plot of predicted MPG against observed MPG for the validation data

figure('Name','Predicted MPG vs Observed MPG','NumberTitle','off');

plot(x_oe_val(:,4),y_prd,'x','MarkerSize',8);

ylabel('MPG');
```

```
xlabel('Weights');

hold on;

plot(x_oe_val(:,4),y_val,'o','MarkerSize',8);

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

b) computeCost.m

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

m = length(y);

J = 0;


hypothesis = X * theta;

J = (1/ (2*m)) * (hypothesis - y)' * (hypothesis - y);

end
```

c) gradientDescent.m

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


m = length(y);

J_history = zeros(iteration, 1);


for i = 1:iteration

    hypothesis = X*theta;

    theta = theta - (alpha / m) * (X' * (hypothesis - y));

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


end


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

plot(J_history); % plot J vs iteration index

ylabel('J');

xlabel('Iteration');

end
```

d) normalizing.m

```
function x_nor = normalizing(x)

x_nor = x;

%delta = x_max - x_min;

x_mean = mean(x);

x_nor = (x - x_mean)./ (std(x));


end
```