Machine Learning

Prepared by,

Debasis Mohanty (UEBA19001)

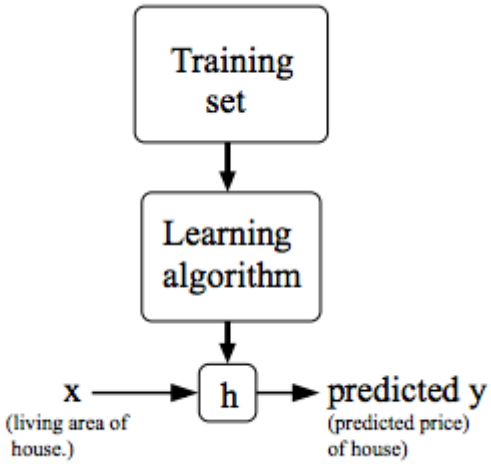
CONTENTS

Page

Model Representation:

To establish notation for future use, we'll use $x^{(i)}$ to denote the "input" variables (living area in this example), also called input features, and $y^{(i)}$ to denote the "output" or target variable that we are trying to predict (price). A pair $(x^{(i)}, y^{(i)})$ is called a training example, and the dataset that we'll be using to learn—a list of m training examples $(x^{(i)}, y^{(i)})$; i = 1, ... , m—is called a training set. Note that the superscript "(i)" in the notation is simply an index into the training set and has nothing to do with exponentiation. We will also use X to denote the space of input values, and Y to denote the space of output values. In this example, X = Y = $\mathbb{R}$.

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function h : X → Y so that h(x) is a "good" predictor for the corresponding value of y. For historical reasons, 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. When y can take on only a small number of discrete values (such as if, given the living area, we wanted to predict if a dwelling is a house or an apartment, say), we call it a classification problem.

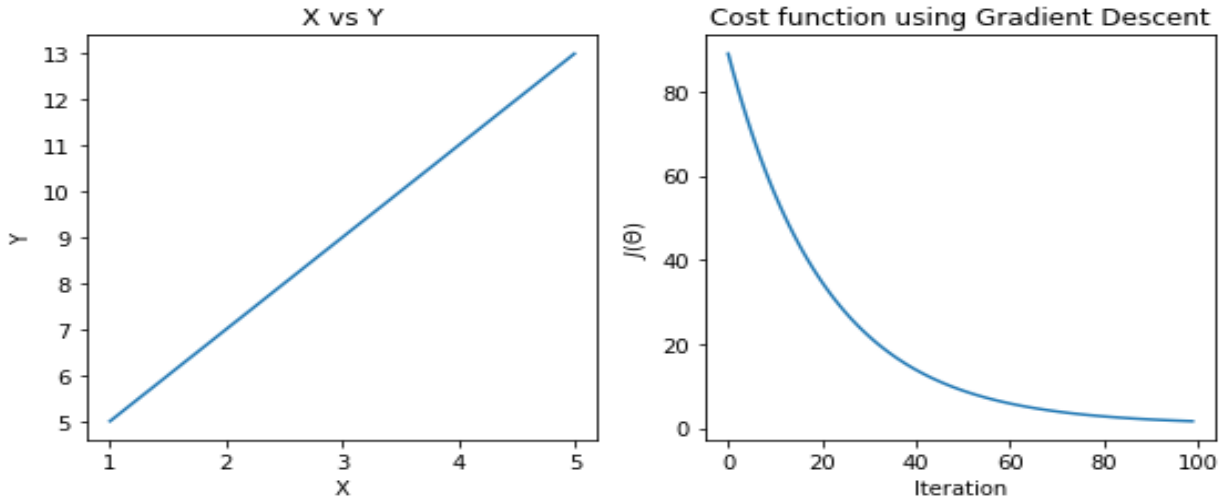
Cost Function:

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

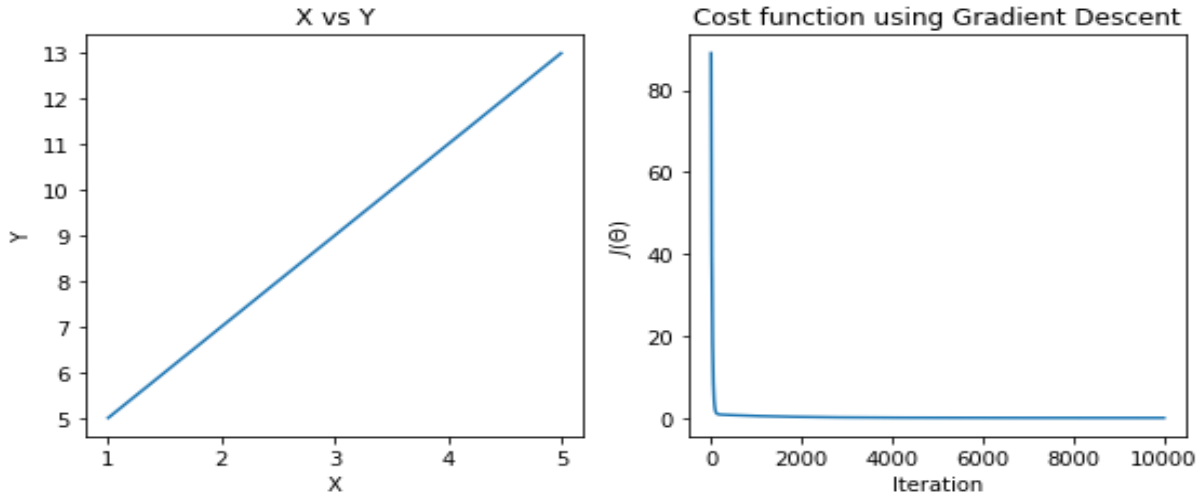
$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

To break it apart, it is ½ $\bar{x}$ where $\bar{x}$ is the mean of the squares of $h_\theta(x_i) - y_i$, or the difference between the predicted value and the actual value.

This function is otherwise called the "Squared error function", or "Mean squared error". The mean is halved ½ as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the ½ term. The following image summarizes what the cost function does: For number of iterations = 100 and learning rate = 0.001. The value of m = 2.368 and b = 0.737 for $\hat{y} = mx + b$ and the cost function $J(\theta) = 1.64$.



For number of iterations = 10000 and learning rate = 0.001. The value of m = 2.021 and b = 2.922 for $\hat{y} = mx + b$ and the cost function $J(\theta) = 0.001$.



Kindly change the value of iteration to get the above figures.

Cost Function Intuition:

If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make a straight line (defined by $h_\theta(x)$) which passes through these scattered data points.

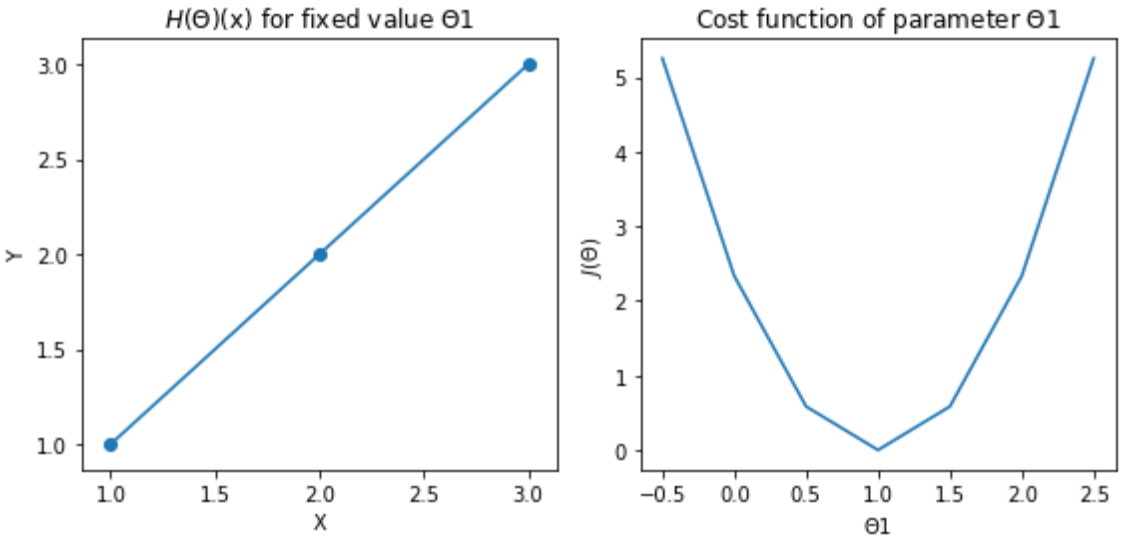
Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. Ideally, the line should pass through all the points of our training data set. In such a case, the value of $J(\theta_0, \theta_1)$ will be 0. The following example shows the ideal situation where we have a cost function of 0.

For $\theta_1 = 1$, $J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$ for predicted values $h_\theta(x_i) = [1,2,3]$ for $x_i = [1,2,3]$ and actual values $y_i = [1,2,3]$ being same. Thus for $\theta_1 = 1, J(\theta_1) = 0$.

Similarly for $\theta_1 = 0.5$, $J(\theta_1) = \frac{1}{2*3}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right] = 0.58$ for predicted values $h_\theta(x_i) = [0.5,1,1.5]$ and actual values $y_i = [1,2,3]$ being same. Thus for $\theta_1 = 1, J(\theta_1) = 0$.

Similarly for different values of theta we will have the following cost function as depicted below with the figure where Cost = $J(\theta_1)$

Cost= 5.25, theta= -0.5
Cost= 2.333333333333333, theta= 0
Cost= 0.5833333333333333, theta= 0.5
Cost= 0.0, theta= 1
Cost= 0.5833333333333333, theta= 1.5
Cost= 2.333333333333333, theta= 2
Cost= 5.25, theta= 2.5

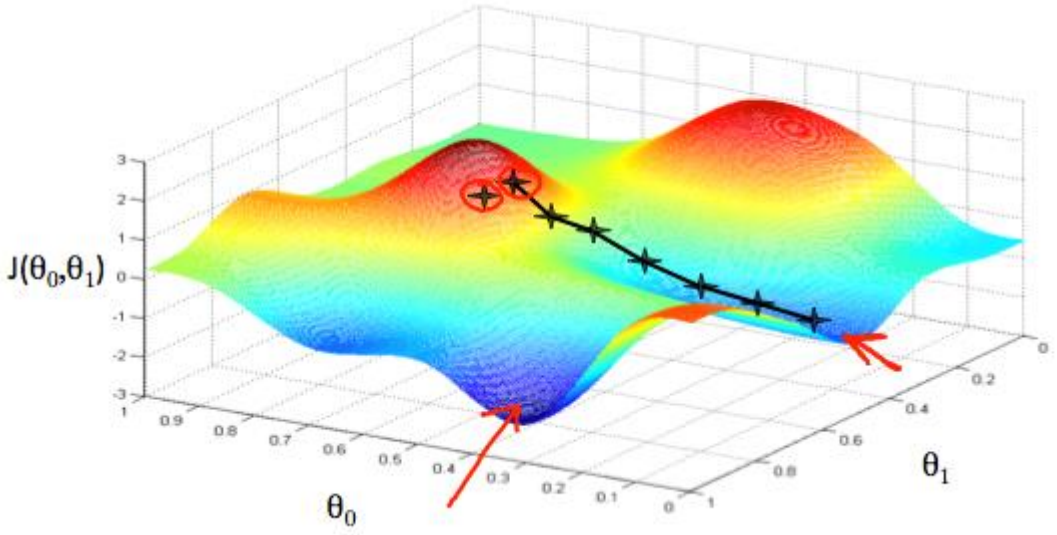


Gradient Descent:

So we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in the hypothesis function. That's where gradient descent comes in.

Imagine that we graph our hypothesis function based on its fields θ_0 and θ_1 (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 put θ_0 on the x axis and $\theta_1$on the y axis, with the cost function on the vertical z axis. The points on our graph will be the result of the cost function using our hypothesis with those specific theta parameters. The graph below depicts such a setup.



We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum. The red arrows show the minimum points in the graph.

The way we do this is by 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 $J(\theta_0, \theta_1)$. Depending on where one starts on the graph, one could end up at different points. The image above shows us two different starting points that end up in two different places.

The gradient descent algorithm is:

repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\delta}{\delta \theta_j} J(\theta_0, \theta_1)$$

where $j = 0,1$ represents the feature index number.

At each iteration j, one should simultaneously update the parameters $\theta_1, \theta_2, \ldots, \theta_n$. Updating a specific parameter prior to calculating another one on the j^{th} iteration would yield to a wrong implementation.

Correct: Simultaneous update	**Incorrect**
$temp0 := \theta_0 - \alpha \frac{\delta}{\delta \theta_j} J(\theta_0, \theta_1)$	$temp0 := \theta_0 - \alpha \frac{\delta}{\delta \theta_j} J(\theta_0, \theta_1)$
$temp1 := \theta_1 - \alpha \frac{\delta}{\delta \theta_j} J(\theta_0, \theta_1)$	$\theta_0 = temp0$
$\theta_0 = temp0$	$temp1 := \theta_1 - \alpha \frac{\delta}{\delta \theta_j} J(\theta_0, \theta_1)$
$\theta_1 = temp1$	$\theta_1 = temp1$

As $\theta_0 = temp0$, will update the $temp1$ details which will result in inconsistency of the θ_1.

Gradient Descent for Linear Regression:

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to :

repeat until convergence: {

$$\theta_0 \coloneqq \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

$$\theta_1 \coloneqq \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x_i) - y_i) x_i)$$

}

where m is the size of the training set, θ_0 a constant that will be changing simultaneously with θ_1 and x_i, y_i are values of the given training set (data).

Note that we have separated out the two cases for θ_j into separate equations for , θ_0 and θ_1; and that for θ_1 we are multiplying x_i, at the end due to the derivative. The following is a derivation of $\frac{\delta}{\delta\theta_j} J(\theta)$ for a single example :

$$\frac{\delta}{\delta\theta_j} J(\theta) = \frac{\delta}{\delta\theta_j} \frac{1}{2} (h_\theta(x) - y)^2$$

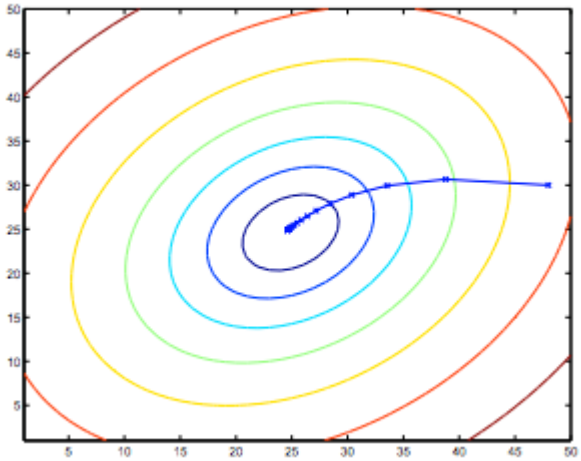
$$= 2 . \frac{1}{2} (h_\theta(x) - y) . \frac{\delta}{\delta\theta_j} (h_\theta(x) - y)$$

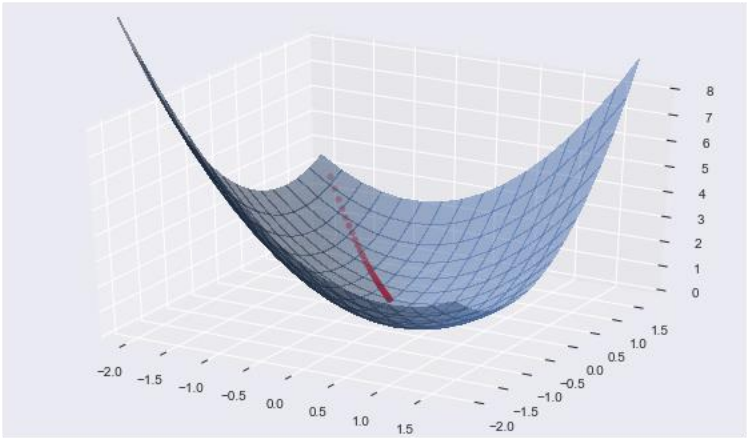
$$= ((h_\theta(x) - y) . \frac{\delta}{\delta\theta_j} (\sum_{i=0}^{n} \theta_i x_i - y)$$

$$= (h_\theta(x) - y) x_j$$

The point of all this is that if we start with a guess for our hypothesis and then repeatedly apply these gradient descent equations, our hypothesis will become more and more accurate.

So, this is simply gradient descent on the original cost function J. This method looks at every example in the entire training set on every step, and is called batch gradient descent. Note that, while gradient descent can be susceptible to local minima in general, the optimization problem we have posed here for linear regression has only one global, and no other local, optima; thus gradient descent always converges (assuming the learning rate α is not too large) to the global minimum. Indeed, J is a convex quadratic function. Here is an example of gradient descent as it is run to minimize a quadratic function.





The ellipses shown above are the contours of a quadratic function. Also shown is the trajectory taken by gradient descent, which was initialized at (48,30). The x's in the figure (joined by straight lines) mark the successive values of θ that gradient descent went through as it converged to its minimum.

Bibliography:

1. Book_Bishop_Pattern_Recognition_and_Machine_Learning__Information_Science_and_Statistics
2. Book_Deisenroth, Faisal, Ong_Mathematics for Machine Learning
3. Machine Learning Course at Coursera. https://www.coursera.org/learn/machine-learning
4. Code few developed by own and taken help from faculty and from https://stackoverflow.com/