A Developer Diary

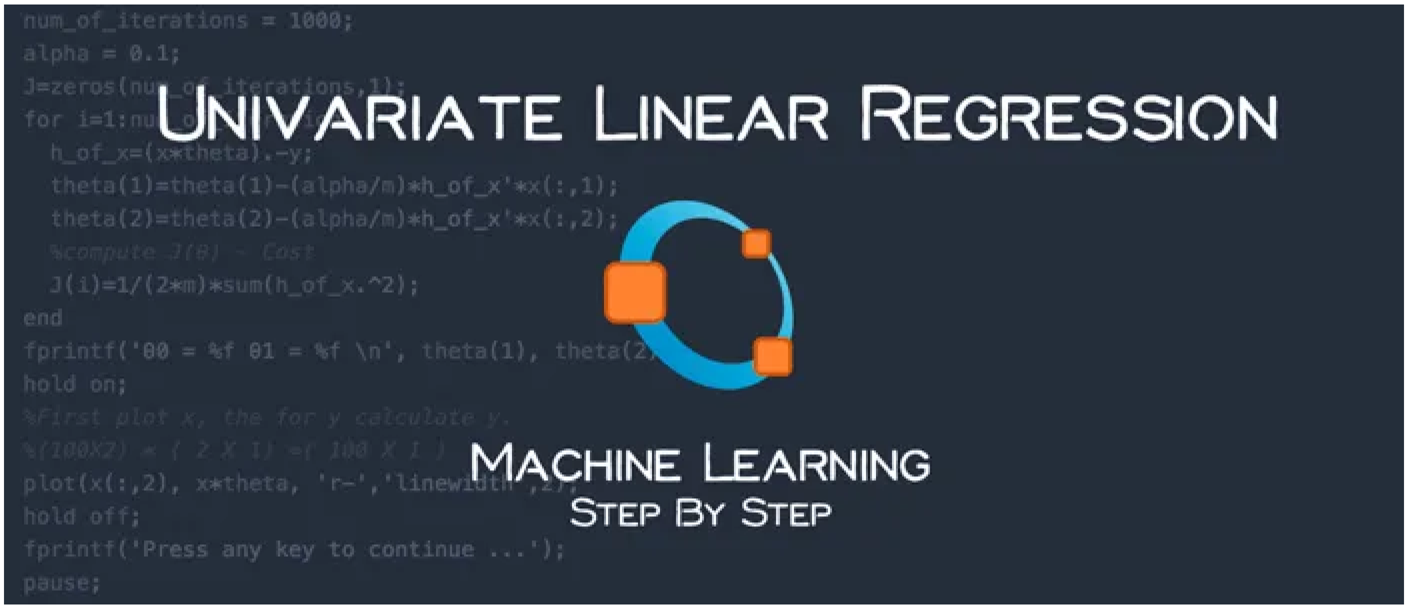
{about:"code learn and share"}

Home Data Science Java JavaScript jBPM Tools Tips

About

October 18, 2016 By Abhisek Jana — Leave a Comment (Edit)

Univariate Linear Regression using Octave – Machine Learning Step by Step



Univariate Linear Regression is probably the most simple form of Machine Learning. Understanding the theory part is very important and then using the concept in programming is also very critical.In this **Univariate Linear Regression using Octave – Machine Learning Step by Step** tutorial we will see how to implement this using Octave.Even if we understand something mathematically, understanding the implementation can be tedious. Since many professor/researcher uses Octave/MatLab for teaching Machine Learning, it could be very well the the first tool you might be using to understand machine learning.This tutorial will help you to get familiar with Octave/MATLAB and understand the implementation in much easier way, without spending lot of time in Octave/MATLAB.

Objective:

Implement Univariate Linear Regression using Gradient Descent and Normal Equation in Octave/MATLAB.

Case Study:

We will use a data set from UCI (http://archive.ics.uci.edu/ml/) regarding Car millage. I have truncated the training set to only 100 data set and uploaded it to the git repository (Link given below). The data set has two columns:

1. weight of the cars in thousand lb – [x_1]
2. mpg for the cars – [y]

We will first visualize the relationship between the weight and mpg. Then use the data in order to predict mpg of a car by providing the weight of the car using Linear Regression.

Data Structure:

Here is the data, (weight,mpg). The weight is in (1000 X lb) factor. So `3.5040 = 3.5040 X 1000lb = 3504lb`

```
3.5040,18.00
3.6930,15.00
3.4360,18.00
3.4330,16.00
3.4490,17.00
4.3410,15.00
4.3540,14.00
....
```

Gradient Descent:

In this tutorial we will use the most simple form of Gradient Descent. Just to recap, lets go through the equations:

1. Linear Regression hypothesis function:
   $\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$
2. Cost Function:
   $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

3. Gradient Descent Generic Function :
   $\theta_j := \theta_j - \alpha \frac{d}{d\theta_j} J(\theta_0, \theta_1)$
4. Gradient Descent for Linear Regression :
   Assuming, $x_0 = 1$
   $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x^{(i)}) - y^{(i)}) . x_j^{(i)})$
   for j= 1,2,….,n

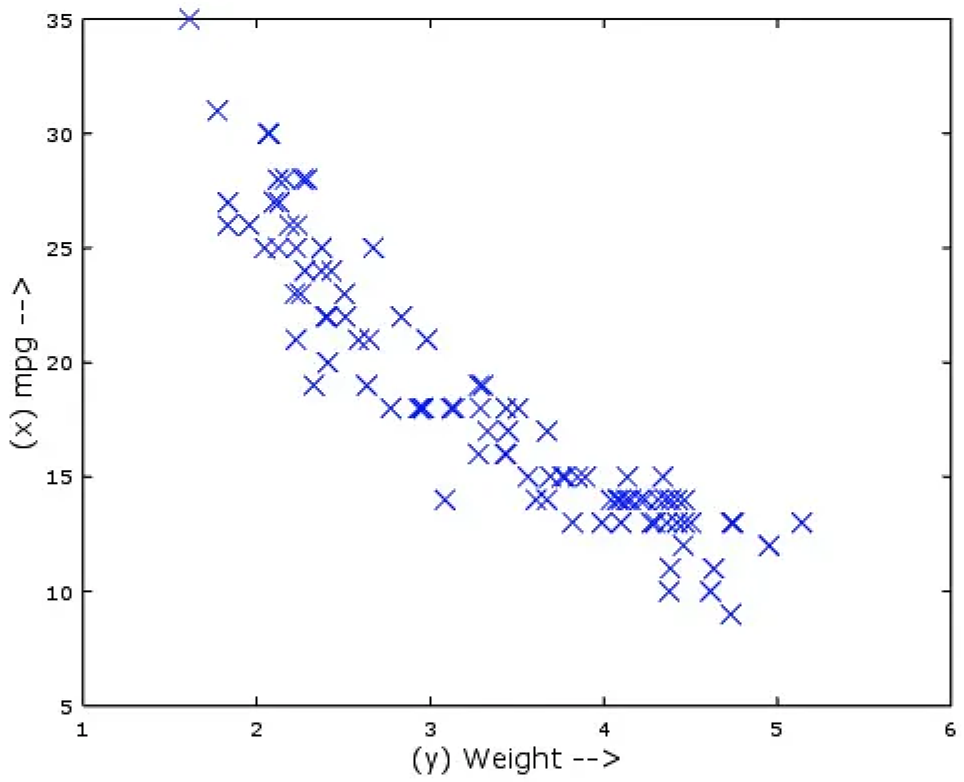
Visualize the Data:

Our first step would be to visualize the data. Lets open Octave/MATLAB and create a file named `main.m`. Make sure the `data.txt` is in the same folder. Load the data using `load()` function. Then assign the first column to `x` variable and the 2nd column to `y`. The `x` and `y` matrix both will have 100 rows and 1 col.

Next, use the `plot()` function to draw the plot. Also set the labels for the axis.

```
car_data=load('data.txt');

x=car_data(:,1); % X would have 100 col and 1 row
y=car_data(:,2); % y would have 100 col and 1 row

figure;
plot(x,y,'bx','MarkerSize',10);
xlabel('(y) Weight -->','fontsize',14);
ylabel('(x) mpg -->','fontsize',14);
```

Once you execute the above code the following chart will be displayed. You can see how the weight of the cars are inversely related to the mpg. The car with weight `5X1000lb=5000lb` have below 10mpg.



Calculate θ_0 and θ_1 :

Initialize the required variables :

As per Gradient Descent for Linear Regression equation, we need α and the number of iterations to be set. Here for our example we will set the α to `0.1` and `num_of_iterations` to `1000`. Later we will find our whether the num_of_iterations is enough,more or less.

We will calculate the number of training set `m` using the `length()` function.

Afterwards, we will initialize θ to `[0,0]`.

Our `x` matrix has only 1 variable which is x_1, however as we have assumed x_0 = 1 in order to generalize the Gradient Descent equation, we need to add x_0 to the `x` matrix as a new column.
Refer the last line of the code fragment below, added another column to the matrix `x` using the `ones()` function. The `ones()` function will return a matrix

of `Length of training set X 1` (col X row) which will get added as a new column. The x now will be a 100 X 2 dimensional matrix.

```
alpha = 0.1;
num_of_iterations = 1000;

theta=zeros(2,1); %using the zeros() function.

x=[ones(m,1),x];
```

Here is the part of x matrix after we have added x_0.

```
1.0000   3.5040
1.0000   3.6930
1.0000   3.4360
1.0000   3.4330
1.0000   3.4490
1.0000   4.3410
1.0000   4.3540
.....
```

Iterate the Gradient Descent Function :

Our next task is to Calculate the θ and iterate 1000 times for convergence. So lets create a `for` loop, then calculate $h_\theta(x)$ by multiplying `x` and `theta` (Refer the equation above). `x` is `(100 X 2)` matrix and `theta` is `(2 X 1)` matrix. The resultant matrix would be a `(100 X 1 )` matrix.
Then we will do an element wise subtraction. The `h_of_x` would still be a `(100 X 1 )` matrix.

Note : The below code can also write using more loops, however matrix calculations are much easier and faster to compute.

```
for i=1:num_of_iterations

        h_of_x=(x*theta).-y;

end
```

Its time to calculate θ_j, since we have only 1 feature, we will calculate both θ_0 AND θ_1 without using loops/matrix multiplication.

The below calculations are easy, however notice we are using transpose of `h_of_x`. Here `h_of_x` and `x(:,1) or x(:,2)` both are `( 100 X 1 )` matrix. So we can't multiply them without transposing one of them. If you take transpose of `h_of_x` and then multiply with `x(:,1) or x(:,2)` then the resultant would be a scaler value (element wise multiplication then summation). This is also called products of `row vector` and `column vector`, which will always yield a scaler value. Here is the simplified version of the equation.

$$\begin{bmatrix} a & b \end{bmatrix} . \begin{bmatrix} c \\ d \end{bmatrix} = [a * c + b * d]$$

Here is the code to calculate θ.

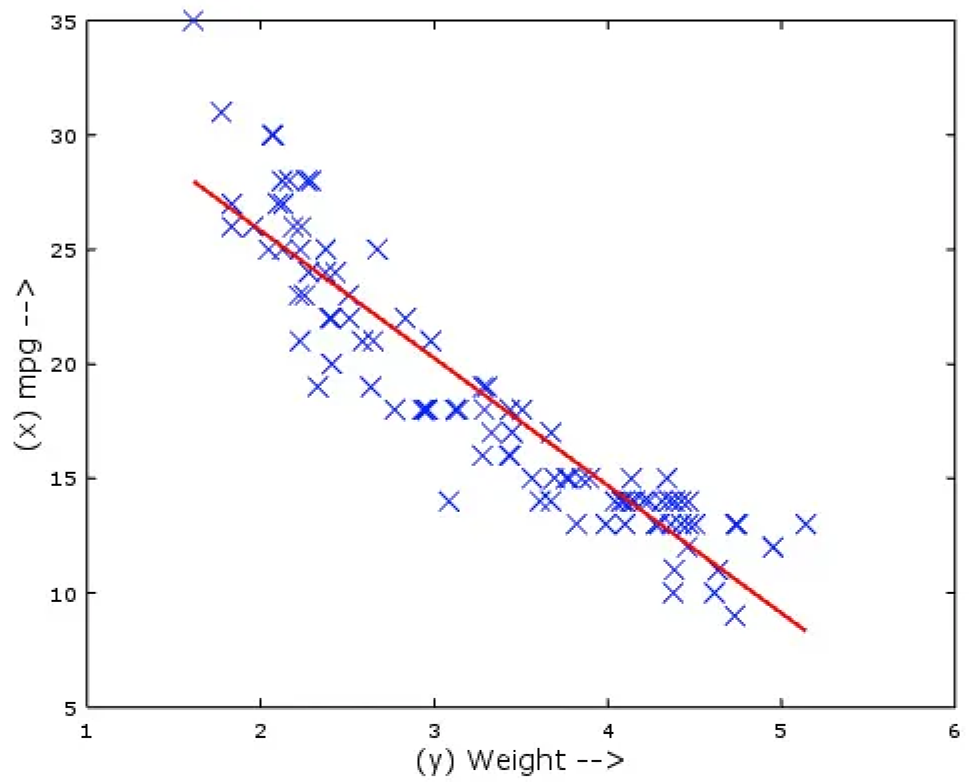
```
for i=1:num_of_iterations

  h_of_x=(x*theta).-y;

  theta(1)=theta(1)-(alpha/m)*h_of_x'*x(:,1);
  theta(2)=theta(2)-(alpha/m)*h_of_x'*x(:,2);

end
```

Plot the Hypothesis:

Let's first print the value of θ then the will plot the line in the previous scatter plot. We already had the x_1 (weight), so we will pass that to the plot function as the first argument. Remember to pass only the 2nd column from the `x` matrix. In order to calculate the value of `y`, we will multiply the `x (100 X 2)` matrix with the `theta (2 X 1)`, which will output `( 100 X 1 )` matrix (this was already discussed during calculation of `h_of_x`).

```
fprintf('θ0 = %f θ1 = %f \n', theta(1), theta(2));

hold on;

%First plot x, the for y calculate y.
%(100X2) * ( 2 X 1) =( 100 X 1 )
plot(x(:,2), x*theta, 'r-','linewidth',2);
legend('Training data', 'Linear regression');
hold off;
```

Once you run the code you can see the straight line.

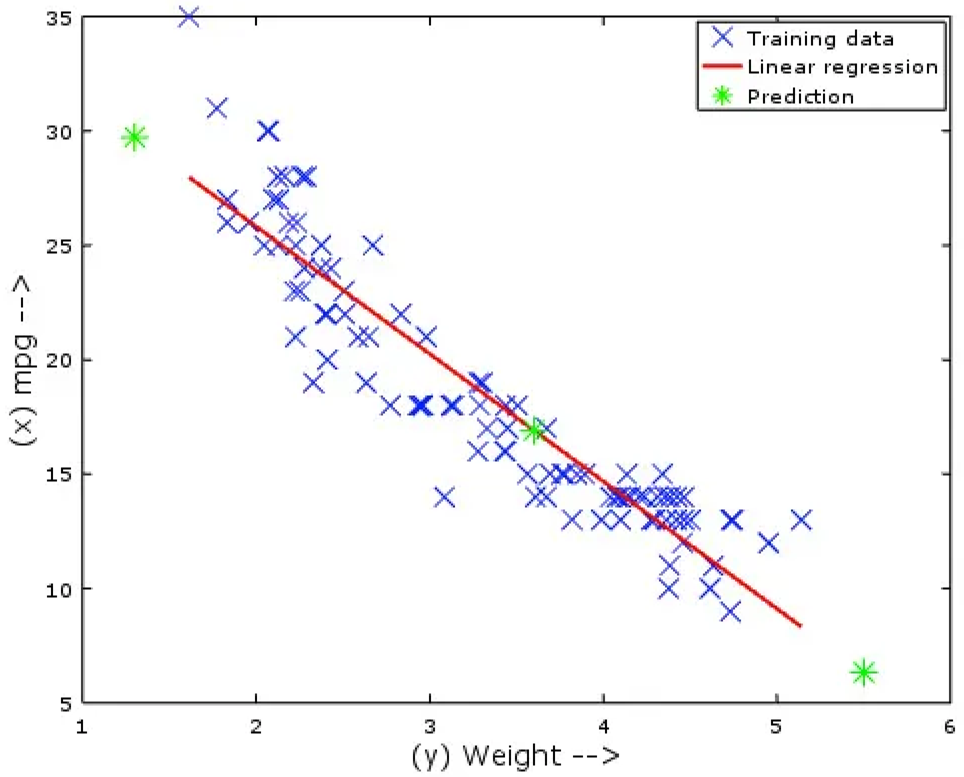


Prediction:

We can predict the mpg for any given weight. Lets try to get 3 predictions and also plot them in the same chart. We will predict the mpg for 1.3 X 1000lb=1300lb, 3600 and 5500lb respectively.
Here, the `x` is supplied as `i` in the `plot` function.The y needs to be calculated like previous code `(x * theta)`, however this time we will manually pass x as `[1, i]`, since $x_0 = 1$ and $x_1 = i$

```
predict=[1.3,3.6,5.5];

hold on;

for i=predict
        plot(i, [1, i]*theta, 'g*','MarkerSize',14);
end

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

hold off;
```

Here is the final output. The three predictions are displayed as green star.



Cost Function:

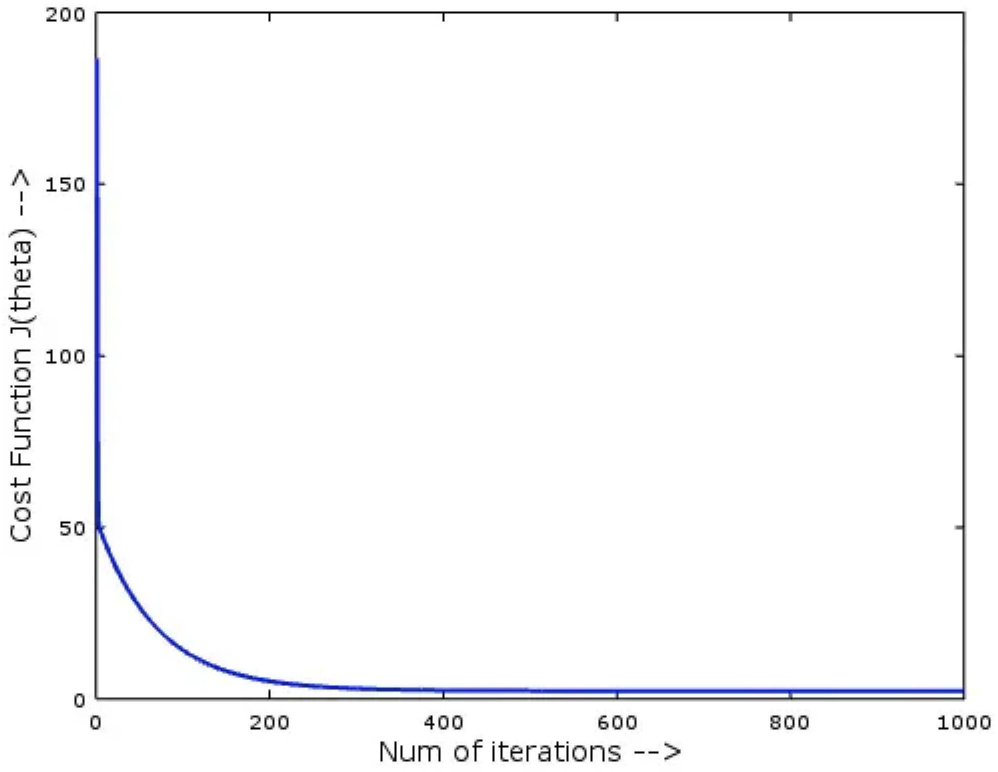
Another very important aspect is cost function. Initially we had set the iterations as 1000, however how to make sure 1000 is enough, more or less? also how we can find whether the gradient descent is converging at all?

The cost function would provide the answers for the above questions. We will store the cost function in an Array and plot that against the iterations. Then we can notice how the cost gradually converging to zero.

Here is the code.We have added the last line in the `for` loop to calculate the `cost`. Later you can plot it.

```
J=zeros(num_of_iterations,1);

for i=1:num_of_iterations

  h_of_x=(x*theta).-y;
  ....... skipping the code

  %compute J(θ) - Cost
  J(i)=1/(2*m)*sum(h_of_x.^2);

end
.....

figure;

plot([1:num_of_iterations],J,'linewidth',2);
xlabel('Num of iterations -->','fontsize',14);
ylabel('Cost Function J(theta) -->','fontsize',14);
```

From the plot below you can clearly see how $J(\theta_0, \theta_1)$ has decreased over iterations and converging towards zero. You can also see that the line kind of getting flat from 300 iteration onwards. This tells us that function is already converged.



Normal Equation :

We can also use the Normal Equation in order to calculate the θ. The Normal Equation is easy to understand/implement and faster for smaller number of features.However if you have 100s of features it tends to take more compute time than gradient descent. Again, for recap , here is the Normal Equation:

$\theta = (X^T X)^{-1} X^T y$

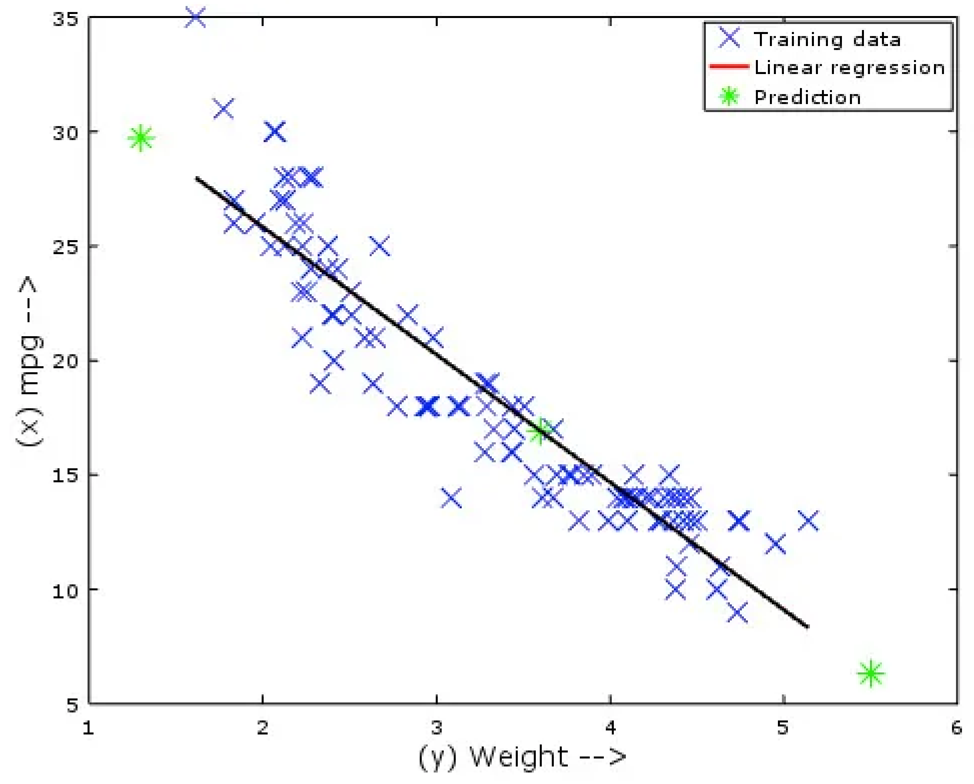
Calculate θ :

In Octave/MATLAB we can implement this in just one line. We will then plot in the same chart as Gradient Descent in order to compare both the results.

```
thetaNormal = (pinv(x'*x))*x'*y;

hold on;
plot(x(:,2), x*thetaNormal, 'k-','linewidth',1);
```

```
hold off;
```

Refer the chart below, the black line is from the Normal Equation, which is overlaying the line from Gradient Descent. This also confirms that both the equations are resulting the same output.



Full Octave Code:

```
clear ; close all; clc;

%Load Car Data
%weight,mpg - data.txt ( 100 rows )
car_data=load('data.txt');

x=car_data(:,1); % X would have 100 cols & 1 row
y=car_data(:,2); % y would have 100 cols & 1 row
```

```
m=length(y);

%Now lets plot the data in scatter plot
figure;
plot(x,y,'bx','MarkerSize',10);
xlabel('(y) Weight -->','fontsize',14);
ylabel('(x) mpg -->','fontsize',14);

fprintf('Press any key to continue ...');
pause;

%now we will calculate Gradient Descent
%add x0 to the feature matrix
x=[ones(m,1),x];

%initialize theta θ=[θ0,θ1]
theta=zeros(2,1);

num_of_iterations = 1000;
alpha = 0.1;

J=zeros(num_of_iterations,1);

for i=1:num_of_iterations

  h_of_x=(x*theta).-y;

  %h_of_x so that we would get a real number ( element
wise multiplication + sum )

  theta(1)=theta(1)-(alpha/m)*h_of_x'*x(:,1);
  theta(2)=theta(2)-(alpha/m)*h_of_x'*x(:,2);
  %compute J(θ) − Cost
  J(i)=1/(2*m)*sum(h_of_x.^2);

end

fprintf('θ0 = %f θ1 = %f \n', theta(1), theta(2));

hold on;

%First plot x, the for y calculate y.
%(100X2) * ( 2 X 1) =( 100 X 1 )
plot(x(:,2), x*theta, 'r-','linewidth',2);

hold off;

fprintf('Press any key to continue ...');
pause;

predict=[1.3,3.6,5.5];

hold on;

for i=predict

plot(i, [1, i]*theta, 'g*','MarkerSize',14);
legend('Training data', 'Linear regression','Prediction')

end

hold off;

fprintf('Press any key to continue ...');
pause;
```

```
thetaNormal = (pinv(x'*x))*x'*y;

hold on;

plot(x(:,2), x*thetaNormal, 'k-','linewidth',1);
hold off;

fprintf('Press any key to continue ...');
pause;

figure;

plot([1:num_of_iterations],J,'linewidth',2);
xlabel('Num of iterations -->','fontsize',14);
ylabel('Cost Function J(theta) -->','fontsize',14);
```

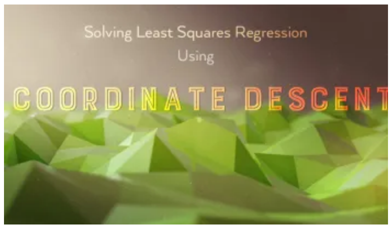
You can find the `data.txt` and also the above Octave code (main.m) in GitHub. Here is the link below.

Code

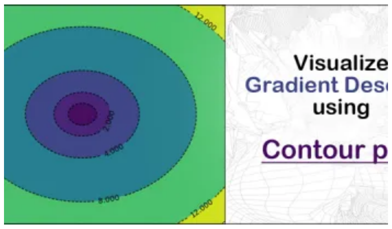
Conclusion:

We saw very simple implementation of Univariate Linear Regression using both the Gradient Descent and Normal Equation. In the next tutorial we will learn more on Multivariate Linear Regression.

Related



Introduction to Coordinate Descent using Least Squares Regression
In "Data Science"



How to visualize Gradient Descent using Contour plot in Python
In "Data Science"

Support Vector Machines for Beginners – Linear SVM
In "Data Science"

Filed Under: Machine Learning | Tagged With: Cost Function, Data Science, Gradient Descent, Linear Regression, Machine Learning, MATLAB, Normal Equation, Octave, Univariate

Subscribe to stay in loop

* indicates required

Email Address *

Subscribe

Leave a Reply

Logged in as Abhisek Jana. Edit your profile. Log out? Required fields are marked *

Comment *

Post Comment