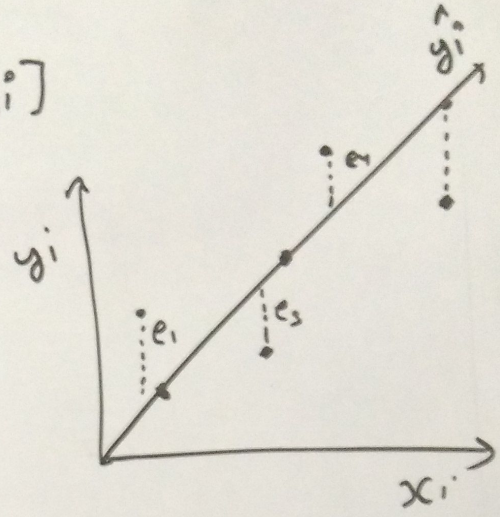
Machine learning

Linear Regression $[x_i, y_i]$

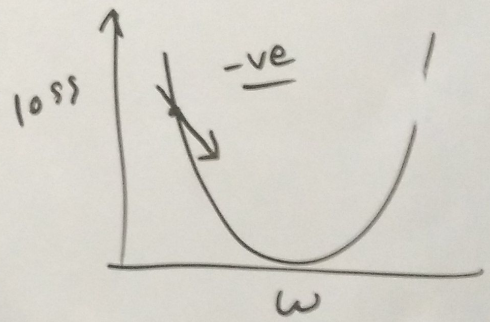


Model / hypothesis:

$$\hat{y_i} = \omega \cdot x_i + b$$

Cost / Loss function:

$$\text{Loss/cost (MSE)} = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y_i} - y_i)^2$$



Optimization objective:

$$\omega^*, b^* = \underset{\omega, b}{\text{argmin}} (\text{loss})$$

calculation

$$\frac{\partial \text{Loss}}{\partial \omega} = \frac{\partial}{\partial \omega} \left[\frac{1}{2n} \sum_{i=1}^{n} \left[(\omega \cdot x_i + b) - y_i \right]^2 \right.$$

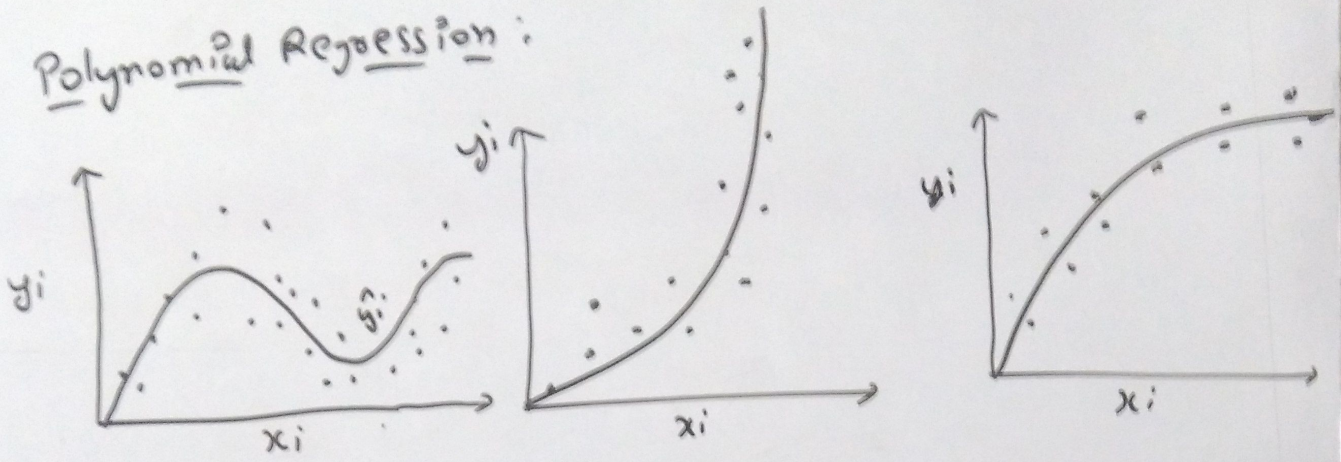
$$\Delta\omega = \begin{cases} \frac{\partial \text{loss}}{\partial \omega} = \frac{1}{2n} \sum_{i=1}^{n} 2(\hat{y_i} - y_i) x_i = \frac{1}{n} \sum_{i=1}^{n} (\hat{y_i} - y_i) \cdot x_i \\ \Delta b \quad \frac{\partial \text{Loss}}{\partial \omega} = \frac{1}{n} \sum (\hat{y_i} - y_i) \end{cases}$$

updating

$$\begin{bmatrix} \omega = \omega - \Delta\omega \\ b = b - \Delta\omega \end{bmatrix} \longrightarrow \begin{bmatrix} \omega = \omega - \alpha \cdot \Delta\omega \\ b = b - \alpha \Delta\omega \end{bmatrix} \quad \text{L.R}$$

Machine learning

Polynomial Regression :



Model/Hypothesis :

$$\hat{y}_i = w_1 \cdot x_i^2 + w_3 \cdot x_i + w_2 \cdot \sin(x_i) + b$$

loss/cost :

$$\underset{(MSE)}{\text{loss}} = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

Gradient descent & derivative

$$\Delta w_1 = \frac{1}{n} \sum (\hat{y}_i - y_i) \cdot (x_i^2)$$

$$\Delta w_2 = \frac{1}{n} \sum (\hat{y}_i - y_i) \cdot (x_i)$$

$$\Delta w_3 = \frac{1}{n} \sum (\hat{y}_i - y_i) \cdot (\sin(x_i))$$

$$\Delta b = \frac{1}{n} \sum (\hat{y}_i - y_i)$$

$$\boxed{w = w - \alpha \cdot \Delta w}$$

Machine learning

feature Scaling

Standardization :

General

$$x_i' = \frac{x_i - \mu}{\sigma}$$

where $x_i' \sim N(0,1)$ (μ, σ)

μ = mean(x_i) ; σ = std. deviation

Normalization (MinMax Scaling)

Images

$$x_i' = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

where $x_i' \in [0,1]$

Why these

- to make all input into same scale (kg, Pound)
- Gradient descent work efficiently if input on different scale which lead to update some weight faster than others weights. bcz

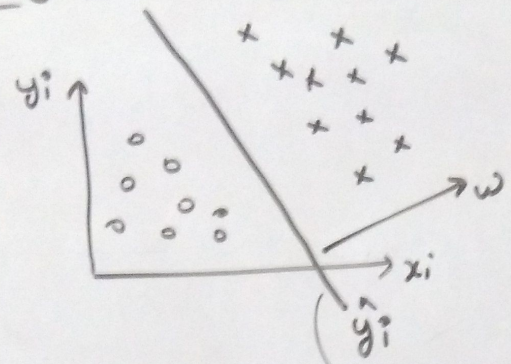
$$\Delta w = \frac{1}{n}\sum(\hat{y}_i - y_i) \cdot x_i$$ [Simple regression]

update is depend upon input.

- Must do for G.D and distance based algoithm like (KNN, K-means, Logistic R., SVM, N/Nw, PCA)
- Not important in tree based algo, Naive Bayes Algo.

Machine learning

Logistic Regression [Probabilistic Approch]



$\hat{y_i} = wx_i + b$

where $\hat{y_i} \in (-\infty, \infty)$

Decision Boundary

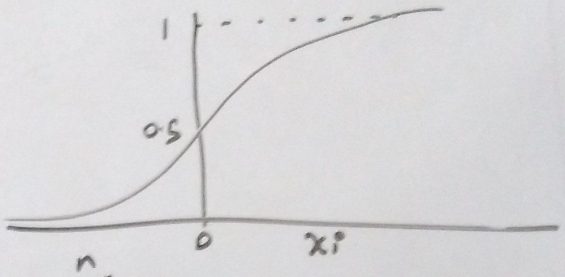
[if ($\hat{y_i} \to$ +ve) : ⊗
else : ⊙ :

- Badly impact by Outlier.

Tampering (Squeezing)

$\hat{y_i} = \sigma(wx_i + b) = p_i$ [Prob. Score]

Sigmoid (σ) $= \frac{1}{1 + e^{-x}}$



$0 \le \hat{y_i} \le 1$;

Threshold (Z) = 0.5 means

$\hat{y_i} \ge 0.5$: ⊗ +ve class

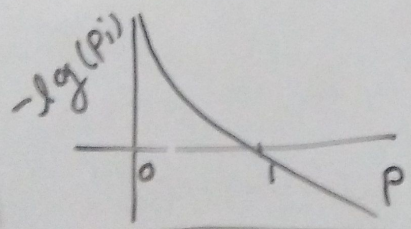
$\hat{y_i} < 0.5$: ⊙ -ve class

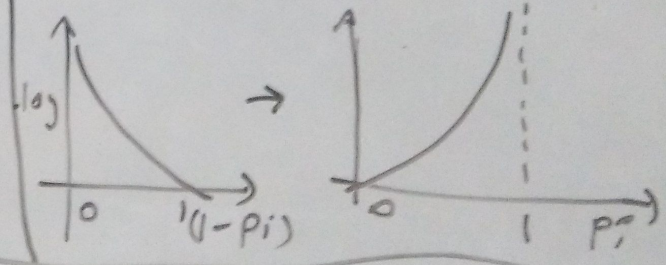
Why Sigmoid : easy to differentiate, Output in probability Score

$\sigma'(x) = \sigma(x)(1 - \sigma(x))$

Loss / Log loss

$y_i = 1 \to -\log(p_i)$ | $y_i = 0 \to -\log(1 - p_i)$





$$\text{log loss} = \frac{1}{n}\sum\left[-y_i(\log(p_i)) - (1 - y_i)(\log(1 - p_i))\right]$$

Machine learning

Regularization:

→ L1 Regularization (Lasso)

$$= \frac{1}{n}\left(\sum_{i=1}^{n} \text{Loss}_i + \lambda \cdot \sum_{j=1}^{d} |w_j|\right)$$

w: [0, 1, 0, 0, 5, 0]

$\lambda \uparrow$ → Sparsity in w ↑

→ L2 Regularization (Ridge)

$$= \frac{1}{n}\left(\sum_{i=1}^{n} \text{Loss}_i + \lambda \cdot \sum_{j=1}^{d} w_j^2\right)$$

w: [0.2, 0.7, 0.3, 0.5]

$\lambda \uparrow$ → distributed w increases.

Overall: $\lambda \uparrow$ → underfit | if $\lambda = 0$: dont penalize weights
$\lambda \downarrow$ → overfit

Gradient descent:

$$\Delta w_j = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \cdot x_i \rightarrow \Delta w_j = \left[\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \cdot x_i + \frac{\lambda}{n} w_j\right]$$