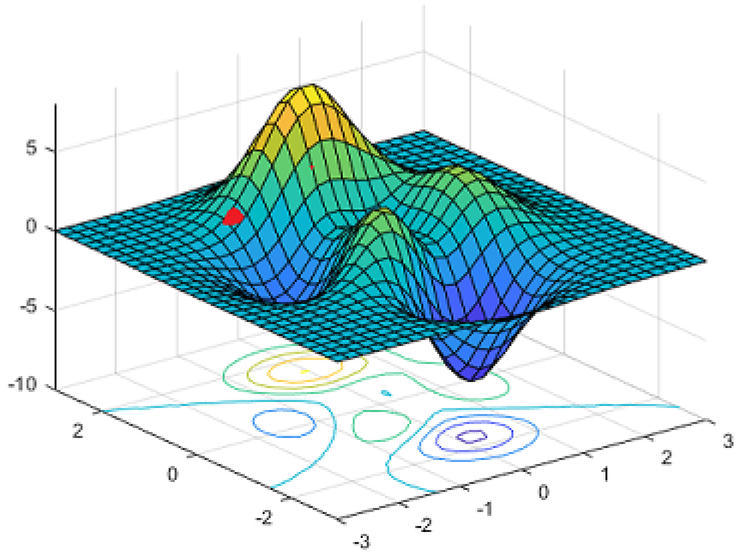
Gradient Descent



Outlines

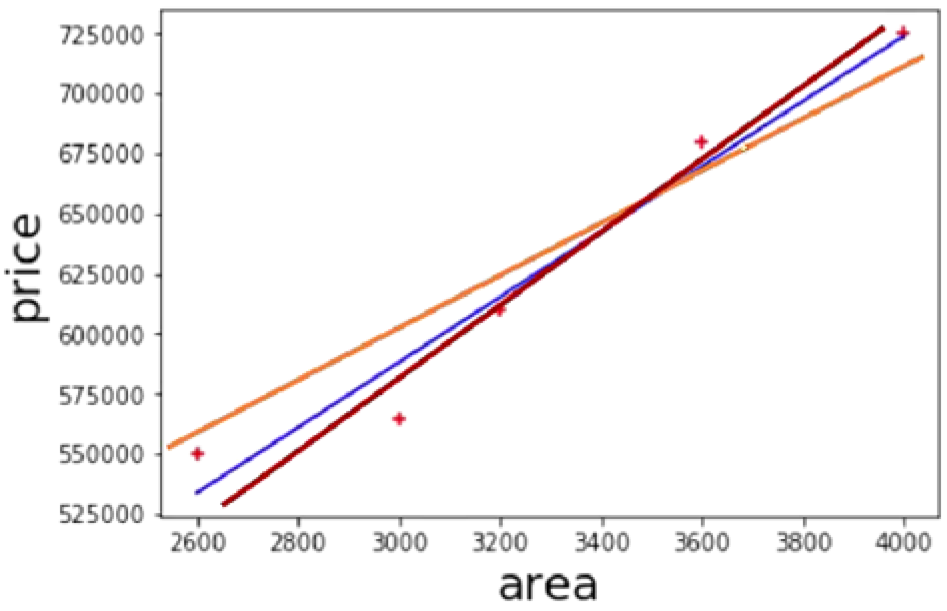
- Problem
- What is Gradient Descent?
- How Gradian Descent Works?
- Mean Square Error
- Python code for implementing Gradient Descent

if x=[1,2,3,4,5] & y=2x+3 so y=[5,7,9,11,13] However, in machine learning problem we have obsevation or training dataset which is an input and output like: x & y using that you can drive an equation. Also known as prediction function as a result you can use y=2x+3 to predict the future value of x.

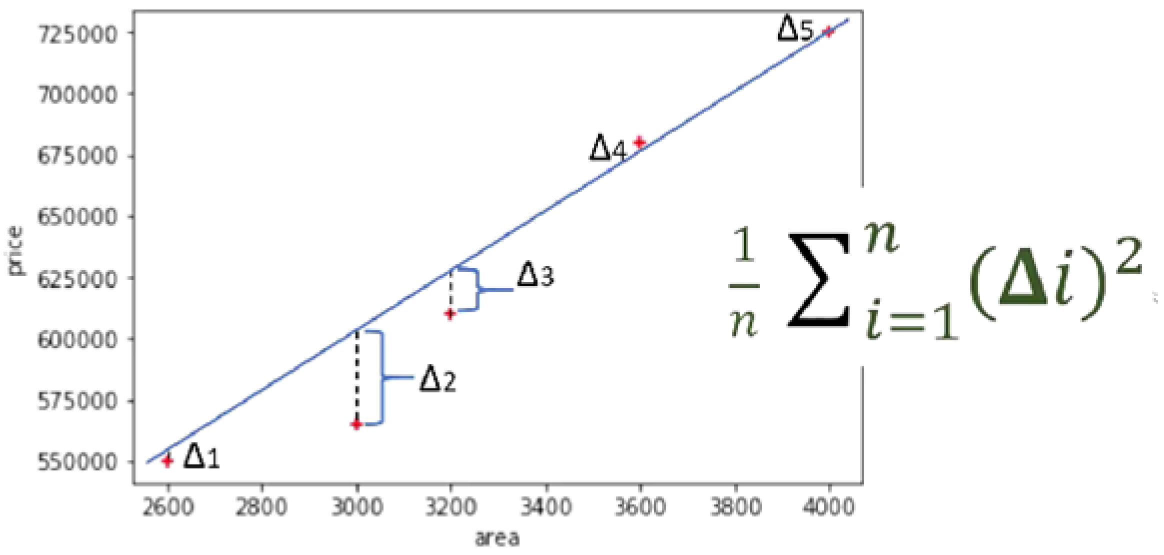
Problem:

How can you drive this equation price = 135.78*are + 180616.43 give this dataset are=[3600,3000,3200,3600,4000] price=[550k,565k,610k,680k,725k]? That equation is the best fit line going through all datapoints.

The problem here you might have so many lines. How do you know which line is the best fit line?



One way is you draw any random line then from your actual data point you calculate the error between that datapoint the datapoint predicted by your line. So call it the delta. You collect all this deltas and square them. The reason you want to square them is this deltas could also be negative. Then, you sum them up and divided by n. Here n is 5. It is the number of datapoints that is available. The result is called Mean Squared Error (MSE).



$$\frac{1}{n}\sum_{i=1}^{n}(\Delta i)^2$$

This MSE is also called a cost function.

Mean Squared Error

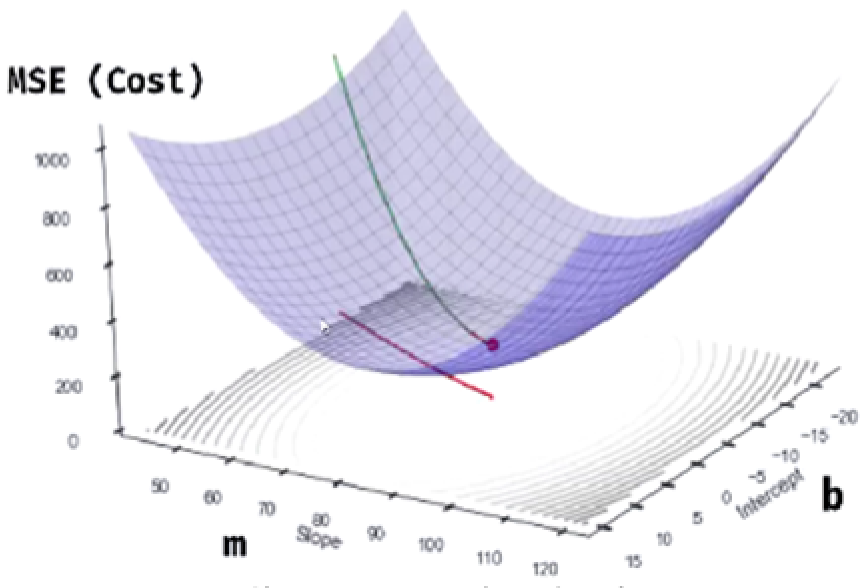
$$mse = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - (mx_i + b)\right)^2$$

Cost Function

What is Grasient Descent?

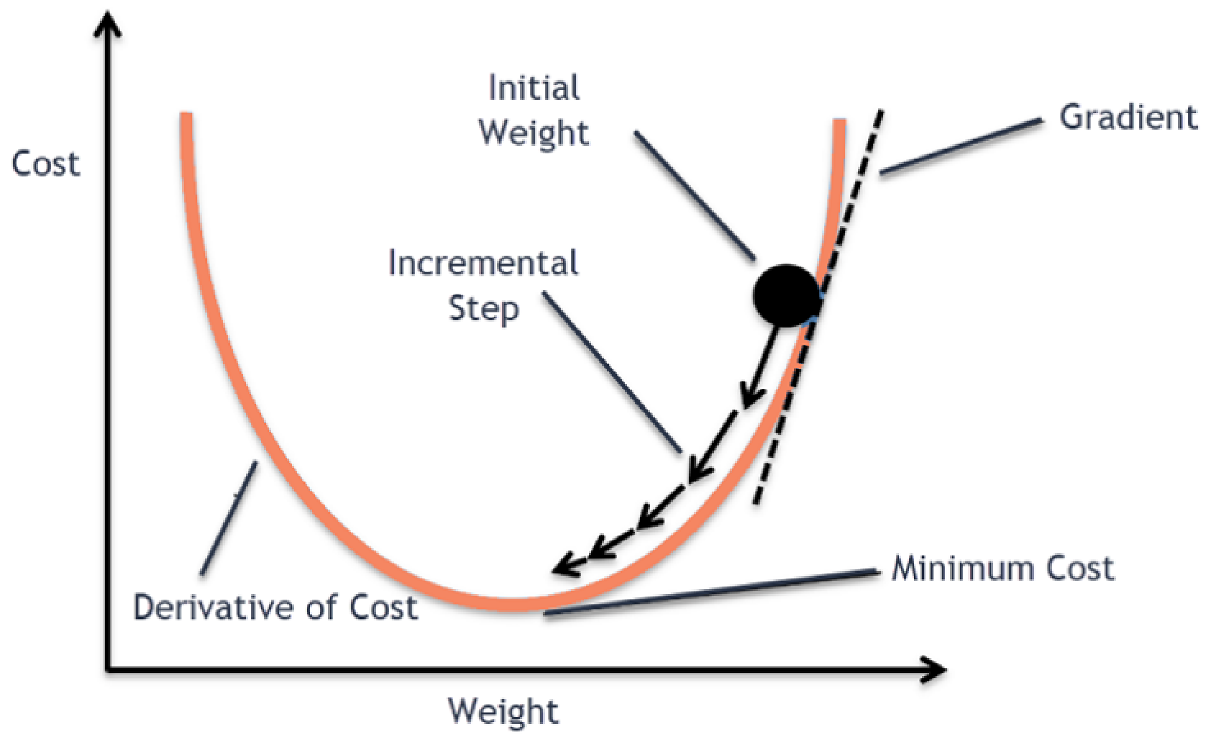
Gradient descent is an algorithm that finds best fit line for given training data set. Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent.

How Gradient Descent works?

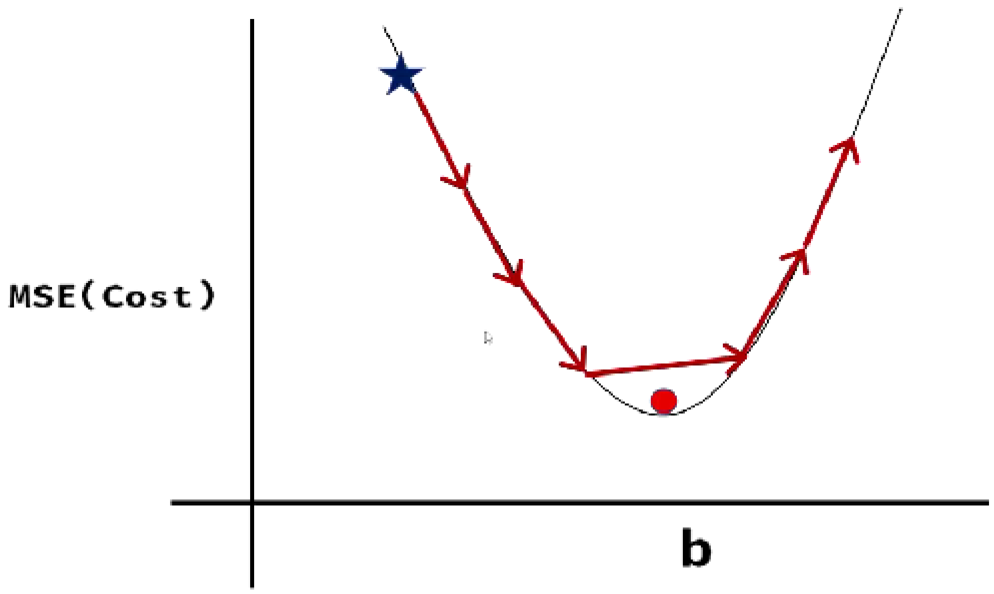


Reference: https://am207.github.io/2017/wiki/gradientdescent.html

Figure above plotted MSE versus different values of m & b. Start with some value of m & b (usually 0). From that point you calculate the cost for example the cost is 100. Then you reduce the valuse of m & b by some amount (Learning rate we talk about this later). By taking this step the error is reduced keep on taking this steps until you reack your minima. Here error is minimum. Once you reach that you have found your answer.

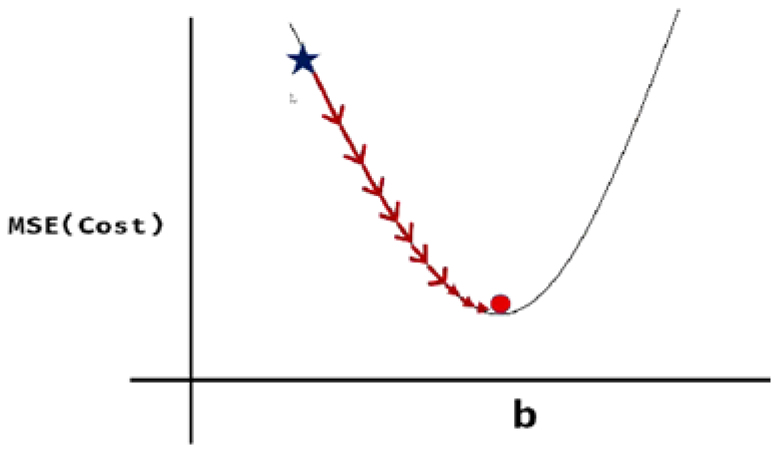


What if we take fix size steps for b?



By taking fix step I might missed the global minima.

The better approcah



This can work. Here in each step I am following the curvature of my chart and also as I reach near the red point the step size is reducing. How can I do that? at each point you need to calculate the slope. Once I have a slop I know which direction I need to go in. There is something called a learning rate which you can use in cunjunction with this slope here to take that step and reach the next point.

MSE and partial derivatives of m & b

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_{predicted})^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

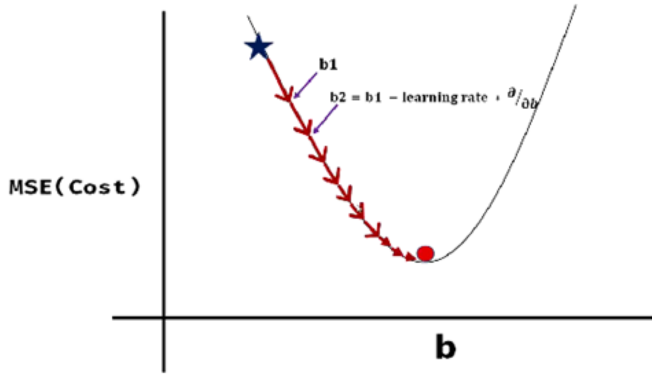
$$\frac{\partial MSE}{\partial m} = \frac{-2}{n}\sum_{i=1}^{n}x_i(y_i - (mx_i + b))$$

$$\frac{\partial MSE}{\partial b} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - (mx_i + b))$$

Once you have partial derivative you have your slope. Now you need to take step for step you use something called learning rate.

$$m = m - learning\_rate \times \frac{\partial MSE}{\partial m}$$

$$b = b - learning\_rate \times \frac{\partial MSE}{\partial b}$$



Python code for implementing Gradient Descent:

In [14]:
```
import numpy as np
```

In [15]:
```
def gradient_descent(x,y):
    m_curr=b_curr=0
    n=len(x)
    iterations = 100000
    learning_rate = 0.08
    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        cost = (1/n)*sum([val**2 for val in (y-y_predicted)])
        m_d = -(2/n)*sum(x*(y-y_predicted))
        b_d = (2/n)*sum(-(y-y_predicted))
        m_curr = m_curr - learning_rate * m_d
        b_curr = b_curr - learning_rate * b_d
        print("m{}, b{}, cost{} ,iteration {}".format(m_curr,b_curr,cost,i))
```

In [16]:
```
x= np.array([1,2,3,4,5])
y=np.array([5,7,9,11,13])
```

In [17]:
```
gradient_descent(x,y)
```

```
m1.6308855378034224, b1.0383405553279617, cost12.046787238456794 ,iteration 9
m3.2221235247119777, b1.5293810083298451, cost9.691269350698109 ,iteration 10
m1.7770832372205707, b1.1780607551353204, cost7.8084968312098315 ,iteration 11
m3.0439475772474127, b1.5765710804477953, cost6.302918117062937 ,iteration 12
m1.8898457226770244, b1.3032248704973899, cost5.098330841763168 ,iteration 13
m2.898169312926714, b1.6275829443328358, cost4.133961682056365 ,iteration 14
m1.9761515088959358, b1.4160484030347593, cost3.361340532576948 ,iteration 15
m2.7784216197824048, b1.6809279342791488, cost2.741808050753047 ,iteration 16
m2.0415541605113807, b1.5183370872989306, cost2.244528230107478 ,iteration 17
m2.6796170361078637, b1.735457156285639, cost1.8449036666988363 ,iteration 18
m2.090471617540917, b1.611567833948162, cost1.5233119201782324 ,iteration 19
m2.5976890103737853, b1.790290604096816, cost1.2640979056612756 ,iteration 20
m2.1264168621494517, b1.6969533824619085, cost1.0547704368105268 ,iteration 21
m2.529385561184701, b1.8447607474362664, cost0.8853615531285766 ,iteration 22
m2.1521818147302194, b1.7754939584778073, cost0.7479156468369821 ,iteration 23
m2.472104720735685, b1.8983676540508527, cost0.6360820885229722 ,iteration 24
m2.1699839382964696, b1.8480185634495874, cost0.5447903801652151 ,iteration 25
m2.423763296438881, b1.950743302915348, cost0.4699911136477278 ,iteration 26
m2.1815831093070837, b1.9152179921582295, cost0.4084494012702221 ,iteration 27
m2.3826922006906663, b2.0016232209455125, cost0.35758014655339476 ,iteration 2
```

Excersie:

1. For test scores in .csv file, run gradient descent algorithm to find out value of m, b and appropriate learning rate

2. On each iteration, compare previous cost with current cost. Stop when costs are similar (use math.isclose function with 1e-20 thresold)

3. Now using sklearn.linear_model find coefficient (i.e. m) and intercept (i.e b). Compare them with m, b generated by your gradient descent algorithm

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import math
```

In [3]:
```
df = pd.read_csv('D:/Data_Science/My Github/Machine-Learning-with-Python/3. Gradie
df.head()
```

Out[3]:

	name	math	cs
0	david	92	98
1	laura	56	68
2	sanjay	88	81
3	wei	70	80
4	jeff	80	83

In [4]:
```
r = LinearRegression()
r.fit(df[['math']],df.cs)
```

Out[4]:
```
LinearRegression()
```

In [8]:
```
def gradient_descent(x,y):
    m_curr=b_curr=0
    n=len(x)
    iterations = 100000
    learning_rate = 0.0001
    cost_previous = 0

    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        cost = (1/n)*sum([val**2 for val in (y-y_predicted)])
        m_d = -(2/n)*sum(x*(y-y_predicted))
        b_d = (2/n)*sum(-(y-y_predicted))
        m_curr = m_curr - learning_rate * m_d
        b_curr = b_curr - learning_rate * b_d
        if math.isclose(cost,cost_previous, rel_tol=1e-20):
            break
        cost_previous = cost

        print("m{}, b{}, cost{} ,iteration {}".format(m_curr,b_curr,cost,i))

    return m_curr, b_curr
```

In [10]:
```
x = np.array(df.math)
y=np.array(df.cs)
m,b = gradient_descent(x,y)
print("Using Gradient Descent Function: m={}, b={}".format(m,b))
print("Using sklearn: m={}, b={}".format(r.coef_,r.intercept_))
```

```
m1.0329800009641052, b0.834933388400767, cost31.67150668740983 ,iteration 4920
9
m1.032979825982256, b0.8349457889102064, cost31.67150514935082 ,iteration 4921
0
m1.0329796510024156, b0.8349581892773015, cost31.671503611327093 ,iteration 49
211
m1.0329794760245838, b0.8349705895020537, cost31.671502073338715 ,iteration 49
212
m1.0329793010487605, b0.8349829895844648, cost31.67150053538562 ,iteration 492
13
m1.0329791260749457, b0.8349953895245364, cost31.671498997467832 ,iteration 49
214
m1.0329789511031393, b0.8350077893222703, cost31.67149745958538 ,iteration 492
15
m1.0329787761333413, b0.8350201889776678, cost31.671495921738217 ,iteration 49
216
m1.0329786011655522, b0.8350325884907308, cost31.671494383926365 ,iteration 49
217
m1.0329784261997712, b0.8350449878614609, cost31.671492846149803 ,iteration 49
218
```

Algorithm	m	b
Gradient Descent	1.02	1.31
Sklearn	1.01	1.91

Date	Author
2021-08-22	**Ehsan Zia**