Multi-Variate Linear Regression Using Gradient Descent

Subhash,Ishwar,Atul,Shubhangi

July 2020

1 Introduction

We have seen the case of approximating a straight line in 2D $Y = m * X + c$. In this case we have one input say y is used to predict the price of houses from the area of the houses, here Y is price of the house and X is the area of the house.

Here's is a question for you :) will price of the house be completely depended on area only.

No, it depends on many factors. The main factors include number of bedrooms, How old the house is, availability of schools and public transport and many more.
To include all these factors into account for predicting price of houses we use multiple linear regression.

Instead of taking only one feature (X) into account we take many features say $x_1, x_2, x_3, \ldots$ For these kind of cases we approximate y using linear combination of all these features and additional bias term. As like straight line for the case of single input and single output They approximate hyper planes $Y = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + \ldots$.

2 Linear Regression

This is quite similar to the single variable linear regression method which was discussed previously. This is implemented to problems which have more than one features.

In the case of Housing Prices we can consider the Price as a variable which is dependent upon a lot of independent variables. So, the trick here is to choose the best independent variable that actually determine the price.

So all these independent variable contribute to the price of house in a certain magnitude. But we don't know the values of the $w_0, w_1, w_2 \ldots$ for the equation. So we first randomize the values of $w_0, w_1, w_2 \ldots$ for the equation and then check the magnitude of error relating to different parameters we use. This is done by using a function called the Cost Function denoted by :

$$J = \frac{1}{2m}\sum_{i=0}^{2m}(y_i - \bar{y}_i)^2 \tag{1}$$

where y_i is the actual value and $\bar{y}_i$ is the predicted value. m is the total number of inputs. This Equation is called the *Mean Squared Error* equation. It determines the magnitude of error between the actual y_i value and predicted $\bar{y}_i$ value using the equation.

3 Gradient Descent

Look at the algorithm blindly, later we can dive into the concept
Gradient Descent algorithm:

Step 1: Initialize the weights $w_0, w_1, w_2, \ldots$ with random values and calculate loss or cost with those weights.(It will be high in the beginning, if zero you are lucky)

Step 2: Calculate the gradient i.e. change in cost funtion when the weights $w_0, w_1, w_2, \ldots$ are changed by a very small value from their original randomly initialized value. This helps us move the values of weights $w_0, w_1, w_2, \ldots$ in the direction in which cost is minimized.

Step 3: Adjust the weights $w_0, w_1, w_2, \ldots$ with the gradients to reach the optimal values where cost is minimized.

$$w_i := w_i - \frac{1}{m}\alpha\sum_{j=1}^{m}[y\_pred - y]x_i^j$$

Step 4: Use the new weights $w_0, w_1, w_2, \ldots$ for prediction and to calculate the new cost

Step 5: Repeat steps 2 and 3 till further adjustments to weights $w_0, w_1, w_2, \ldots$ doesn't significantly reduce the cost.

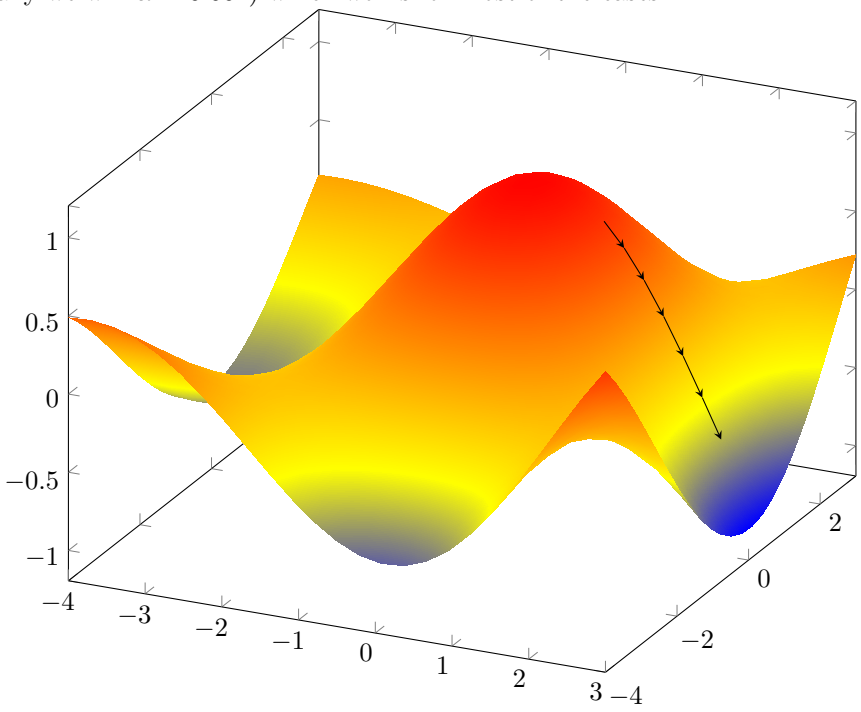
This is a algorithm which you see most of the time from simple linear regression to complex neural networks. The idea is simple, We have studied about the gradient of a multivariate function in our vector calculus chapter :) did we??

just to sink with you : mutivariate function are functions which take vector as inputs and produce either scalar (scalar valued function of a vector) or vector as output (vector valued function of a vector).The gradient gives us the direction of maximum ascent and negative of gradient gives the direction of maximum descent

Back to theory : Our loss function or cost is a scalar valued function of a vector. Here the vector input is the vector of all the weights or coefficients of the linear equation which we use to estimate our prediction(house price).

In the previous PDF you have seen the plot of cost function wrt weights w_0, w_1 for linear regression for single variable which is just a paraboloid.
To minimize the cost we need to travel in the direction opposite to the gradient but not very fast as the gradient depends on the position we are at.

The below shown is a figure of cost function wrt the weights not for a simple linear regression.
In this figure starting for a point (here the point near the maxima) we are moving in the direction of maximum descent but slowly with a small value of learning rate.The learning rate(usually denoted by α) controls the rate at which we move towards the minima. There are many ways to select the learning rate usually we will $\alpha = 0.001$) which works for most of the cases.



3.1 Normalization (Feature scaling)

We can speed up gradient descent by having each of our input values in roughly the same range. This is because w_i will descend quickly on small ranges and slowly on large ranges, and so willoscillate inefficiently down to the optimum when the variables are very uneven

The way to prevent this is to modify the ranges of our input variables so that they are all roughly the same. Ideally:

$$-1 \leq x_i \leq 1 \tag{2}$$

or

$$-0.5 \leq x_i \leq 0.5 \tag{3}$$

These aren't exact requirements; we are only trying to speed things up. The goal is to get all input variables into roughly one of these ranges, give or take a few.
Essentially, scaling the inputs (through mean normalization) gives the error surface a more spherical shape, where it wouldbe otherwise be a very high curvature ellipse.

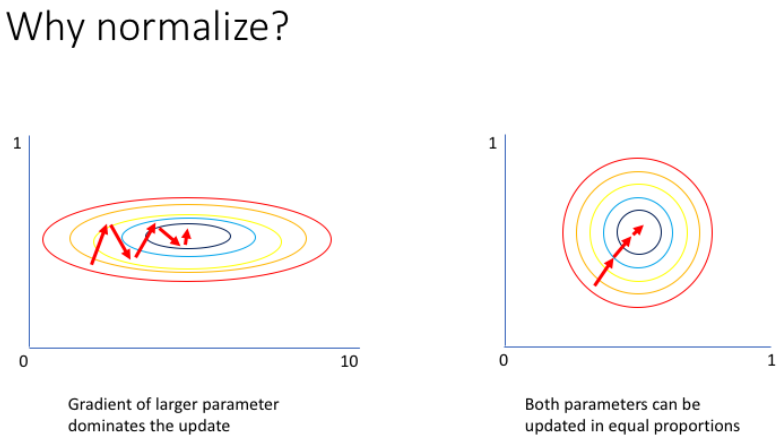


Figure 1: Contour Plots of Cost Function after Normalizing

Since gradient descent is curvature-ignorant, having an error surface with high curvature will mean that we take many steps which may not be in the optimal direction.When we scale the inputs, we reduce the curvature, which makes gradient descent ignore the curvature and work much better. When the error surface is spherical, the gradient points right at the minimum so learning is easy.
MEAN NORMALIZATION :

$$x_{normalized} = \frac{x - \bar{x}}{x_{max} - x_{min}} \tag{4}$$

4 Housing Prices Example

The price of House depending on factors such as Area, number of Bedrooms and nearest distance from station

Index	Price	Area	No of Bedrooms	Nearest Distance from Station
1	100000	1000	2	10
2	150000	2000	2	15
3	200000	2000	3	7
4	350000	2500	4	5

So the equation currently right now is:

$$\bar{y_i} = w_0 + w_1 * x_1 + w_2 * x_2 + w_3 * x_3 \tag{5}$$

where w_i are the coefficients and x_i denotes the independent variables

Lets initialize all the coefficients as 0 first,

$w^0 = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$

Then we can normalize our input and output to prevent divergence in *Gradient Descent*.
So we can normalize y_i as $\frac{y_i}{100000.0}$
and all the Area x_2 as $\frac{x_2}{1000.0}$

4.1 Iterating

The learning rate used : 0.01
No. of Iterations : 100
The Cost Function $J = 2.4375$
and the coefficient vector change to :

$w^1 = \begin{pmatrix} 0.02 & 0.041875 & 0.0625 & 0.16 \end{pmatrix}$

after 100 iterations the Cost function $J = 0.03163602$ and the coefficient vector becomes:

$w^{100} = \begin{pmatrix} 0.10242782 & 0.38100888 & 0.60354985 & -0.05008762 \end{pmatrix}$

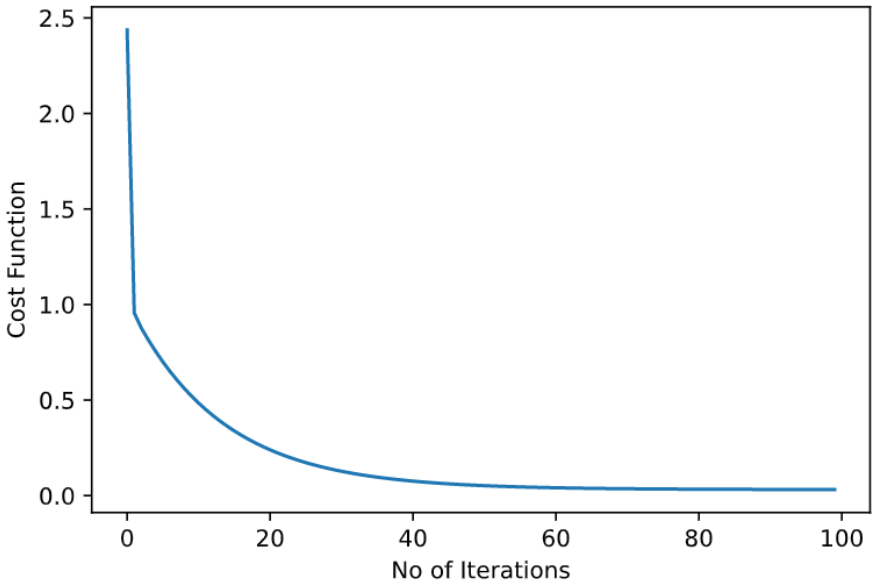


Figure 2: The Cost Function for 100 Iterations

So we can see that the cost functions decreases with each iteration and gives us a more accurate linear function for computing the price of house
The Linear function is now :

$$\bar{y_i} = 0.10242782 + 0.38100888 * x_1 + 0.60354985 * x_2 - 0.05008762 * x_3 \quad (6)$$

4.2 Impact of learning Rate

The learning rate controls how quickly the model is adapted to the problem. Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epochs.
A learning rate that is too large can cause the model to converge too quickly to a suboptimal solution, whereas a learning rate that is too small can cause the process to get stuck.

Using learning rate 0.1 for the above example :

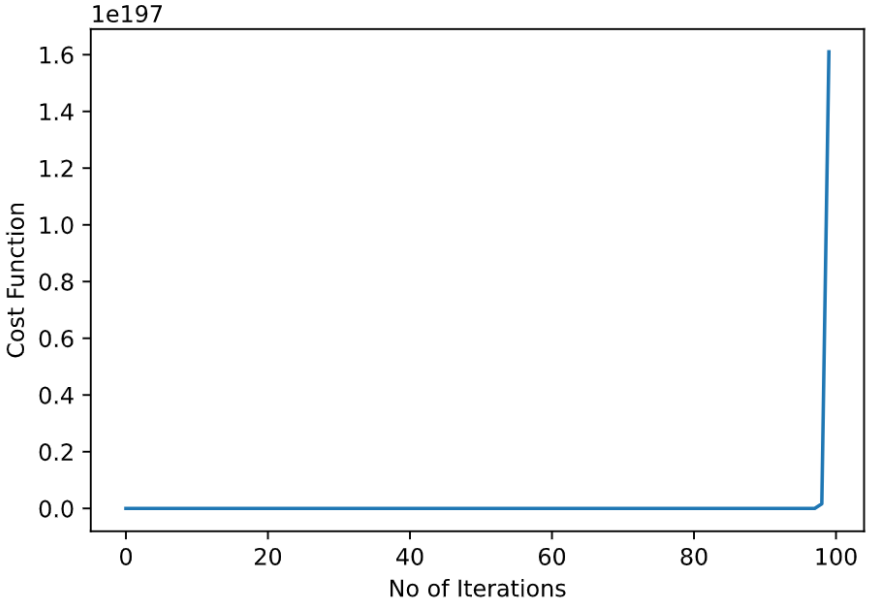


Figure 3: The Cost Function for 100 Iterations with 0.1 Learning rate

We can see that the value of the cost function diverges to $1.60984054 * 10^{197}$