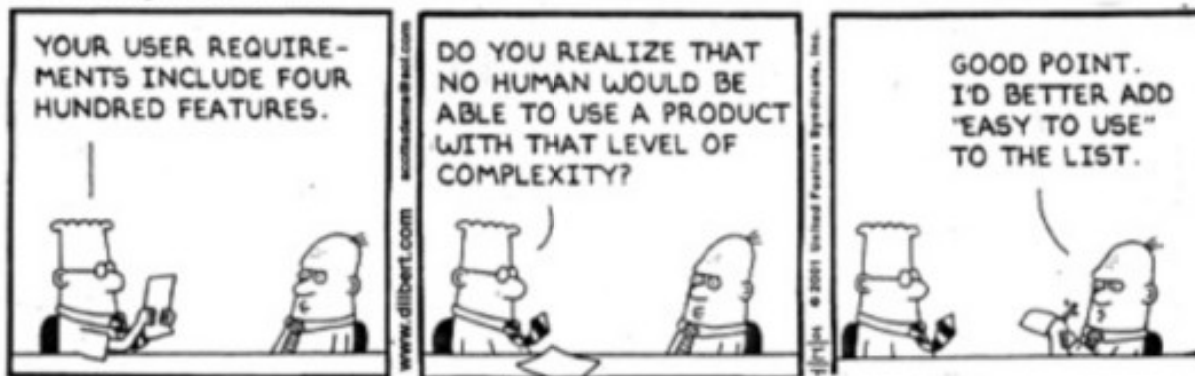


Gradient Descent

<https://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html>

By Jahnavi Mahanta.

DILBERT by Scott Adams



Source: dilbert.com

When I first started out learning about machine learning algorithms, it turned out to be quite a task to gain an intuition of what the algorithms are doing. Not just because it was difficult to understand all the mathematical theory and notations, but it was also plain boring. When I turned to online tutorials for answers, I could again only see equations or high level explanations without going through the detail in a majority of the cases.

It was then that one of my data science colleagues introduced me to the concept of working out an algorithm in an excel sheet. And that worked wonders for me. Any new algorithm, I try to learn it in an excel at a small scale and believe me, it does wonders to enhance your understanding and helps you fully appreciate the beauty of the algorithm.

Let me explain to you using an example.

Most of the data science algorithms are optimization problems and one of the most used algorithms to do the same is the Gradient Descent Algorithm.

Now, for a starter, the name itself Gradient Descent Algorithm may sound intimidating, well, hopefully after going through this post, that might change.

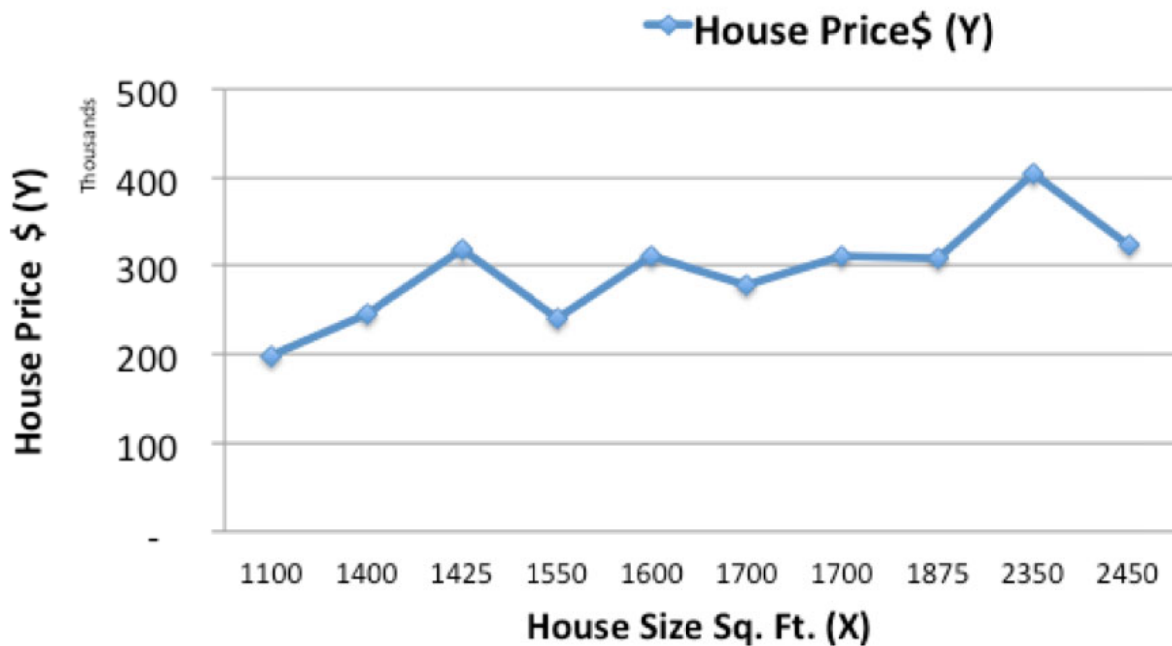
Lets take the example of predicting the price of a new house from housing data:

Now, given **historical housing data**, the task is to create a model that predicts the price of a new house given the house size.

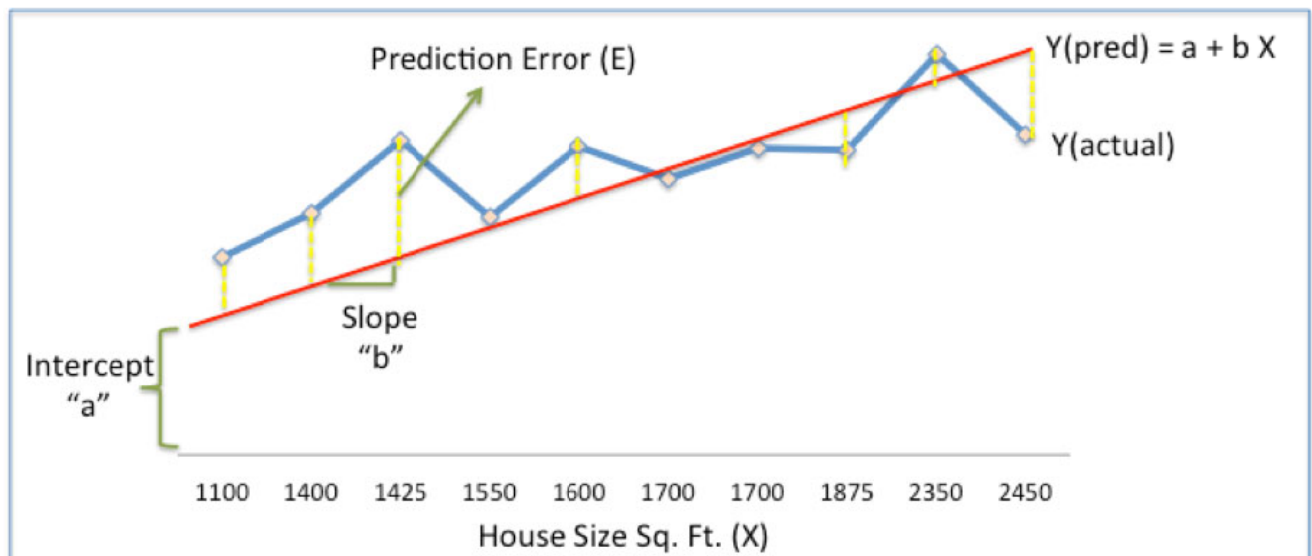
House Size sq.ft (X)	1400	1600	1700	1875	1100	1550	2350	2450	1425	1700
House Price\$ (Y)	245,000	312,000	279,000	308,000	199,000	219,000	405,000	324,000	319,000	255,000

The task – for a new house, given its size (X), what will its price (Y) be?

Lets start off by plotting the historical housing data:



Now, we will use a simple linear model, where we fit a line on the historical data, to predict the price of a new house (Y_{pred}) given its size (X)



In the above chart, the red line gives the predicted house price (Y_{pred}) given house size (X).

$$Y_{\text{pred}} = a + bX$$

The blue line gives the actual house prices from historical data (Y_{actual})

The difference between Y_{actual} and Y_{pred} (given by the yellow dashed lines) is the prediction error (E)

So, we need to find a line with optimal values of a, b (called weights) that best fits the historical data by reducing the prediction error and improving prediction accuracy.

So, our objective is to find optimal a, b that minimizes the error between actual and predicted values of house price ($1/2$ is for mathematical convenience since it helps in calculating gradients in calculus)

Sum of Squared Errors (SSE) = $\frac{1}{2} \text{Sum (Actual House Price – Predicted House Price)}^2$

$$= \frac{1}{2} \text{Sum}(Y - Y_{\text{pred}})^2$$

(Please note that there are other measures of Error. SSE is just one of them.)

This is where Gradient Descent comes into the picture. Gradient descent is an optimization algorithm that finds the optimal weights (a, b) that reduces prediction error.

Lets now go step by step to understand the **Gradient Descent algorithm**:

Step 1: Initialize the weights(a & b) with random values and calculate Error (SSE)

Step 2: Calculate the gradient i.e. change in SSE when the weights (a & b) are changed by a very small value from their original randomly initialized value. This helps us move the values of a & b in the direction in which SSE is minimized.

Step 3: Adjust the weights with the gradients to reach the optimal values where SSE is minimized

Step 4: Use the new weights for prediction and to calculate the new SSE

Step 5: Repeat steps 2 and 3 till further adjustments to weights doesn't significantly reduce the Error

We will now go through each of the steps in detail (I worked out the steps in excel, which I have pasted below). But before that, we have to standardize the data as it makes the optimization process faster.

HOUSING DATA	
House Size (X)	House Price (Y)
1,100	1,99,000
1,400	2,45,000
1,425	3,19,000
1,550	2,40,000
1,600	3,12,000
1,700	2,79,000
1,700	3,10,000
1,875	3,08,000
2,350	4,05,000
2,450	3,24,000

Min-Max Standardization	
X (X-Min/Max-min)	Y (Y-Min/Max-Min)
0.00	0.00
0.22	0.22
0.24	0.58
0.33	0.20
0.37	0.55
0.44	0.39
0.44	0.54
0.57	0.53
0.93	1.00
1.00	0.61

Step 1: To fit a line $Y_{pred} = a + bX$, start off with random values of a and b and calculate prediction error (SSE)

Calculate production error (SSE)					
a	b	X	Y	YP=a+bX	SSE=1/2(Y-YP)^2
0.45	0.75	0.00	0.00	0.45	0.101
		0.22	0.22	0.62	0.077
		0.24	0.58	0.63	0.001
		0.33	0.20	0.70	0.125
		0.37	0.55	0.73	0.016
		0.44	0.39	0.78	0.078
		0.44	0.54	0.78	0.030
		0.57	0.53	0.88	0.062
		0.93	1.00	1.14	0.010
		1.00	0.61	1.20	0.176
Total SSE					0.677

Step 2: Calculate the error gradient w.r.t the weights
 $\partial SSE / \partial a = -(Y - YP)$

$\partial SSE / \partial b = -(Y - YP)X$

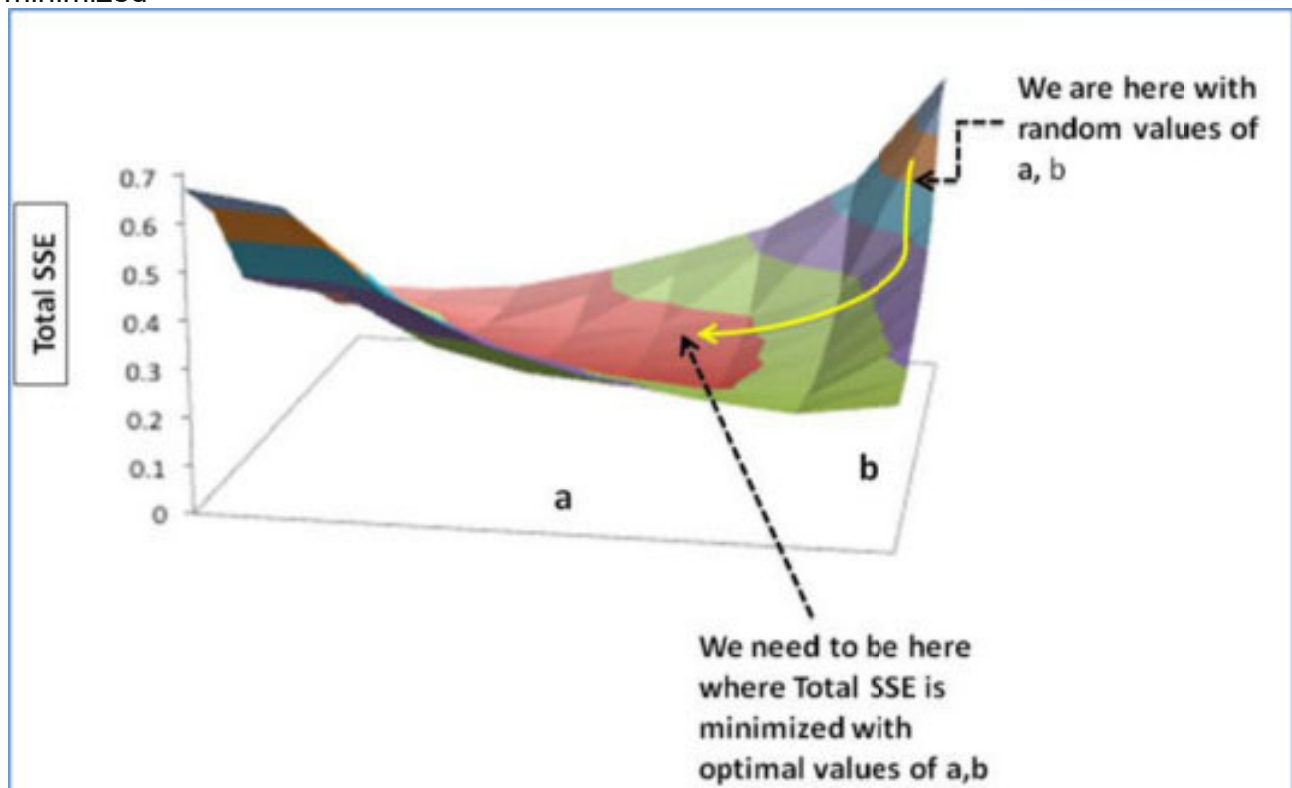
Here, $SSE = \frac{1}{2} (Y - YP)^2 = \frac{1}{2} (Y - (a + bX))^2$

You need to know a bit of calculus, but that's about it!!

$\partial \text{SSE} / \partial a$ and $\partial \text{SSE} / \partial b$ are the **gradients** and they give the direction of the movement of a, b w.r.t to SSE.

a	b	X	Y	YP=a+bX	SSE	$\partial \text{SSE} / \partial a$ = -(Y-YP)	$\partial \text{SSE} / \partial b$ = -(Y-YP)X
0.45	0.75	0.00	0.00	0.45	0.101	0.45	0.00
		0.22	0.22	0.62	0.077	0.39	0.09
		0.24	0.58	0.63	0.001	0.05	0.01
		0.33	0.20	0.70	0.125	0.50	0.17
		0.37	0.55	0.73	0.016	0.18	0.07
		0.44	0.39	0.78	0.078	0.39	0.18
		0.44	0.54	0.78	0.030	0.24	0.11
		0.57	0.53	0.88	0.062	0.35	0.20
		0.93	1.00	1.14	0.010	0.14	0.13
		1.00	0.61	1.20	0.176	0.59	0.59
Total SSE					0.677	Sum	3.300
							1.545

Step 3: Adjust the weights with the gradients to reach the optimal values where SSE is minimized



We need to update the random values of a, b so that we move in the direction of optimal a, b .

Update rules:

- $a - \partial \text{SSE} / \partial a$
- $b - \partial \text{SSE} / \partial b$

So, update rules:

1. New $a = a - r * \partial SSE / \partial a = 0.45 - 0.01 * 3.300 = 0.42$
 2. New $b = b - r * \partial SSE / \partial b = 0.75 - 0.01 * 1.545 = 0.73$
- here, r is the learning rate = 0.01, which is the pace of adjustment to the weights.

Step 4: Use new a and b for prediction and to calculate new Total SSE

a	b	X	Y	YP=a+bX	SSE	∂SSE/∂a	∂SSE/∂b
0.42	0.73	0.00	0.00	0.42	0.087	0.42	0.00
		0.22	0.22	0.58	0.064	0.36	0.08
		0.24	0.58	0.59	0.000	0.01	0.00
		0.33	0.20	0.66	0.107	0.46	0.15
		0.37	0.55	0.69	0.010	0.14	0.05
		0.44	0.39	0.74	0.063	0.36	0.16
		0.44	0.54	0.74	0.021	0.20	0.09
		0.57	0.53	0.84	0.048	0.31	0.18
		0.93	1.00	1.10	0.005	0.10	0.09
		1.00	0.61	1.15	0.148	0.54	0.54
Total SSE				0.553	Sum	2.900	1.350

You can see with the new prediction, the total SSE has gone down (0.677 to 0.553). That means prediction accuracy has improved.

Step 5: Repeat step 3 and 4 till the time further adjustments to a , b doesn't significantly reduces the error. At that time, we have arrived at the optimal a, b with the highest prediction accuracy.

This is the Gradient Descent Algorithm. This optimization algorithm and its variants form the core of many machine learning algorithms like Neural Networks and even Deep Learning.

Disclaimers:

Please note that this post is primarily for tutorial purposes, hence:

1. The data used is fictitious and data size is extremely small. Also to simplify the example, the data and model, is a one variable example.
2. This post is primarily meant to highlight how we can simplify our understanding of the math behind algorithms like Gradient descent by working them out in excel, hence there is no claim here that gradient descent gives better /worse results as compared to least square regression.
3. Since the data is very small, for tutorial purposes, the entire data is being used for training. However, while building actual predictive models, various data validation techniques are leveraged (Example – Train / Test split or N- Cross Validation)

Liked what you read? To learn other algorithms in a similar simplified manner, register for the 8 week Data science course on www.deeplearningtrack.com.

Bio: **Jahnavi** is a machine learning and deep learning enthusiast, having led multiple machine learning teams in American Express over the last 13 years. She is the co-founder of Deeplearningtrack, an online instructor led data science training platform—www.deeplearningtrack.com