XG Boost Regressor

Exp	Yap	Salary	BasePredict	Residual
2	Yes	40	51	-11
2.5	Yes	42	51	-9
3	No	52	51	1
4	No	60	51	9
4.5	Yes	62	51	11

$$\text{Base Prediction} = \frac{(40+42+52+60+62)}{5} = 51.2 \Rightarrow 51$$

$$SW_{root} = \frac{(-11-9+1+9+11)^2}{5+1} = \frac{1}{6}$$

$$SW_{LC} = \frac{(-11)^2}{1+1} = \frac{121}{2} = 60.5$$

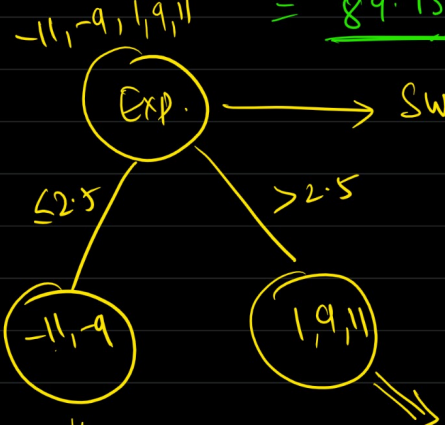
$$SW_{RC} = \frac{(-9+1+9+11)^2}{4+1} = \frac{144}{5} = 28.8$$

$$SW_{gain} = (SW\ child) - SW_{root}$$

$$= (60.5 + 28.8) - 0.166$$

$$= 89.133$$

* Split will be based on highest SW gain =



SW = 0.166

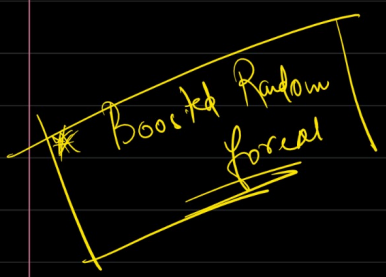
$$SW_{RC} = \frac{(1+9+11)^2}{3+1} = \frac{(21)^2}{4} = 110.23$$

$$\frac{(-11-9)^2}{2+1} = \frac{400}{3} = 133.3$$

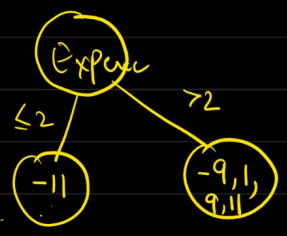
$$SW_{gain} = 133.3 + 110.23 - 0.166$$

$$\approx 244$$

Highest gain, so split on this row

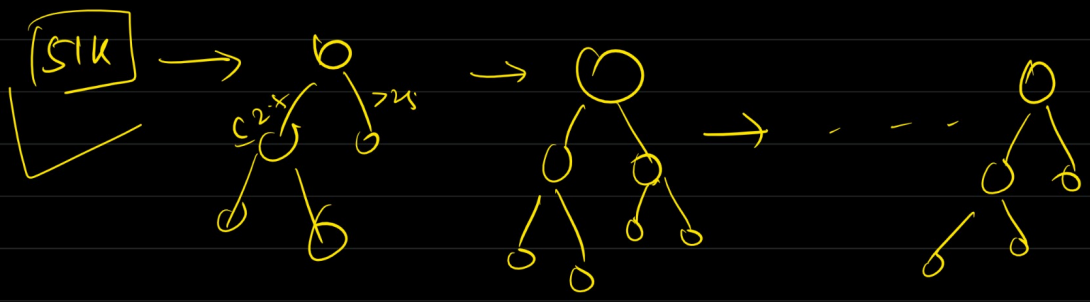


[-11, -9, 1, 9, 11]



$$*\ SW_{gain} = \frac{(\sum R)^2}{\sum Pr(1-pr) + \lambda}$$

$$*\ SW_{req} = \frac{(\sum R)^2}{\text{No of residual} + \lambda} = L$$

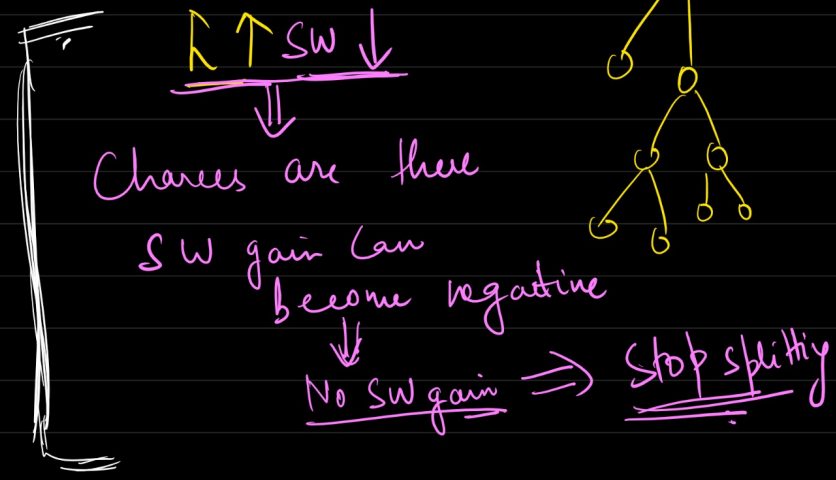


(until the requirement of DT is a ~~parameter~~ model)

Prediction = $51 + \alpha_1(DT_1) + \alpha_2(DT_2) + \cdots \alpha_n(DT_n)$

$SW = \frac{(\sum \text{Residual})^2}{\sum Pr(1-Pr) + \lambda}$ ↑

↳ hyper parameter



$\lambda \uparrow SW \downarrow$

⇓

Chances are there SW gain can become negative

⇓

No SW gain ⇒ Stop splitting

$\sum Pr(1-pr)$ ⇒ cover value.

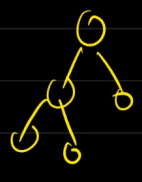
Any SW less $Pr(1-Pr)$, we stop splitting

$SW = \frac{(\sum R)^2}{Pr(1-Pr) + \lambda}$ ✓

$SW < 0.25$

$0.5(0.5) = 0.25$

λ ↑ ⇑ regularization Para



IG

Entropy / Gini

binary / multiclass

Xtreme → ^

* XGBoost

↳ optimised version of GB

↳ ^ (regularization → to prevent overfitting)

↳ fast (Parallel computation)

↳ Separate library itself.

↳ Good with speed and accuracy.

↳ Works very well with large dataset.

* Time complexity

↳ N → no of training sample

↳ M → No of features

↳ T → is no of trees.

d → maximum dep.

✓ XGBoost ⇒ $O(T \cdot NM \log N)$ ⇒ in practice this is very very fast

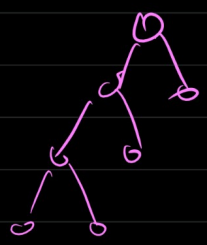
✓ Gradient Boosting ⇒ $O(T \cdot NM \log N)$

→ Separate library

→ Parallel computation

AdaBoost ⇒ N, M, T

$O(T \cdot NM)$



→ DT is built recursively

$N \log N$ =

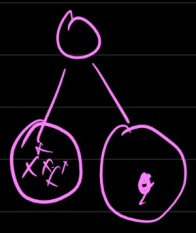
* Gradient Boost.

→ Computationally expensive

→ Prone to overfitting. (better than AdaBoost)

* AdaBoost.

↳ for small dataset

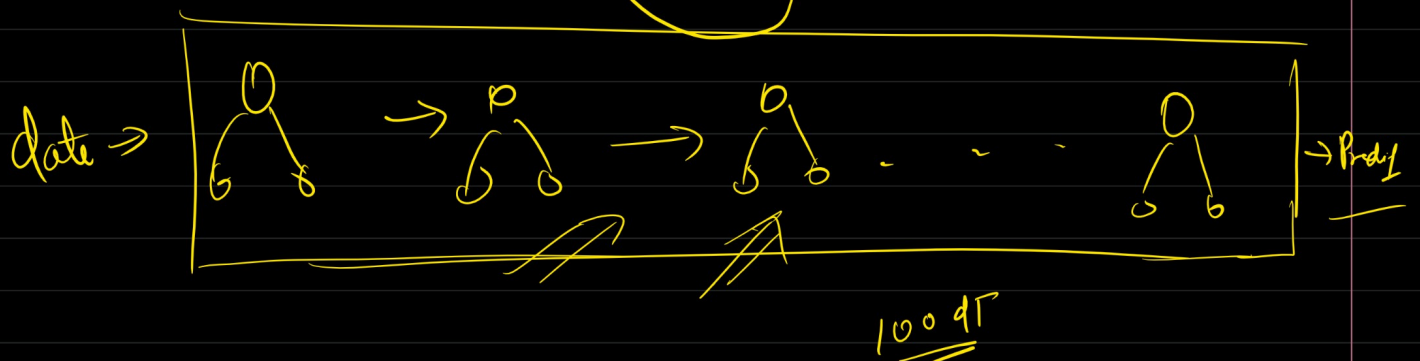
↳ Sensitive to noise & outlier → 

Black box model

⇓

→ Model which has not explainability power

$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$ → White - Box Model



100 DT