⇒ Week-2 : Multivariable linear regression

Linear regression with multiple variables is known as multivariable linear regression.

→ Notation :

$x_j^{(i)}$ = value of feature j in the i^{th} training example

$x^{(i)}$ = the input (features) of the i^{th} training example → Generally a vector

m = the number of training examples

n = the number of features

eg :

	Size (feet²) x_1	No. of bedrooms x_2	No. of floors x_3	Age of home (years) x_4	Price ($1000) y
x^1	2104	5	1	45	460
x^2	1416	3	2	40	232
x^3	1534	3	2	30	315
x^4	852	2	1	36	178
	⋮	⋮	⋮	⋮	⋮

$n = 4$, $m = 47$ (say), $x_3^1 = 1$

$$x_1 = \begin{bmatrix} 2104 \\ 1416 \\ 1534 \\ 852 \end{bmatrix}$$

→ Hypothesis :

The multivariable form of the hypothesis func^n accommodating these multiple features is as follows :

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

→ Vectorization of hypothesis func^n

$$h_\theta(x^i) = [\theta_1 \quad \theta_2 \quad \cdots \quad \theta_n] \begin{bmatrix} x_0^i \\ x_1^i \\ \vdots \\ x_n^i \end{bmatrix}$$

$$= \theta^T x^i$$

* For convenience reasons, we assume $x_0^{(i)} = 1$ for $(i \in 1, \ldots, m)$

=/ Gradient descent for multiple variables

Hypothesis : $h_\theta(x) = \theta^T x$

$$= \theta_0 \underset{\downarrow \atop x_0 = 1}{x_0} + \theta_1 x_1 + \cdots + \theta_n x_n$$

Parameters : $\theta_0, \theta_1, \ldots, \theta_n$

Cost function :

$$J(\theta_0, \theta_1, \ldots, \theta_n) \equiv J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i\right)^2$$

→ Gradient descent algo :

Repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

$$\forall j = 0, \ldots, n$$

}

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

$$h_\theta x^i = \theta_0 \underset{\overset{\downarrow}{1}}{x_0^i} + \theta_1 x_1^i + \theta_2 x_2^i \ldots + \theta_n x_n^i$$

$$\Rightarrow \frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta x^i - y_i) \left[\frac{\partial}{\partial \theta_j} [\theta_0 x_0^i + \ldots + \theta_n x_n^i] \right]$$

$$\therefore \frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta x^i - y_i) \left[\frac{\cancel{\partial}\, \theta_0 x_0^i}{\partial \theta_0} \right]$$

As $x_0^i = 1$

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

Similarly,

$$\frac{\partial J(\theta)}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta x^i - y_i)\, x_1^i \cancel{\frac{\partial \theta_1}{\partial \theta_1}}$$

$$\frac{\partial (J(\theta))}{\partial \theta_2} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta x^i - y_i)\, x_2^i$$

$$\vdots \qquad \vdots \qquad \vdots$$

* Notice ~~the~~ that $\frac{\partial J(\theta)}{\partial \theta_0}$ & $\frac{\partial J(\theta)}{\partial \theta_1}$ are same for both multivariate & linear regression.

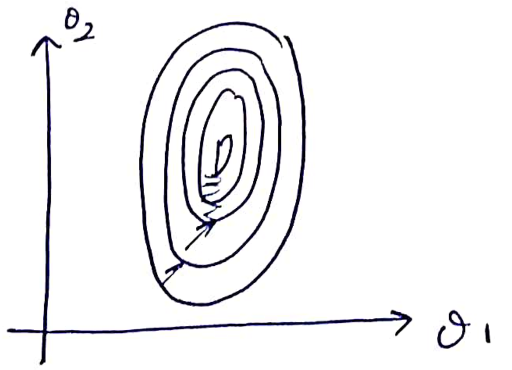
→ As no. of feature = 1 in linear regression, so

$x_1^i = x^i$ {or x_i}

⇒ Gradient descent in practice : Feature scaling

* If there is a huge range difference b/w the features, ~~it can take~~ running gradient descent on this kind of cost function can take a long time to find the global minimum.

eg: x_1 = Size (0-2000 ft)
x_2 = No. of bedrooms (1-5)



* We can speed up gradient descent by having each of our input values in roughly the same range. This is because θ will descend quickly on small ranges & slowly on large ranges, &

will oscillate 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,

$-1 \leq x_i \leq 1$ or $-0.5 \leq x_i \leq 0.5$

or $0 \leq x_i \leq 2$

The goal is to get all input variables into roughly one of these ranges, give or take a few.

For this, 2 techniques :

1) Feature scaling : Divide input values by the range (i.e. the max value minus the min value) of the input variable.

2) Mean normalization : $x_i := \dfrac{x_i - \mu_i}{s_i}$

μ_i = average / mean of all values for feature i

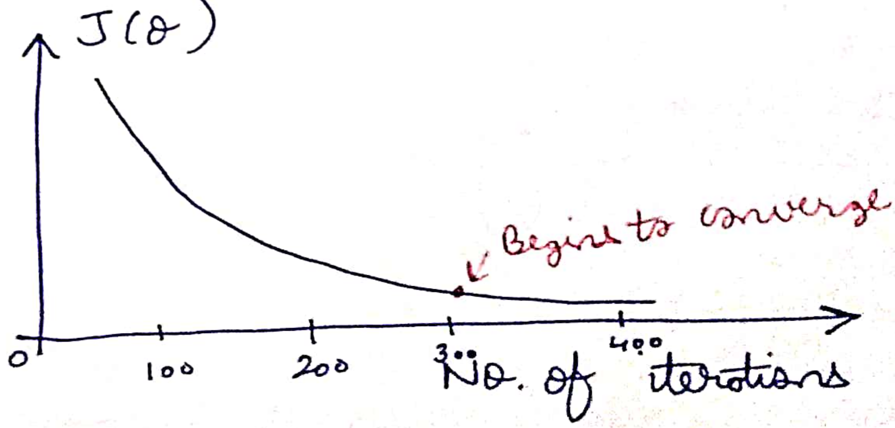
s_i = range of values (max - min), or it can be also be standard deviation.

⇒ Gradient descent in practice - Learning rate

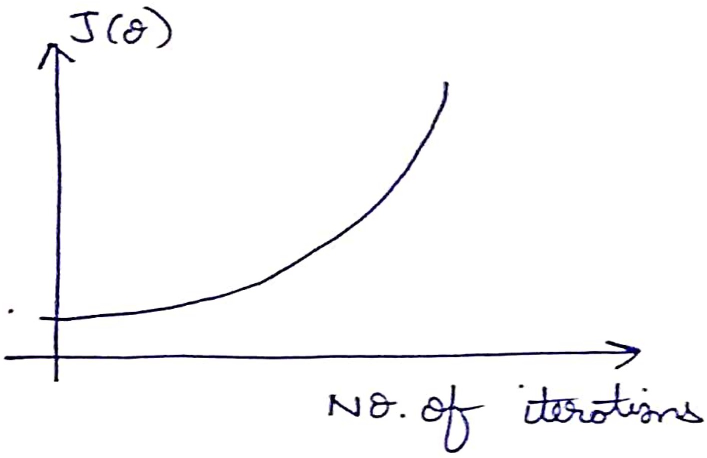
→ Debugging gradient descent : Make a plot with number of iterations of gradient descent on the x axis & the Cost function $J(\theta)$ on the y axis.

If $J(\theta)$ keeps decreasing, you're good to go! But if $J(\theta)$ ever increases, then you probably need to decrease α.

→ Automatic convergence test : Declare convergence if $J(\theta)$ decreases by less than E in one iteration, where E is some small value such as 10^{-3}. However, in practice it's difficult to choose this threshold value. So, ~~for~~ whenever possible, just plot it!

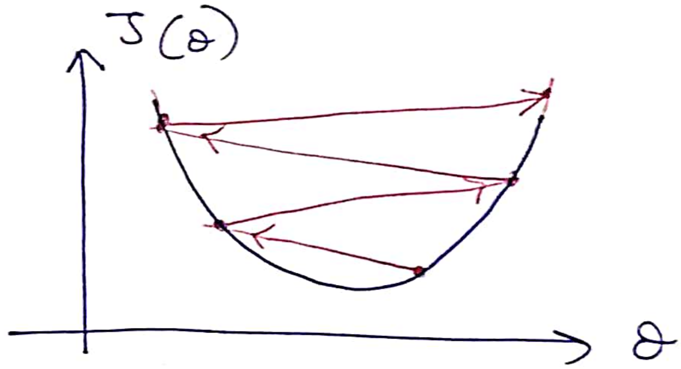


*



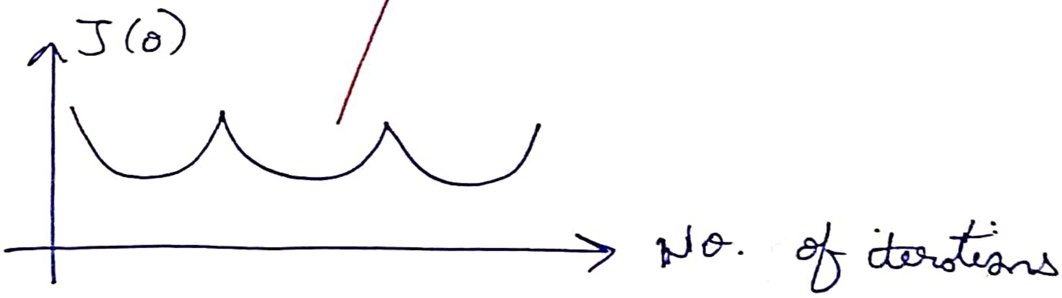
~~Bck~~

Behind the scenes ⇒



So, choose a smaller α in such case!

*



→ It's proven that for sufficiently small α, $J(\theta)$ should decrease on every iteration.

→ But if α is too small, gradient descent can be slow to converge.

→ If α is too large: $J(\theta)$ may not decrease on every iteration & thus may not converge.

⇒ Choosing features

→ We can improve our features & the form of our hypothesis function in a couple different ways.

We can combine multiple features into one.

eg: House price prediction:

→ 2 features : frontage (x_1) & depth (x_2)

→ You might decide that an imp. feature is the land area.

→ So create a new feature $x_3 = x_1 \times x_2$

→ $h(x) = \theta_0 + \theta_1 x_3$

→ Area is indeed a better indicator

→ Often by defining new features, you may get a more efficient model.

⇒ Polynomial regression

* Our hypothesis func^n need not be linear (a straight line) if that does not fit the data well.

* We can change the behaviour or curve of our hypothesis func^n by making it quadratic, cubic, square root func^n (or any other form).

eg: $h_\theta(x) = \theta_0 + \theta_1 x_1$

we can create additional features based on x_1, to get:

the quadratic func^n: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$

or the cubic func^n: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$

or the square root func^n: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$

→ How do we fit the model to this data?

* Set,

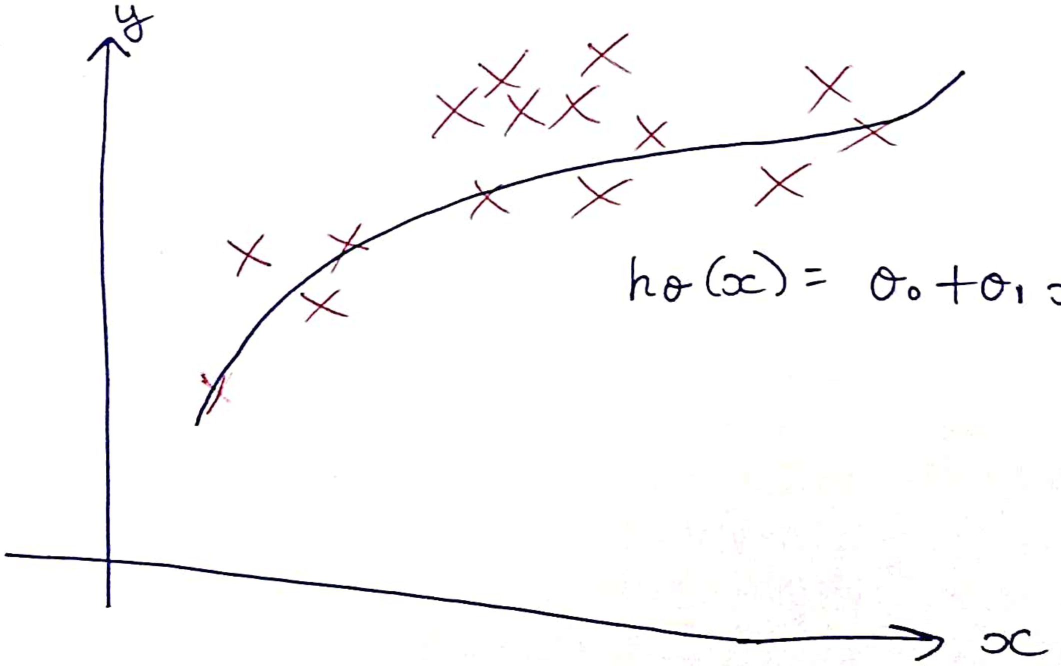
$x_1 = x$

$x_2 = x^2$

$x_3 = x^3$, $x_4 = \sqrt{x}$

$\vdots$

* By selecting features like this & applying the linear regression algos, you can do polynomial linear regression.



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

Polynomial regression

* Remember, feature scaling becomes even imp. here.

eg: if $x_1 \in \cancel{1-1000}\ [1, 1000]$

then $x_1^2 \in [1, 1{,}000{,}000]$

& $x_1^3 \in [1, 1{,}000{,}000{,}000]$

⇒ How does it work?

→ Supp. our cost func^n depends ~~and~~ on only 1 variable θ,

Let $J(\theta) = a\theta^2 + b\theta + c$; θ → real number

* $J(\theta)$ → quadratic func^n
* How do we minimize this?

Ans: Put $\frac{d}{d\theta} J(\theta) = 0$ & ~~solve for the~~ find values of θ.

~~value of~~

* ~~Alternatively~~ Hence, we find value of θ which minimizes $J(\theta)$.

→ In more complex problems:

* θ → $1 \times (n+1)$ vector of real numbers
* $J(\theta)$ → func^n of vector value
* How do we minimize this?

Ans: → Calculate $\frac{\partial J(\theta)}{\partial \theta_j}$ $\forall$ $j \in [0, n]$ & set to 0

→ Do that & solve for every value from $[\theta_0, \theta_n]$.

→ Gives values of θ which minimize $J(\theta)$.

⇒ Normal equation

* Instead of minimizing the cost function iteratively (through gradient descent), we can do it analytically – in other words, a mathematical eq^n that gives the result directly. This is called the Normal eq^n.

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

$\hat{\theta}$ → value of θ that minimizes the cost function

y → vector of target values containing y^1 to y^m.

X → "design matrix" — contains all the training data features in an $[m \times n+1]$ matrix

n → No. of features

m → No. of training samples

⇒ Workflow of normal eqⁿ method

eg :

	Size (feet^2) x_1	No. of bedrooms x_2	No. of floor x_3	Age of home (years) x_4	Price ($ 1000) y
x^1	2104	5	1	45	460
x^2	1416	3	2	40	232
x^3	1534	3	2	30	315
x^4	852	2	1	36	178

→ $m = 4$, $n = 4$

→ X = design matrix = $\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}_{m \times (n+1)}$

→ $y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$

→ Compute $(X^T X)^{-1} X^T y$ to get the value of θ which minimize the cost function.

→ ~~X is con~~

→ For m training samples & n features, each training sample is a (n+1) dimensional feature column vector.

$$x^i = \begin{bmatrix} x_0^i \\ x_1^i \\ \vdots \\ x_m^i \end{bmatrix}_{1\times(n+1)} \in \underset{\text{vector space}}{R}^{n+1}$$

→ Design matrix X is constructed by taking each training sample, determining its transpose & using it as a row for itself.

$$X = \begin{bmatrix} (x^1)^T \\ (x^2)^T \\ \vdots \\ (x^m)^T \end{bmatrix}_{m\times(n+1)}$$

→ ** If you're using the normal eq^n method then there's no need for feature scaling!

→ Gradient descent	Normal eq^n
* Need to choose α	* No need to choose α
* Needs many iterations	* No need to iterate
* Time complexity → $O(n^2)$	* Time complexity → $O(n^3)$, bcuz need to calculate X^TX
* Work well when n is large ($>10^4$)	* Slow when n is large ($>10^4$)

⇒ Normal eq^n non-invertibility

⇒ When computing $(X^TX)^{-1} X^Ty$, what if X^TX is non-invertible (singular / degenerate) ?!

→ Octave / Matlab can invert such invertible matrices with the pinv () function (pseudo inverse of func^n). So use pinv () instead of inv().

→ What does it mean for $X^T X$ to be non-invertible?

2 causes generally :

1) Redundant features , where 2 features are very closely related (i.e. they are linearly dependent)

eg: x_1 = size in feet

x_2 = size in meters

$\therefore x_2 \approx 3.2\, x_1$

2) Too many features : (eg : $m \leq n$, $m = 10$, $n = 100$)

Not enough data. So delete some features or use "regularization".

* Sol^n to above problems include deleting a feature that is linearly dependent with another or deleting one or more features when there are too many features.