MULTIVARIATE LINEAR REGRESSION

Multivariate Regression :

Linear Regression with multiple variables is also known as multivariate regression.

Consider our same example in Univariate Linear Regression, but in here we have additional features along with size(feet^2).

Size in sq. Ft (X_1)	Number of bedrooms (X_2)	Number of Floors (X_3)	Age of house(years) (X_4)	Price($) in 1000's (y)
2104	**5**	**1**	**45**	**460**
1416	**3**	**2**	**40**	**232**
1534	**3**	**2**	**30**	**315**
852	**2**	**1**	**36**	**178**
..	**..**	**..**	**..**	**..**
..	**..**	**..**	**..**	**..**

Notation:

m → number of data samples.

n → number of features.

x^i → input features of i^{th} training example(vector of n-dimension).

$X_j^{(i)}$ →value of feature j of i^{th} training example.

Hypothesis :

$$h_\theta(\mathrm{x}) = \theta_0 + \theta_1 \mathrm{x}1 + \theta_2 x2 + \theta_3 x3 + \theta_4 x4$$

consider X_0 =1 then our hypothesis becomes,

$$h_\theta(\mathrm{x}) = \theta_0\,\mathrm{x}0 + \theta_1 \mathrm{x}1 + \theta_2 x2 + \theta_3 x3 + \theta_4 x4$$

and the size of x^i becomes (n+1).

Now we can perform a matrix multiplication to get the above equation.

$$h_\theta(\mathrm{x}) = \theta^T X$$

Gradient Descent for Multiple Variables :

Cost function is defined as

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

$$= \frac{1}{2m} * \sum_{i=1}^{m} (\theta^T(x^i) - y^i)^2$$

$$= \frac{1}{2m} * \sum_{i=1}^{m} ((\sum_{j=0}^{n} \theta_j(x_j^i)) - y^i)^2$$

Gradient descent :

Repeat until converges {

$$\Theta\mathrm{j} = \Theta\mathrm{j} - \alpha \frac{\partial \mathrm{J}}{\partial \theta} J(\theta_0, \theta_1, \dots\dots, \theta_n)$$

}

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

$$= \frac{\partial}{\partial \theta_j} \frac{1}{2m} \cdot \sum_{i=1}^{m} (\theta_0 + \theta_1 x^i - y^i)^2$$

$$= \frac{\partial}{\partial \theta_0} \frac{1}{2m} \sum_{i=1}^{m} 2(\theta_0 + \theta_1 x^i - y^i) \frac{\partial}{\partial \theta_0}(\theta_0 + \theta_1 x^i - y^i)$$

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

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)$$

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{2m} \sum_{i=1}^{m} \frac{\partial}{\partial \theta_1} (\theta_0 + \theta_1 x^i - y^i)^2$$

$$= \frac{1}{2m} \sum_{i=1}^{m} 2(\theta_0 + \theta_1 x^i - y^i) \frac{\partial}{\partial \theta_1} (\theta_0 + \theta_1 x^i - y^i)$$

$$= \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)\left(0 + \frac{\partial}{\partial \theta_1}(\theta_1 x^i) + 0\right)$$

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)(x^i)$$

Gradient Descent

Previously (n=1):

Repeat {

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

(simultaneously update θ_0, θ_1)

}

New algorithm $(n \geq 1)$:

Repeat {

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

(simultaneously update θ_j for $j = 0, \ldots, n$)

}

$x_0^{(i)} = 1$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

...

Repeat until converges

{

$$\Theta j = \Theta j - \alpha \frac{1}{m} * \sum_{i=1}^{m} (h_{theta}(x^i) - y^i)\, x_j^i$$

}

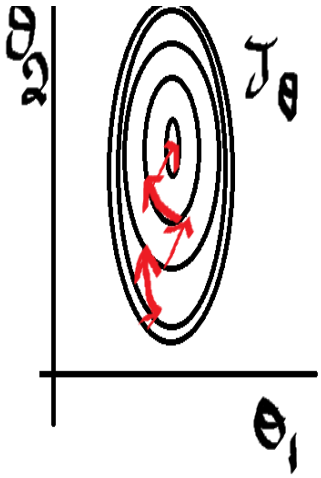
Don't forget to update "Θ" simultaneously from j=0,....,n.

While working on multivariate regression, if our model has lot of insightful features, then there will be a problem of overfitting, to overcome that we need to use regularization.

Feature Scaling: Making sure the features are in similar scale.

Ex. Suppose we have 2 parameters,

x1 →(0-2000)(size in ft^2) and x2→0-5(number of bedrooms), if we draw a contour plot for cost function then we get a graph that is lean and long so it takes a lot of time to find the minimum, so that we need to use feature scaling to speed up the convergence.

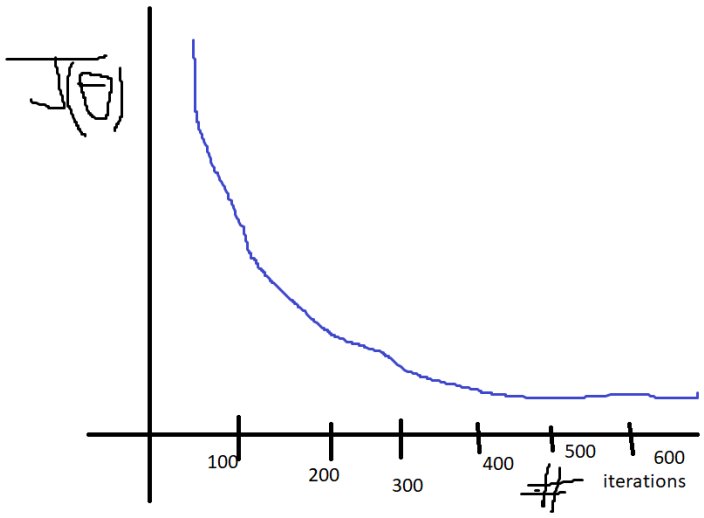


By using feature scaling we scale the features as $x_j^i = \frac{x_j^i}{\max(x^i) - \min(x^i)}$

Or we can use Mean Normalization to make the features on same scale, i.e. $x_j^i = \frac{x_j^i - \mu^i}{s^i}$

Debugging: How to make sure the gradient descent is working correctly ?

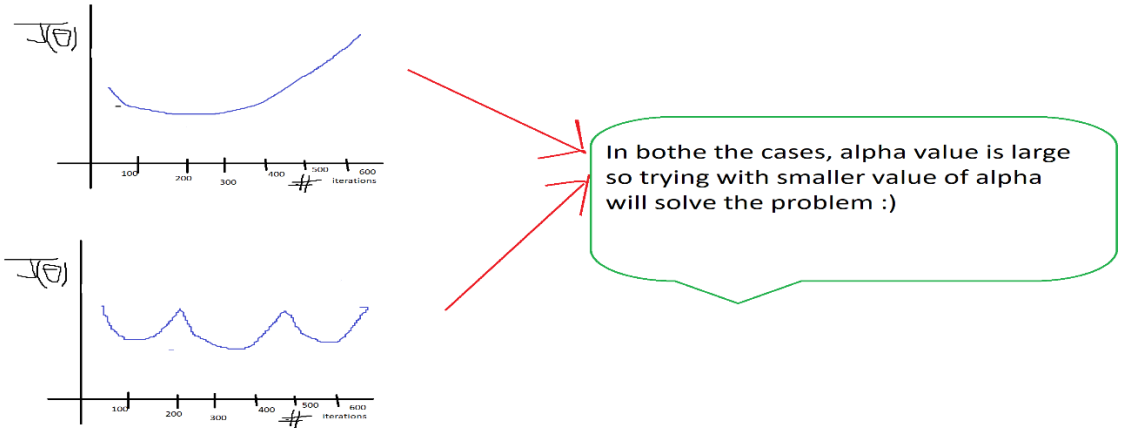
Plot a graph between your const function and the number of iterations.



After every iteration the cost function value should decrees, we can also use automatic convergence test i.e. is the value decreases less than 10^-3 we can stop, but it is not always the case, so better to go with plotting, so we can know whether our cost function is working fine or not.

How to choose learning parameter alpha: To choose alpha try, different learning rates.

Alpha = 0.001, 0.003,0.01,0.03,.0.1,0.3,..1. Also, from the cost function graph on the number of iterations also gives an idea on learning parameter, i.e.



- If alpha value is too small → slow convergence.
- If alpha value is too big → cost function value may not decrease, may not converge(slow converges can also occurs).

Polynomial Regression :

Our hypothesis function need not to be linear, if it doesn't fit the data.

We can change the behavior or curve if our hypothesis function by making it a quadratic, cubic or square root function(or any other form).

One thing is we need to keep in mind is, if we choose features this way then feature scaling becomes necessary, because if one of our features(say x1) ranges(1-1000), then its square and cube will have higher order, so it slow down the learning process.

Normal Equation :

Instead of running iterative algorithm, we can solve theta values analytically, i.e. we get weight parameters all in one go.

The equation is $\theta = (\mathrm{X}^T X)^{-1}(\mathrm{X}^T y)$

In the "Normal Equation" method, we will minimize J by explicitly taking its derivatives with respect to the θj 's and setting them to zero.

Derivation:

Suppose we have "m" training examples (x_i, y_i) where i= 1..m, and we have 'n' features for every training example i.e. $x^j = [x_i^1, x^2_{i,\ldots\ldots}, x_i^n]$, we add $x_i^0 = 1$ (n X1) vector, for easy matrix manipulations, like it acts as coefficients for our bias theta0), then our feature vector is of size(n+1,1).

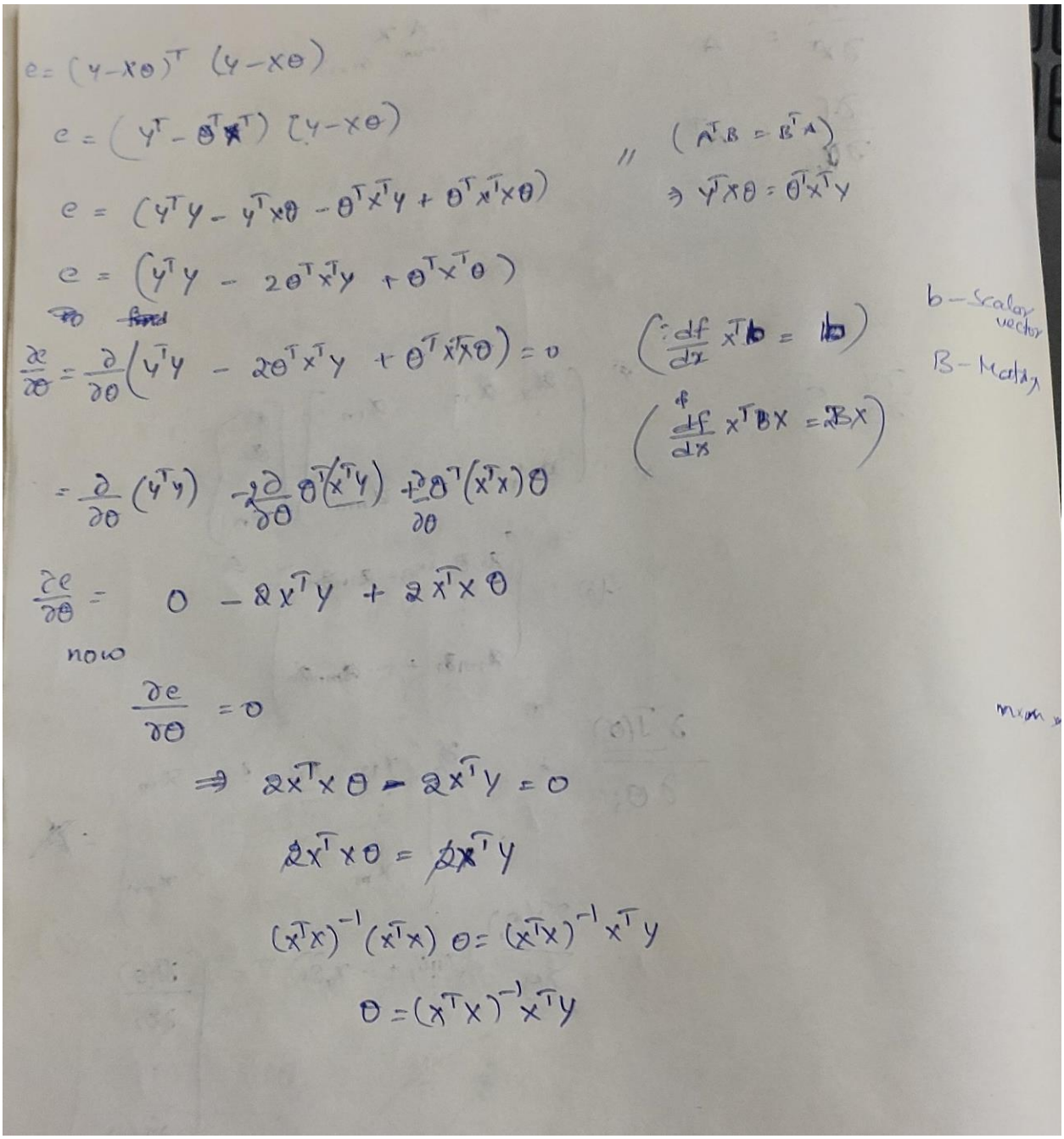
Now combining all these feature vectors to form a matrix representation, i.e.

$\mathrm{X} = \begin{bmatrix} x_1^0 & \cdots & x_1^n \\ \vdots & \ddots & \vdots \\ x_m^0 & \cdots & x_m^n \end{bmatrix}$ a (n+1 , m) matrix and our corresponding weight matrix $\theta = \begin{matrix} \theta^0 \\ \vdots \\ \theta^n \end{matrix}$ an n+1 dimensional vector. Each row of X represents one data sample.

Here we need to predict y with given X, both X and y are in different space, so we use normalization to find θ, for which $\mathrm{X}\,\theta \approx \mathrm{y}$, i.e. the deference between X θ and y i.e.

(y - X θ) should be minimum.

For minimization we use least square, e =(y - Xθ)^2, In matrix implementation we multiply the matrix by its transpose to generate square of that matrix.



If $X^T X$ is non-invertible (Matrix can't inverse), the possible reasons might be:

1. Redundant features, where two features are very closely related (i.e. they are linearly dependent)
2. Too many features (e.g. m ≤ n). In this case, delete some features or use "regularization".

https://www.coursera.org/learn/machine-learning/supplement/pKAsc/regularized-linear-regression

References :

1. http://mlwiki.org/index.php/Normal_Equation
2. https://www.youtube.com/watch?v=uoejt0FCWWA
3. http://www.rmi.ge/~kade/LecturesT.Kadeishvili/MathEconomics/Term3/Week3QuadraticLEC.pdf
4. http://www.gatsby.ucl.ac.uk/teaching/courses/sntn/sntn-2017/resources/Matrix_derivatives_cribsheet.pdf

The following is a comparison of gradient descent and the normal equation:

Gradient Descent	**Normal Equation**
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn^2)	O (n^3), need to calculate inverse of $(X^T X)^{-1}(X^T y)$
Works well when n is large	Slow if n is very large

Regularization:

- Keep all the features but reduce the magnitude of weight parameter (***θ***).
- Regularization works well when we have lot of insightful features.

Ridge Regression :

Let out hypothesis be,

$$h_\theta(\mathrm{x}) = \theta_0\,\mathrm{x0} + \theta_1\mathrm{x1} + \theta_2 x2 + \theta_3 x3 + \cdots + \theta_n xn$$

Ridge Regression learn weights (***θ***) using same least-squares but adds a penalty for large variations in ***θ*** parameters.

$$J(\theta) = \frac{1}{2m} * \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2 + \frac{\lambda}{2m} * \sum_{j=1}^{n} {\theta_j}^2$$

Updating θ values :

Repeat until converges

{

$$\Theta_0 \;=\; \Theta_0 - \alpha \frac{1}{m} * \sum_{i=1}^{m} (h_{theta}(x^i) - y^i)\, x_0^i$$

$$\Theta_j \;=\; \Theta_j - \alpha \left[\left(\frac{1}{m} * \sum_{i=1}^{m} (h_{theta}(x^i) - y^i)\, x_j^i \right) + \frac{\lambda}{m} * \theta_j \right]$$

$j \in \{1,2 \ldots n\}$

}

Where, λ is regularized parameter, it determines how much the cost of our theta are inflated.

Using the above cost function, we can smooth out our hypothesis to reduce overfitting,

- if the λ value is large then it may smooth out function too much and cause underfitting(because it reduces all the weight parameters $\theta_j \approx 0$).
- If the λ value is 0 or small then it may include all the weight parameters and leads to overfitting.

Lasso Regression :

Same as Ridge Regression, but here we use l1 regularization, i.e. we use absolute values of weight parameters θ_j. So, it is more accurate than Ridge, but it takes time compared to ridge, Lasso will be useful if the features are less.

$$J(\theta) = \frac{1}{2m} * \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2 + \frac{\lambda}{2m} * \sum_{j=1}^{n} \theta_j$$