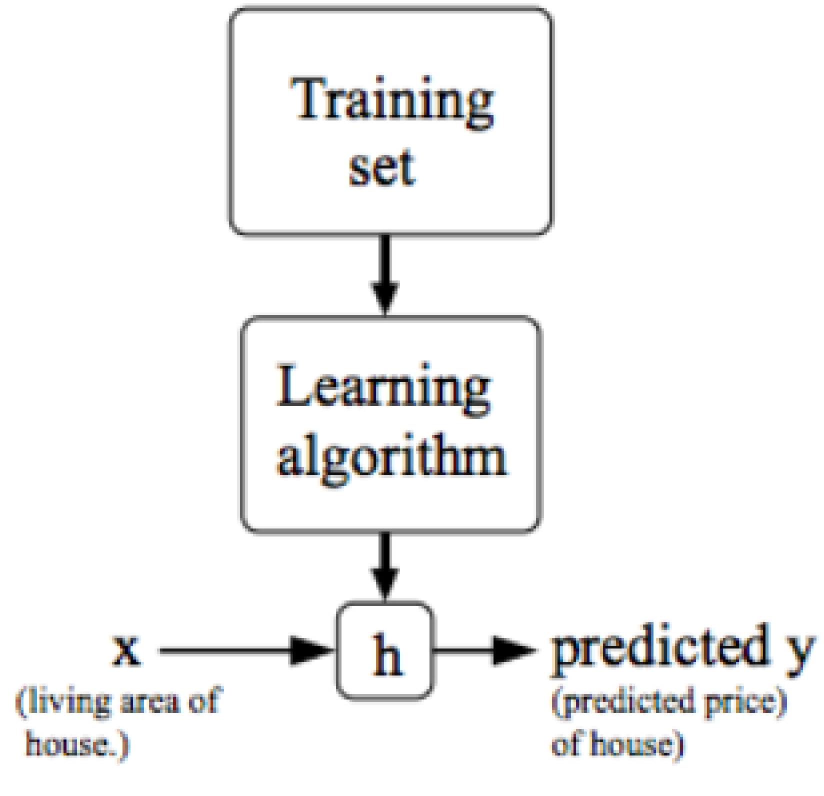
Linear Regression and Gradient Descent - Scribe

SMAI

25 January 2021

1 Introduction

Supervised learning , our goal is given a training set, to learn a function h : X $\rightarrow$ Y so that h(x) is a "good" predictor for the corresponding value of y. This function h is called a hypothesis. Seen pictorially, the process is therefore like this:



When the target variable that we're trying to predict is continuous, such as in our housing example, we call the learning problem a **regression** problem.

2 Linear Regression

Idea behind linear regression is that we have some training data(supervised) and we have to fit this training data in such a way that on any test value we are able to predict our result correct with great accuracy.So we try to fit a curve which pass through the training points(may not be perfectly) and with the help of our fitting we try to guess our output on other data(apart from training data).

2.1 Cost Function

We can measure the accuracy of our hypothesis function by using a **cost** function. This takes an average difference of all the results of the hypothesis with inputs from x's and the actual output y's.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right)^2$$

In simple words it is mean of squares of difference between the predicted value and actual value.Our goal is to minimize this cost function in order to make our prediction true to a great extent.

Now , in order to have cost function to be minimum what should be our parameter θ ?

In order to calculate this we use method of differentiation on our cost function and making that differentiation as 0 , will give us our θ parameter in terms of our training data(both X and Y) often known as "Normal Form Solution of Linear Regression".

2.2 Normal Form Solution

Derivation:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \left(\theta^T \left(x_i \right) - \left(y_i \right) \right)^2$$

$$= \frac{1}{2} \left\{ (\theta^T x_1 - y_1)^2 + (\theta^T x_2 - y_2)^2 + (\theta^T x_3 - y_3)^2 + ... \right\}$$

{ Using, Transpose Property: $\theta^T x = x^T \theta$ }

$$= \frac{1}{2} \left\{ ((x_1^T)\theta - y_1)^2 + ((x_2^T)\theta - y_2)^2 + ((x_3^T)\theta - y_3)^2 + ... \right\}$$

{ **The above expression can be represented in a matrix format as:** }

$$= \frac{1}{2}\begin{bmatrix}((x_1^T)\theta - y_1) & ((x_2^T)\theta - y_2) & .... & ((x_m^T)\theta - y_m)\end{bmatrix}_{1\times m}\begin{bmatrix}((x_1^T)\theta - y_1)\\((x_2^T)\theta - y_2)\\....\\((x_m^T)\theta - y_m)\end{bmatrix}_{m\times 1}$$

Let, A = $\{x_1, x_2, ..., x_m\}$ **and Y =** $\{y_1, y_2, ..., y_m\}$

$$J(\theta) = \frac{1}{2}(A\theta - Y)^T(A\theta - Y)$$

where, $A = \begin{bmatrix}... & x_1^T & ...\\ \\ ... & x_2^T & ...\\ ... & ... & ...\\ ... & ... & ...\\ .. & x_m^T & ..\end{bmatrix}_{m\times d}$, $\theta = \begin{bmatrix}\theta_0\\ \\ \theta_1\\ ...\\ ...\\ \theta_{d-1}\end{bmatrix}_{d\times 1}$, $Y = \begin{bmatrix}y_1\\ \\ y_2\\ ...\\ ...\\ y_m\end{bmatrix}_{m\times 1}$

Now,

$$\frac{\partial J}{\partial \theta} = \frac{1}{2}\frac{\partial}{\partial \theta}\Bigg[(A\theta - Y)^T(A\theta - Y)\Bigg]$$

$$= \frac{1}{2}\frac{\partial}{\partial \theta}\Bigg[(A\theta)^T A\theta - (A\theta)^T Y - y^T(A\theta) + Y^T Y\Bigg]$$

Let, $P = A\theta, Q = Y$

$$= \frac{1}{2}\frac{\partial}{\partial \theta}\Bigg[(A\theta)^T A\theta - 2(A\theta)^T Y + Y^T Y\Bigg]$$

$$= \frac{1}{2}\Bigg[2A^T A\theta - 2A^T Y\Bigg]$$

$$\frac{\partial J}{\partial \theta} = A^T A\theta - A^T Y$$

$If, \frac{\partial J}{\partial \theta} = 0\ then,$

$$A^T A\theta - A^T Y = 0$$

$$A^T A\theta = A^T Y$$

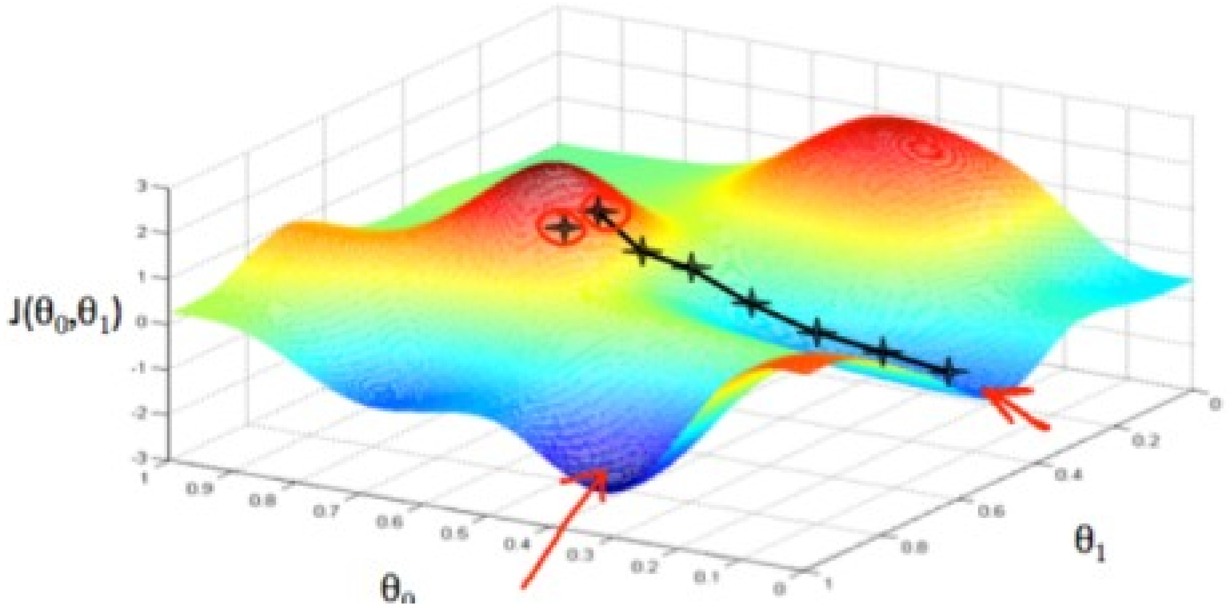
Therefore, the final normal form solution for solving linear regression is:

$$\theta = (A^T A)^{-1} A^T Y$$

3 Gradient Descent

Gradient Descent is yet another method in order to calculate minimum value of our cost function.Imagine we that we graph our hypothesis function based on its fields θ (actually we are graphing the cost function as a function of the parameter estimates).

We are not graphing X and Y itself, but the parameter range of our hypothesis function and the cost resulting from selecting a particular set of parameters.



We will get success when we obtain minimum point in the above graph(red arrow show the minimum points).

Now the way in which we obtain this: Taking the derivative (the tangential line to a function) of our cost function.The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent. The size of each step is determined by the parameter α, which is called the learning rate.

For example, the distance between each 'star' in the graph above represents a step determined by our parameter α. A smaller α would result in a smaller step and a larger α results in a larger step. The direction in which the step is taken is determined by the partial derivative of cost function.

The gradient descent algorithm is:
repeat until convergence:

$$\theta j := \theta j - \alpha \frac{\partial}{\partial \theta j} J(\theta_0, \theta_1)$$

where j=0,1 represent feature index number(just an example with j=0,1 their may be many in case of multivariate gradient decent).

Below is some explaination which will clear your doubts and help to understand:

Finding $J'(\theta)$ for putting in gradient descent θ_j

$$J(\theta + h) = J(\theta) + hJ'(\theta)$$

$$\theta_{t+1} = \theta_t - \lambda J'(\theta_t)$$
$$\{h = -\lambda J'(\theta)\}$$

$$\implies J(\theta_t - \lambda J'(\theta_t)) = J(\theta) - \{\lambda J'(\theta_t)\}J'(\theta)$$
$$\implies J(\theta_t + 1) = J(\theta_t) - \lambda(J'(\theta_t))^2$$
$$\implies J(\theta_t + 1) <= J(\theta_t)$$

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{N}(\theta^T x_i - y_i)^2$$

Its differentiation,

$$J'(\theta) = \sum_{i=1}^{N} x_i(\theta^T x_i - y_i)$$

Now, putting this $J'(\theta)$ in different types of gradient descent.

Some key points:

* If our parameter α is too small then gradient decent will be very slow.
* If our parameter α is large then gradient decent can overshoot minimum and in this way fail to convergence and even divergence happens.
* Gradient Decent can converge to local minimum even by making parameter α as fixed, by taking small steps as it is approaching minimum value.
* At the minimum point our slope of graph(derivative)is "0". So we stop hare and minimum cost value is achieved.
* In practise Gradient Decent is not as directly used on data set , **Feature**

Scaling is done so as to make range of data feature equal to 1, for every feature.
* Debugging gradient descent: Make a plot with number of iterations on the x-axis. Now plot the cost function, J(θ) over the number of iterations of gradient descent. If J(θ) ever increases, then you probably need to decrease α.

4 Types of Gradient Descent

Various variants of gradient descent are defined on the basis of how we use the data to calculate derivative of cost function in gradient descent. Depending upon the amount of data used, the time complexity and accuracy of the algorithms differs with each other.

1. Batch Gradient Descent
2. Stochastic Gradient Descent
3. Mini-Batch Gradient Descent

4.1 Batch Gradient Descent

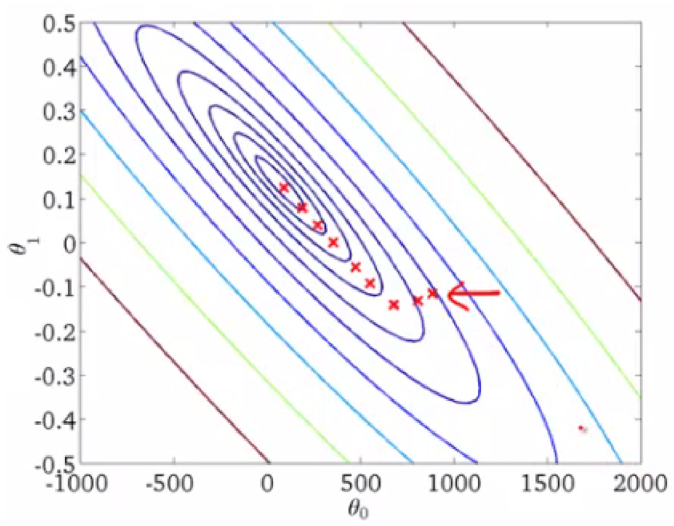
It is the first basic type of gradient descent in which we use the complete data-set available to compute the gradient of cost function. As we need to calculate the gradient on the whole data-set to perform just one update, batch gradient descent can be very slow and is not suitable for data-sets that don't fit in memory. After initializing the parameter with arbitrary values we calculate gradient of cost function using following relation:

Repeat until convergence -

$$\theta j := \theta j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

where 'm' is the number of training examples.

- If you have 300,000,000 records you need to read in all the records into memory from disk because you can't store them all in memory.
- After calculating sigma for one iteration, we move one step.
- Then repeat for every step.
- This means it will take a long time to converge.
- Especially because disk I/O is typically a system bottleneck anyway, and this will inevitably require a huge number of reads.



Batch gradient descent is not suitable for huge data-sets.

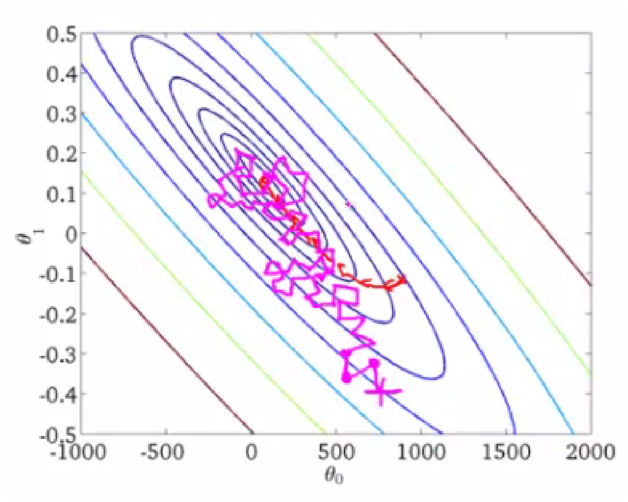
4.2 Stochastic Gradient Descent

Batch Gradient Descent turns out to be a slower algorithm. So, for faster computation, we prefer to use stochastic gradient descent. The first step of algorithm is to randomize the whole training set. Then, for updation of every parameter we use only one training example in every iteration to compute the gradient of cost function. As it uses one training example in every iteration, this algo is faster for larger data set. In SGD, one might not achieve accuracy, but the computation of results are faster. After initializing the parameter with arbitrary values we calculate gradient of cost function by repeating the following for $i = 1$ to m, where m is the number of training examples:

$$\theta j := \theta j - \alpha \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

for every $j = 0, ..., n$

SGD Never actually converges like batch gradient descent does, but ends up wandering around some region close to the global minimum.



4.3 Mini Batch Gradient Descent

Mini batch algorithm is the most favorable and widely used algorithm that makes precise and faster results using a batch of 'm' training examples. In mini batch algorithm rather than using the complete data set, in every iteration we use a set of 'm' training examples called batch to compute the gradient of the cost function. Common mini-batch sizes range between 50 and 256, but can vary for different applications. In this way, algorithm

- reduces the variance of the parameter updates, which can lead to more stable convergence.
- can make use of highly optimized matrix, that makes computing of gradient very efficient.

After initializing the parameter with arbitrary values we calculate gradient of cost function using following relation:
Let $b = 10$ and $m = 1000$
for $i = 1, 11, 21, ..., 991$

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} \left(h_\theta \left(x^{(k)} \right) - y^{(k)} \right) x_j^{(k)}, \text{for every } j = 0, 1, 2, .., n$$

5 Regularization

Regularization discourages learning a more complex or flexible model, so as to avoid the risk of over-fitting. The regularization term, or penalty, imposes a cost on the optimization function for overfitting the function or to find an optimal solution.

5.1 Lasso

Lasso stands for Least Absolute Shrinkage Selector Operator. Lasso assigns a penalty to the coefficients in the linear model using the formula below and eliminates variables with coefficients that zero. This is called shrinkage or the process where data values are shrunk to a central point such as a mean.
Lasso = Sum of Error + Sum of the absolute value of coefficients

$$L = \Sigma(\hat{Y}i - Yi)^2 + \lambda\Sigma|\beta|$$

So Lasso adds a penalty equal to the absolute value of the magnitude of the coefficients multiplied by lambda. The value of lambda also plays a key role in how much weight you assign to the penalty for the coefficients. This penalty reduces the value of many coefficients to zero, all of which are eliminated.

It adds a penalty to coefficients the model overemphasizes. This reduces the degree of overfitting that occurs within the model. The limitation of Lasso is that it does not work well with multicollinearity because Lasso might randomly choose one of the multicollinear variables without understanding the context. Such an action might eliminate relevant independent variables.

5.2 Ridge

Ridge assigns a penalty that is the squared magnitude of the coefficients to the loss function multiplied by lambda. As Lasso does, ridge also adds a penalty to coefficients the model overemphasizes. The value of lambda also plays a key role in how much weight you assign to the penalty for the coefficients. The larger your value of lambda, the more likely your coefficients get closer and closer to zero. Unlike lasso, the ridge model will not shrink these coefficients to zero.
Ridge Formula: Sum of Error + Sum of the squares of coefficients

$$L = \Sigma(\hat{Y}i - Yi)^2 + \lambda\Sigma\beta^2$$

The drawback of ridge is that it does not eliminate coefficients in your model even if the variables are irrelevant. This can be negative if you have more features than observations.

5.3 Elastic Net

The issue with Lasso is that for a high-dimensional data with few examples, it selects at most n variables before it saturates. Also if there is a group of highly correlated variables, then the Lasso tends to select one variable from a group and ignore the others. To overcome these limitations, the elastic net combines Lasso and Ridge.

$$L = \Sigma(\hat{Y}i - Yi)^2 + \lambda\Sigma|\beta| + \lambda\Sigma\beta^2$$

To conclude, Lasso, Ridge, and Elastic Net are excellent methods to improve the performance of your linear model. Lasso will eliminate many features, and reduce overfitting in your linear model. Ridge will reduce the impact of features that are not important in predicting your y values. Elastic Net combines feature elimination from Lasso and feature coefficient reduction from the Ridge model to improve your model's predictions.

6 References

1. https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote08.html
2. https://arxiv.org/abs/1609.04747
3. https://stat.ethz.ch/education/semesters/ss2016/CompStat/Exercises/LassoElasticNet.pdf