Linear Regression

→ It is a supervised learning algorithm used to model the relationship between one or more independent variables (features) and a dependent variable (target), assuming that this relationship is linear.

* Core Concepts

→ It tries to find best fitting line (or hyperplane) through the data that minimizes the error between predicted and actual values.

- Simple Linear Regression (1 feature)

$$y = mx + b$$

→ 'x' : input feature
'y' : target output/predicted
'm' : slope of line (coefficient)
'b' : intercept (bias)

Date:
Page:

- Multiple Linear Regression (n features)

$$y = m_1 x_1 + m_2 x_2 + \dots m_n x_n + b$$

→ m_i : weight / coefficient for feature x_i
x_i : input feature
b : bias term
y : predicted output

* Goal of Linear Regression

→ Calculate the weight and bias for which error is minimum.

→ Error is usually "Mean Squared Error (MSE)"

$$\therefore J(w,b) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \quad \text{— (1)}$$

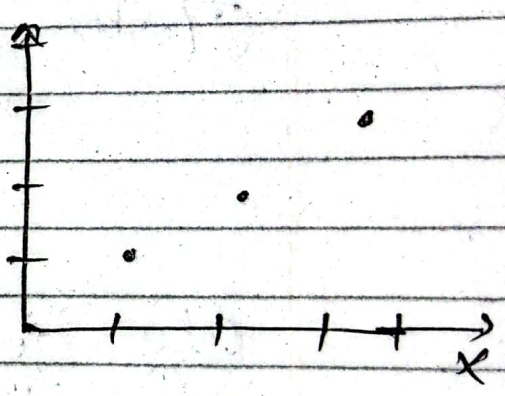
where, $J(w,b)$: cost function
y_i : actual value
$\hat{y}_i$: predicted value
n : no. of samples

For simplicity in calculation, let's take simple linear regression as example.

X	Y
1	1
2	2
3	3

It's equation will be:

$$y = mX + b$$



Initially, initialize weight and bias as:

$m = 1$

$b = 0$

Then, $\hat{y}_1 = 1 \times 1 = 1$

$\hat{y}_2 = 1 \times 2 = 2$

$\hat{y}_3 = 1 \times 3 = 3$

$$\therefore J(w,b) = \frac{1}{3} \sum_{i=1}^{3} (y_i - \hat{y}_i)^2$$

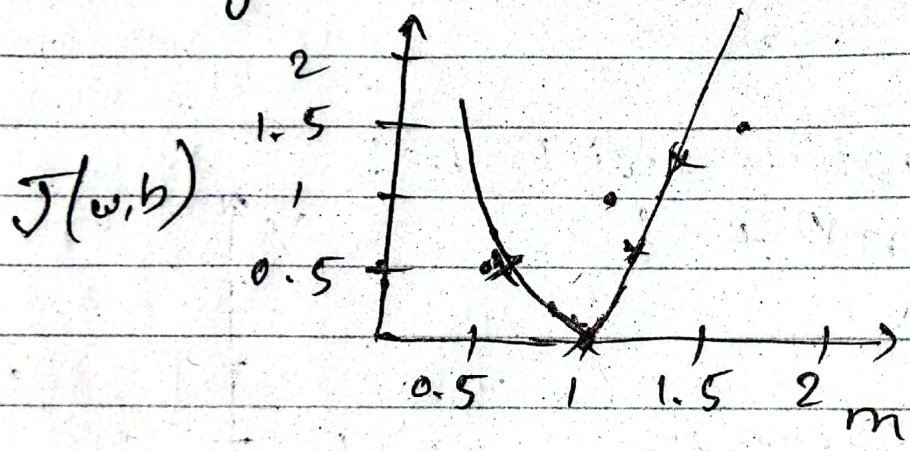
$$= \frac{1}{3} \left[(1-1)^2 + (2-2)^2 - (3-3)^2 \right]$$

$$= 0$$

Similarly, updating weights as:
$m = 0.5$
we will get $J(w,b) \approx 0.6$

For $m = 1.5$, $J(w,b) \sim 9$

Plotting $J(w,b)$ vs m:



We get a curve that has a point of minima for $J(w,b)$ for some value of m.

The curve converges at some point where we have optimal value of weights for minimum value of cost function.

We use "gradient descent" algorithm to converge by iterative method.

Computing gradient of the cost function

$$\frac{dJ(w,b)}{dm} = \frac{d}{dm}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right]$$

since $\hat{y}_i = mx_i + b$

$$\frac{dJ(w,b)}{dm} = \frac{d}{dm}\left[\frac{1}{n}\sum_{i=1}^{n}\{y_i - (mx_i + b)\}^2\right]$$

after partial derivative,

$$\frac{dJ(w,b)}{dm} = \sum_{i=1}^{n}\left[-\frac{2}{n} \times x_i \times (y_i - (mx_i + b))\right]$$

Similarly w.r.t. b

$$\frac{dJ(w,b)}{db} = \sum_{i=1}^{n}\left[-\frac{2}{n} \times \{y_i - (mx_i + b)\}\right]$$

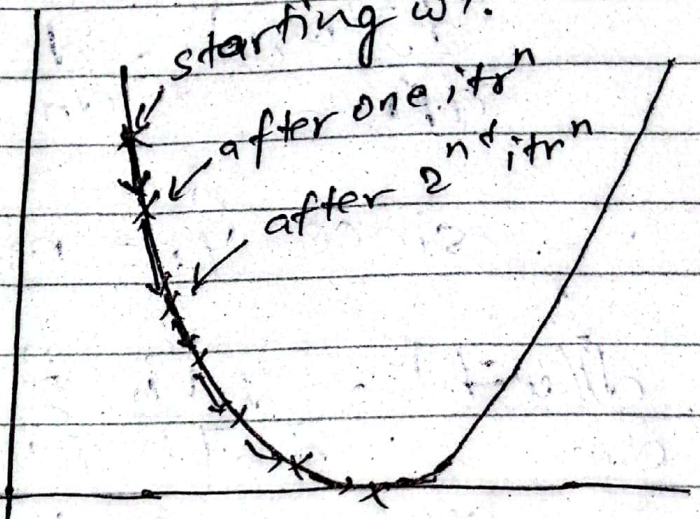
Now, according to convergence theorem

→ Repeat until convergence

$$m = m - \frac{dJ(w,b)}{dm} \times \alpha$$

$$b = b - \frac{d}{db} J(w,b) \times \alpha$$

where α is learning rate that defines step for updating weights.



We reduce the descent i.e if gradient is -ve, we are going downhill & hence we add gradient.
If gradient is +ve, we go opposite direction, hence subtract the gradient.

Hence for multiple linear regression,

i) Initialize weights and bias as 0.

ii) For each weight m_i, calculate gradient as:

$$\frac{dJ(w,b)}{dm_i} = -\frac{2}{n}\sum_{i=1}^{n}\left[x_i \times (y_i - \hat{y}_i)^2\right]$$

→ where $\hat{y}_i = m_1x_1 + m_2x_2 + \ldots m_nx_n + b$

iii) Update weight as:

$$m_i = m_i - \frac{dJ(w,b)}{dm_i} \times \alpha$$

iv) Similarly update bias 'b'.

v) Repeat until convergence.

This method is known as batch gradient descent where we calculate gradient using entire training sample for each change in weight. It can be slow for large datasets.

Introducing stochastic Gradient Descent.

* Changing notations as:

$\theta_j := m_j$ (for any $j \in 1$ to n)

$\hat{y}_i = h_\theta(x^{(i)})$

$J(w,b)$ as $J(\theta)$

$x_j^{(i)}$ → j = feature number / column
i = training sample number / row

Stochastic Gradient Descent

Repeat {

For i = 1 to m {

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

}

}

It tries to fit one training example and increases one by one fitting the parameters respectively rather than scanning through whole dataset ~~rather than~~ for each change in parameter

Let $\nabla_\theta J(\theta)$ be derivative of J w.r.t. θ.

where $\theta \in \mathbb{R}^{n+1}$ for 'n' dimension/feature

$$\therefore \nabla_\theta J(\theta) = \begin{bmatrix} \frac{dJ}{d\theta_0} \\ \frac{dJ}{d\theta_1} \\ \frac{dJ}{d\theta_2} \end{bmatrix}$$ for θ three dimensions

Let $A \in \mathbb{R}^{2\times 2}$ $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$

Then, for $f(A)$ i.e $f\ \mathbb{R}^{2\times 2} \longrightarrow \mathbb{R}$

Let $f(A) = A_{11} + A_{12}^2$

Then, $f\left(\begin{bmatrix} 5 & 6 \\ 2 & 3 \end{bmatrix}\right) = 5 + 6^2$

Hence,

$$\nabla_A f(A) = \begin{bmatrix} \frac{df}{dA_{11}} & \frac{df}{dA_{12}} & \cdots & \frac{df}{dA_{1n}} \\ \vdots & \vdots & & \\ \frac{df}{dA_{n1}} & \frac{df}{dA_{n2}} & \cdots & \frac{df}{dA_{nn}} \end{bmatrix}$$

Date
Page

For above example,

$$\nabla_A f(A) = \begin{bmatrix} 1 & 2A_{12} \\ 0 & 0 \end{bmatrix}$$

To compute value of θ, we find global minima for $J(\theta)$ which is achieved by:

$$\nabla_\theta J(\theta) = \vec{0}$$

And solve for θ.

Some formulae

trace(A) = sum of diagonal entries

$$= \sum_i A_{ii}$$

i) $\text{trace}(A) = \text{trace}(A^T)$

ii) $f(A) = \text{tr}(AB)$

$\nabla_A f(A) = B^T$

iii) $\text{tr}(ABC) = \text{tr}(BCA)$

$\text{tr}(ABC) = \text{tr}(CAB)$

iv) $\nabla_A \operatorname{tr} AA^T C$

$= CA + C^T A$

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

Extracting training example as matrix

$$X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ (x^{(3)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$$

Similarly, $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \end{bmatrix}$

Then,

$$X\theta = \begin{bmatrix} x^{(1)T}\theta \\ x^{(2)T}\theta \\ \vdots \\ x^{m}{}^{T}\theta \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \text{ or } \begin{bmatrix} h_\theta(x^{(1)}) \\ h_\theta(x^{(2)}) \\ \vdots \\ h_\theta x^{m} \end{bmatrix}$$

predic. y

Now,
We know that $X\theta$ gives us prediction of y.

Then, $X\theta - y = \begin{bmatrix} h_\theta x^{(1)} - y^{(1)} \\ h_\theta x^{(2)} - y^{(2)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix}$

And also,

$$z^T \cdot z = \sum_i z^2$$

Hence,

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

Now, we can write $J(\theta)$ as:

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

$$= \frac{1}{2} (X\theta - y)^T \cdot (X\theta - y)$$

$$\therefore \nabla_\theta J(\theta) = \nabla_\theta \frac{1}{2}(X\theta - y)^T (X\theta - y)$$

$$= \frac{1}{2}\nabla_\theta (\theta^T X^T - y^T)(X\theta - y)$$

$$= \frac{1}{2}\nabla_\theta \left[\theta^T X^T X\theta - \theta^T X^T y - y^T X\theta - y^T y\right]$$

$$= \frac{1}{2}\left[X^T X\theta + X^T X\theta - X^T y - X^T y\right]$$

$$= X^T X\theta - X^T y$$

Now, for global minima,

$$\nabla_\theta J(\theta) = \vec{0}$$

$$\text{or, } X^T X\theta - X^T y = \vec{0}$$

$$\text{or, } X^T X\theta = X^T y \quad \text{"Normal equation"}$$

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