Cost function in ML

Cost Function

The cost function helps us to figure out the best possible values for a_0 and a_1 which would provide the best fit line for the data points. Since we want the best values for a_0 and a_1, we convert this search problem into a minimization problem where we would like to minimize the error between the predicted value and the actual value.

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

$$J = \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

Minimization and Cost Function

We choose the above function to minimize.

The difference between the predicted values and ground truth measures the error difference.

We square the error difference and sum over all data points and divide that value by the total number of data points.

This provides the average squared error over all the data points.

Therefore, this cost function is also known as the **Mean Squared Error (MSE) function**.

Now, using this MSE function we are going to change the values of a_0 and a_1 such that the MSE value settles at the minima.

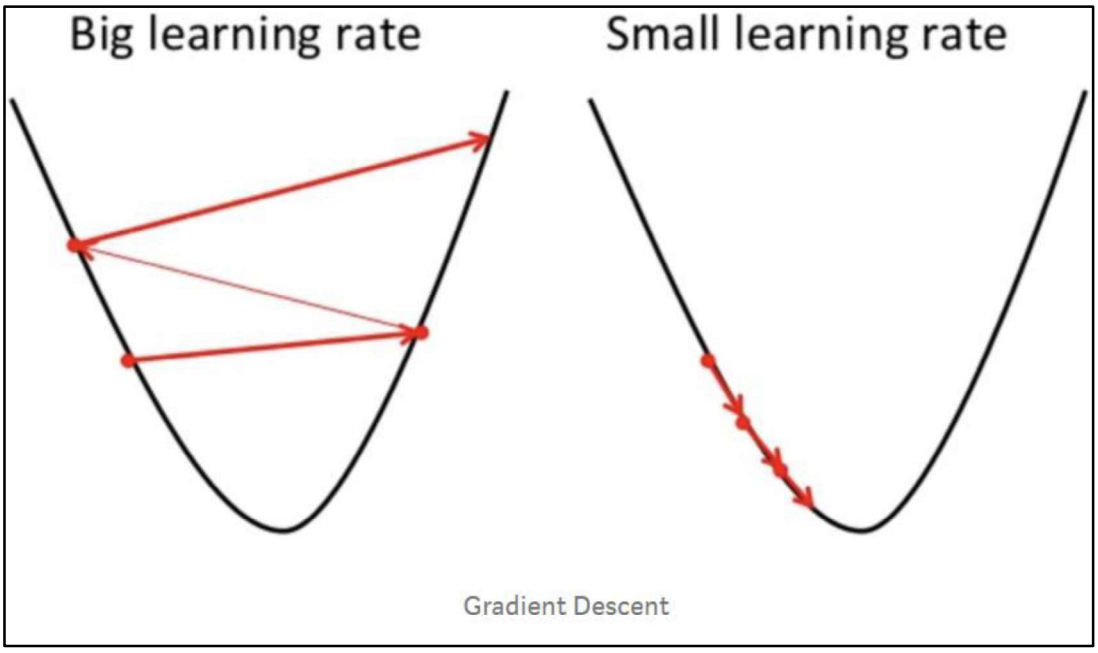
Gradient Descent

The next important concept needed to understand linear regression is gradient descent.

Gradient Descent is a method of updating a_0 and a_1 to reduce the cost function (MSE).

The idea is that we start with some values for a_0 and a_1 and then we change these values iteratively to reduce the cost.

Gradient descent helps us on how to change the values.



To draw an analogy, imagine a pit in the shape of U and you are standing at the topmost point in the pit and your objective is to reach the bottom of the pit.

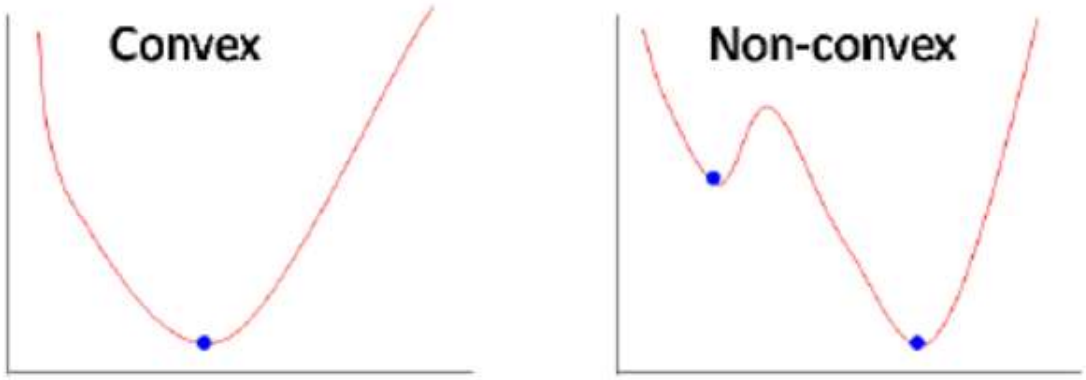
There is a catch, you can only take a discrete number of steps to reach the bottom.

If you decide to take one step at a time you would eventually reach the bottom of the pit but this would take a longer time.

If you choose to take longer steps each time, you would reach sooner but, there is a chance that you could overshoot the bottom of the pit and not exactly at the bottom.

In the gradient descent algorithm, the number of steps you take is the learning rate.

This decides on how fast the algorithm converges to the minima.



Convex vs Non-convex function

Sometimes the cost function can be a non-convex function where you could settle at a local minima but for linear regression, it is always a convex function.

You may be wondering how to use gradient descent to update a_0 and a_1.

To update a_0 and a_1, we take gradients from the cost function.

To find these gradients, we take partial derivatives with respect to a_0 and a_1.

Now, to understand how the partial derivatives are found below you would require some calculus but if you don't, it is alright.

You can take it as it is.

The partial derivates are the gradients and they are used to update the values of a_0 and a_1.

Alpha is the learning rate which is a hyperparameter that you must specify.

A smaller learning rate could get you closer to the minima but takes more time to reach the minima, a larger learning rate converges sooner but there is a chance that you could overshoot the minima.

Training Set

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

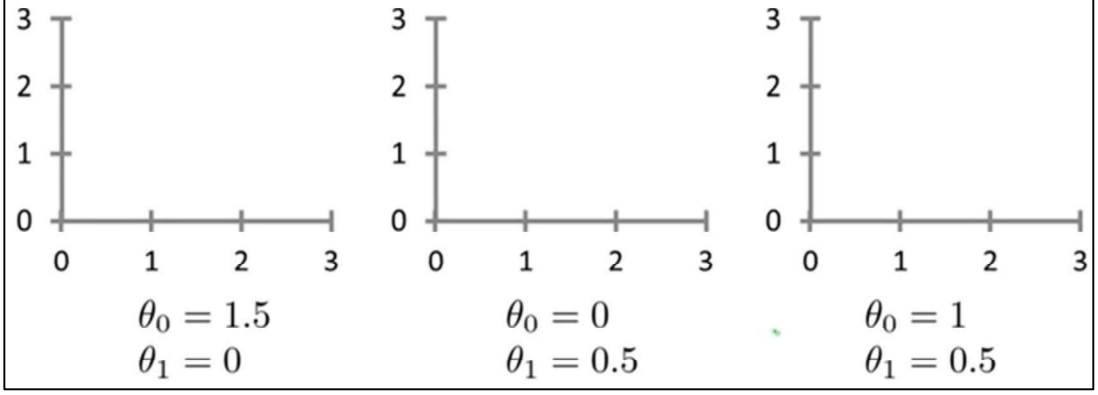
M = 47

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

θ_i's: Parameters

How to choose θ_i's ?

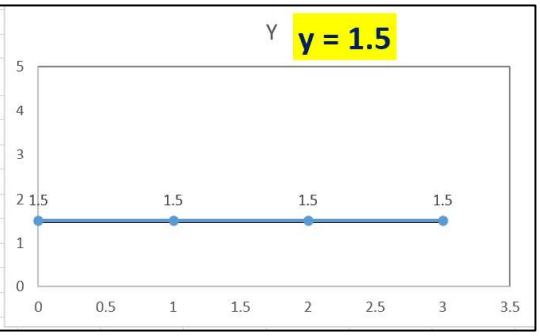
$h_\theta(x) = \theta_0 + \theta_1 x$



Y=MX + C

CASE1: M= 0; C = 1.5

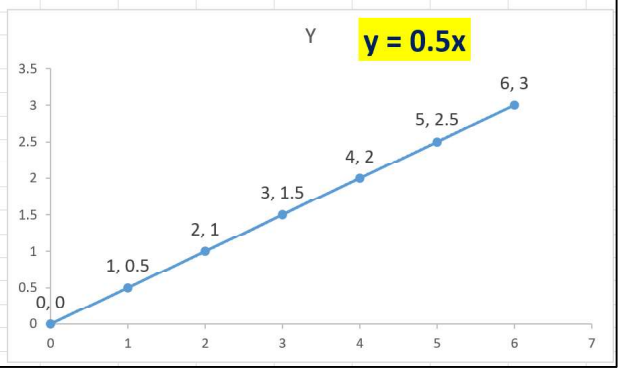
X	Y
0	1.5
1	1.5
2	1.5
3	1.5



Y=MX + C

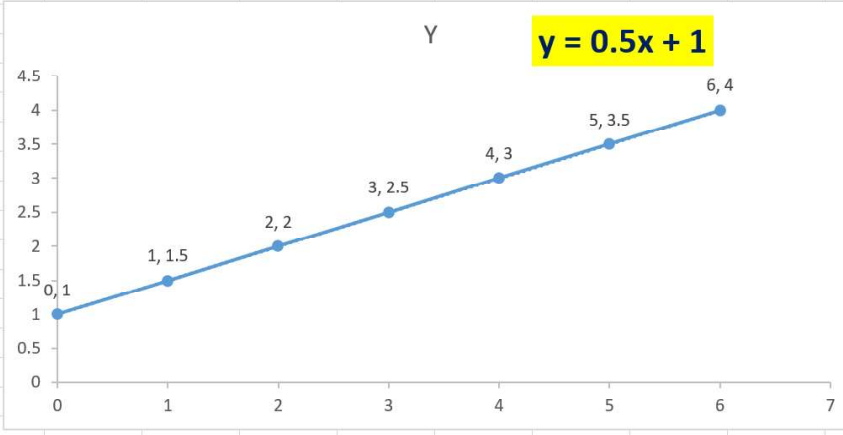
CASE1: M= 0.5; C = 0

M	C	X	Y
0.5	0	0	0
		1	0.5
		2	1
		3	1.5
		4	2
		5	2.5
		6	3



Y=MX + C

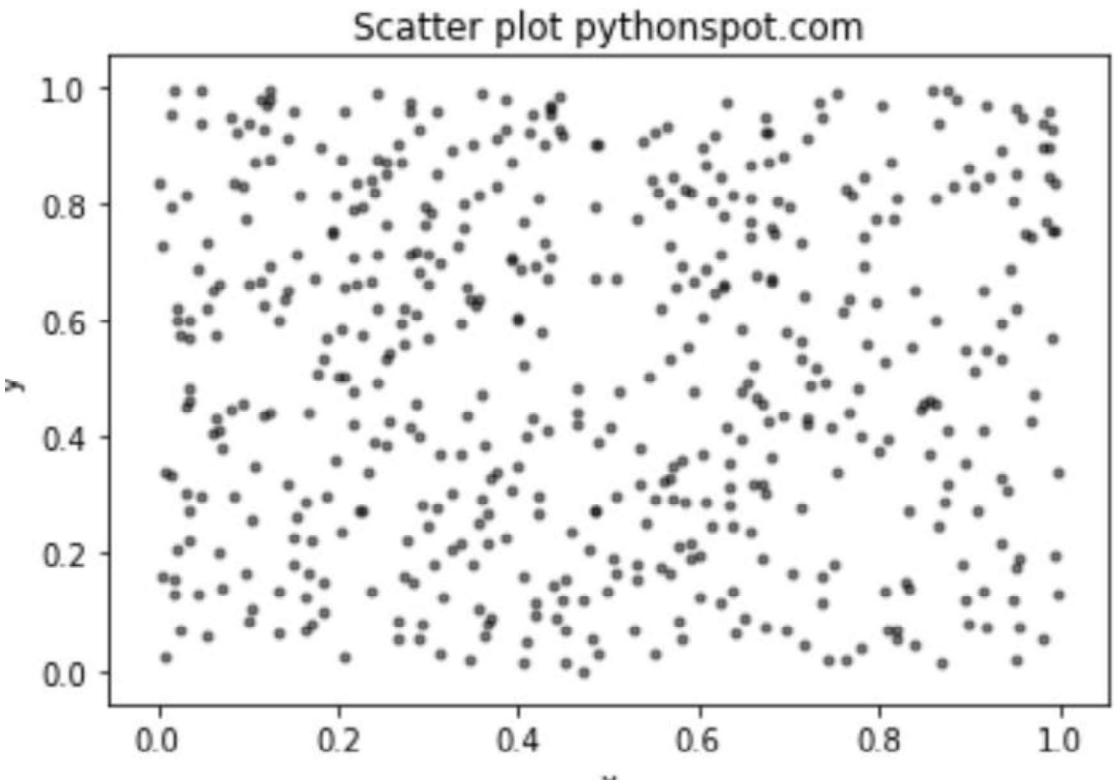
M	C	X	Y
0.5	1	0	1
		1	1.5
		2	2
		3	2.5
		4	3
		5	3.5
		6	4
		8	5




```python
import numpy as np
import matplotlib.pyplot as plt

# Create data
N = 500
x = np.random.rand(N)
y = np.random.rand(N)
colors = (0,0,0)
area = np.pi*3

# Plot
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.title('Scatter plot pythonspot.com')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```




```
#import necessary modules
import csv
with open('D:\MLIIISEMNOTES-09AUG2020\MYPYTHONPRGS-26JULY2020\TEMP1.csv','rt')as f:
  data = csv.reader(f)
  for row in data:
        print(row)
```

```
['X', 'Y ']
['0', '1.5']
['1', '2']
['2', '2.5']
['3', '3']
['4', '3.5']
['5', '4']
['6', '4.5']
['8', '5']
```

```
#import necessary modules
import pandas
result = pandas.read_csv('D:\MLIIISEMNOTES-09AUG2020\MYPYTHONPRGS-26JULY2020\data.csv')
print(result)
```

```
  Programming language        Designed by  Appeared Extension
0               Python   Guido van Rossum      1991       .py
1                 Java      James Gosling      1995     .java
2                  C++  Bjarne Stroustrup      1983      .cpp
```

```
#import necessary modules
import pandas
result = pandas.read_csv('D:\MLIIISEMNOTES-09AUG2020\MYPYTHONPRGS-26JULY2020\TEMP1.csv')
print(result)
```

```
   X   Y
0  0  1.5
1  1  2.0
2  2  2.5
3  3  3.0
4  4  3.5
5  5  4.0
6  6  4.5
7  8  5.0
```

Training Set

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

m = 47

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

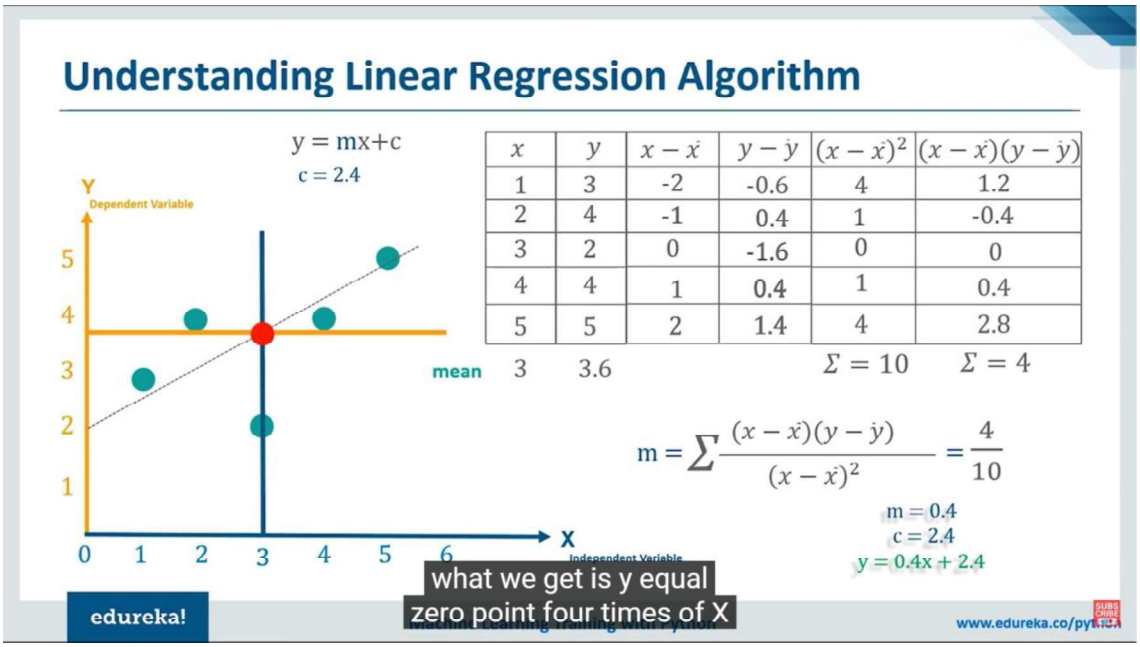
θ_i's: Parameters

How to choose θ_i's ?

Andrew Ng

Cost Function

```
function J = computeCostMulti(X, y, theta)
    m = length(y); % number of training examples
    J = 0;
    predictions =  X*theta;
    sqerrors = (predictions - y).^2;
    J = 1/(2*m)* sum(sqerrors);
end
```



x	y	$x-\bar{x}$	$y-\bar{y}$	$(x-\bar{x})^2$	$(x-\bar{x})(y-\bar{y})$
1	3	-2	-0.6	4	1.2
2	4	-1	0.4	1	-0.4
3	2	0	-1.6	0	0
4	4	1	0.4	1	0.4
5	5	2	1.4	4	2.8

$$m = 0.4$$
$$c = 2.4$$
$$y = 0.4x + 2.4$$

For given $m = 0.4$ & $c = 2.4$, lets predict values for y for $x = \{1,2,3,4,5\}$

$$y = 0.4 \times 1 + 2.4 = 2.8$$
$$y = 0.4 \times 2 + 2.4 = 3.2$$
$$y = 0.4 \times 3 + 2.4 = 3.6$$
$$y = 0.4 \times 4 + 2.4 = 4.0$$
$$y = 0.4 \times 5 + 2.4 = 4.4$$