

## XGBoost (eXtreme Gradient Boosting)

### 1. What is XGBoost?

- XGBoost is an optimized, distributed **gradient boosting** library designed for efficiency, flexibility, and portability.
- It's a highly successful implementation of the gradient boosting framework, often achieving state-of-the-art results on structured/tabular data.
- It builds upon the core ideas of gradient boosting (sequential model building, fitting residuals/gradients) but incorporates several significant enhancements for speed and accuracy.

### 2. Key Features & Enhancements over Standard Gradient Boosting

- **Regularization:** XGBoost includes L1 (Lasso, controlled by `alpha` or `reg_alpha`) and L2 (Ridge, controlled by `lambda` or `reg_lambda`) regularization terms directly in its objective function. This penalizes model complexity (number of leaves and leaf weights), helping prevent overfitting and improve generalization.
- **Parallel Processing:** While trees are still built sequentially (boosting), XGBoost can parallelize parts of the tree construction process internally (e.g., finding the best split across features), making it significantly faster than traditional gradient boosting implementations.
- **Handling Missing Values (Sparsity-Aware Split Finding):** XGBoost has a built-in mechanism to handle missing data. During training, it learns a default direction (left or right child node) for samples with missing values at each split, based on which direction maximizes the gain. This avoids the need for manual imputation.
- **Tree Pruning:** XGBoost grows trees up to a specified `max_depth` and then may prune them backward. It uses a `gamma` (or `min_split_loss`) parameter; a split is only made if the loss reduction (gain) from the split is greater than `gamma`. This acts as a form of post-pruning to control complexity.
- **Hardware Optimization (Cache Awareness & Out-of-Core Computation):** XGBoost employs techniques like organizing data into blocks and using cache-aware algorithms to optimize hardware usage, especially for large datasets. It can also handle datasets that don't fit into memory ("out-of-core").
- **Weighted Quantile Sketch:** For finding split points efficiently on large datasets, XGBoost can use an approximate greedy algorithm involving sketches (specifically, a weighted quantile sketch) to propose candidate split points effectively.
- **Built-in Cross-Validation:** XGBoost allows you to perform cross-validation at each boosting iteration, making it easier to find the optimal number of boosting rounds.

### 3. Objective Function (Conceptual)

XGBoost minimizes a regularized objective function:

$$Obj(\Theta) = \sum_{i=1}^N l(y_i, \hat{y}_i) + \sum_{m=1}^M \Omega(f_m)$$

Where:

- $l(y_i, \hat{y}_i)$  is the loss function for individual predictions (e.g., Log Loss for classification, MSE for regression).
- $\Omega(f_m)$  is the regularization term for tree  $m$ , penalizing complexity.
- $\hat{y}_i$  is the final prediction for sample  $i$ , which is the sum of predictions from all trees  $M$ .

#### 4. Regularization Term ( $\Omega$ )

The regularization term penalizes both the number of leaves ( $T$ ) in a tree and the magnitude of the scores ( $w$ ) assigned to the leaves:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 + \alpha \sum_{j=1}^T |w_j|$$

- $\gamma$ : Penalty for the number of leaves (controls pruning).
- $\lambda$ : L2 regularization penalty on leaf scores (weights).
- $\alpha$ : L1 regularization penalty on leaf scores (weights).

#### 5. Split Finding (Similarity Score & Gain)

- XGBoost uses the first-order gradient ( $g_i$ ) and second-order gradient (Hessian,  $h_i$ ) of the loss function to approximate the objective function.
- It calculates a Similarity Score for a set of residuals in a potential leaf node  $j$ :

$$\text{SimilarityScore} = \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda}$$

Where  $I_j$  is the set of instances in leaf  $j$ .

- The quality of a potential split is evaluated by the Gain:

$$\text{Gain} = \text{Similarity}_{\text{Left}} + \text{Similarity}_{\text{Right}} - \text{Similarity}_{\text{Parent}} - \gamma$$

- The algorithm searches for the split (feature and value) that maximizes this Gain. A split is only performed if the Gain is positive (and greater than  $\gamma$ ).

Steps

- ① Construct a base Model
- ② Construct a Decision Tree with root.
- ③ Calculate Similarity Weight  

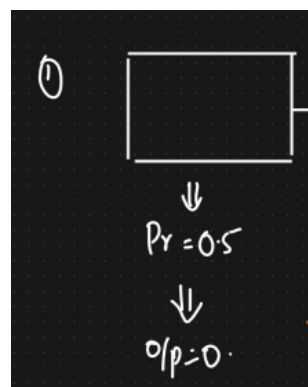
$$= \frac{\sum (\text{Residual})^2}{\sum p + (1-p) + \lambda}$$
- ④ Calculate Gain

One value  $\leftarrow$   $\sum p + (1-p) + \lambda$  ↓  
hyperparameter

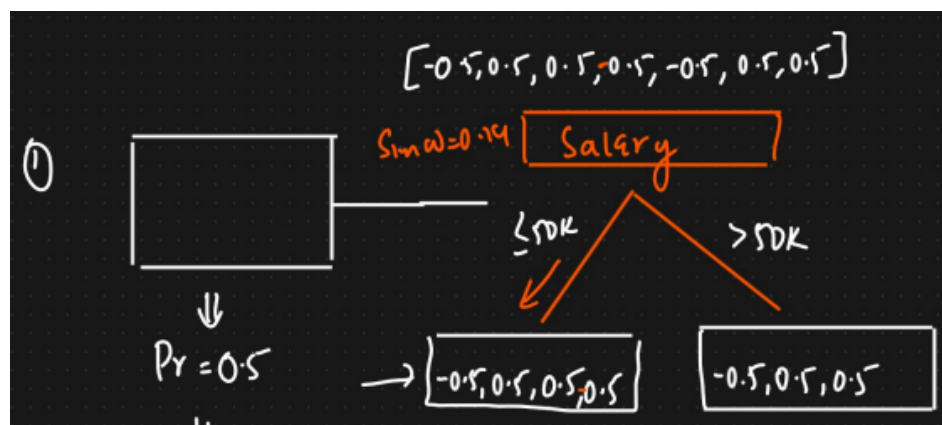
$y = 0.5$

Dataset		
<u>Salary</u>	<u>Credit</u>	<u>y</u> Approval
$\leq 50K$	B	0
$\leq 50K$	G	1
$\leq 50K$	G	1
$> 50K$	B	0
$> 50K$	G	1
$> 50K$	N	1
$\leq 50K$	N	0

Step 1-



<u>Salary</u>	<u>Credit</u>	<u>Approval</u>	<u>R1</u>
$\leq 50K$	B	0	-0.5
$\leq 50K$	G	1	0.5
$\leq 50K$	G	1	0.5
$> 50K$	B	0	-0.5
$> 50K$	G	1	0.5
$> 50K$	N	1	0.5
$\leq 50K$	N	0	-0.5



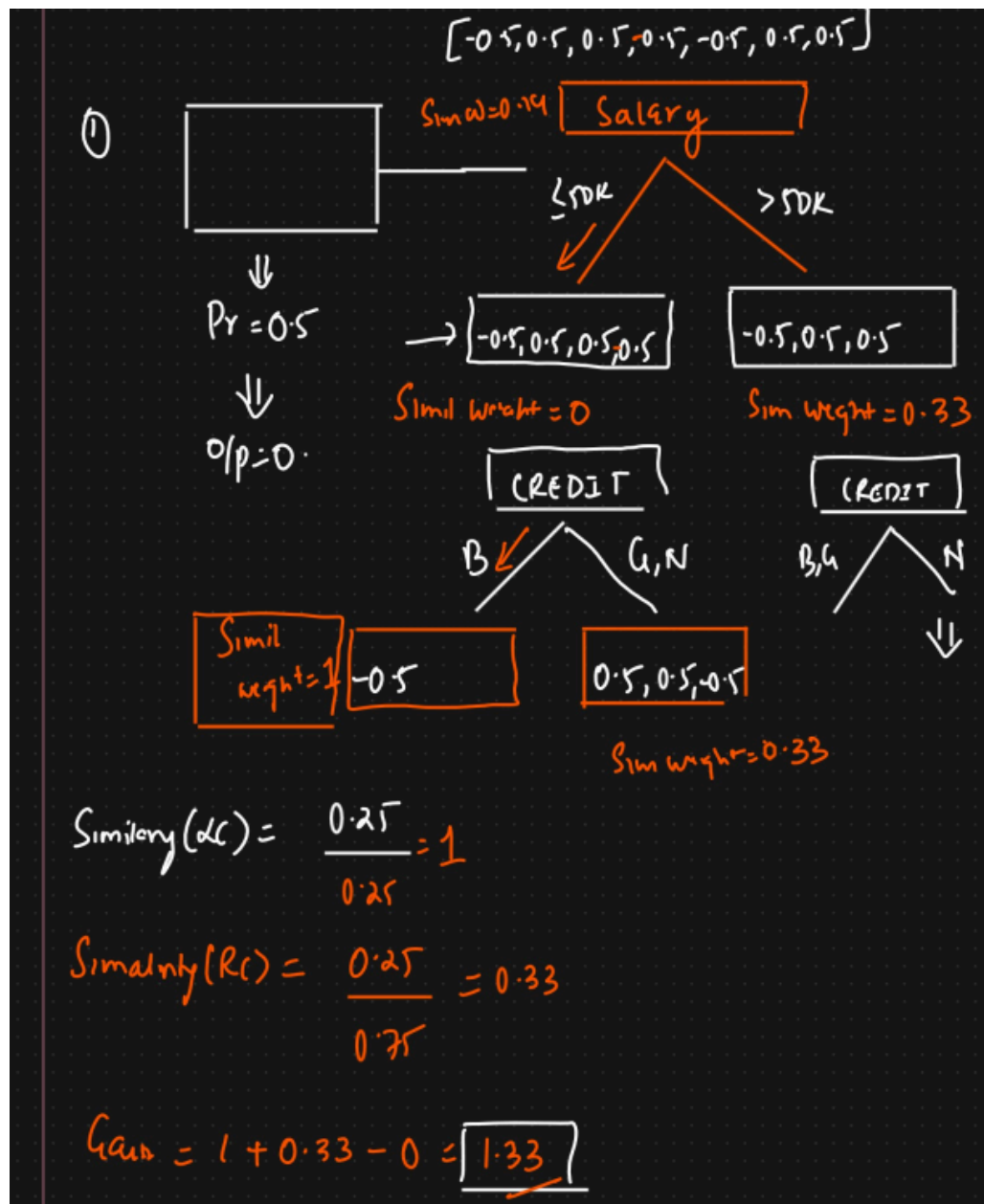
$$\text{Similarity weight } (d_c) = \frac{\sum (\text{Residual})^2}{\sum p_r (1-p_r)}$$

$$= \left[ (-\cancel{0.5} + \cancel{0.5} + \cancel{.5} - \cancel{0.5})^2 \right] = 0$$

$$1 \leq \left[ 0.5(1-.5) + 0.5(1-.5) + .5(1-.5) + 0.5(1-.5) \right]$$

$$\text{Gain} = 0 + 0.33 - .14 = .21$$

$$\text{Similarity weight } (RC) = \frac{(\cancel{.5} + \cancel{0.5} + 0.5)^2}{0.75} = \frac{0.25}{0.75} = \frac{1}{3}$$



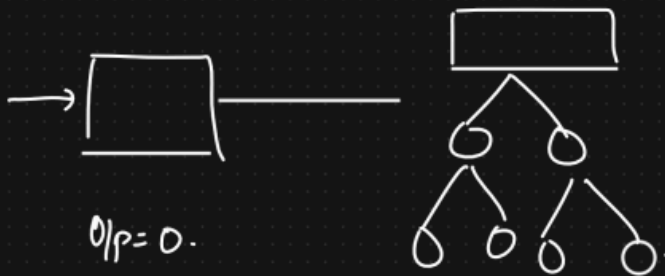
Test data

$$\log(\text{odds}) = \log\left(\frac{p}{1-p}\right)$$

$$\log(\text{odds}) = \log\left(\frac{0.5}{0.5}\right)$$

$$= 0$$

Final o/p



$$\text{New Data} \rightarrow = \sigma \left( 0 + \alpha (1) \right) \quad \alpha = 0.1$$

↓ ↑ Similarity weight

Logistic function  $\Leftarrow$  Sigmoid Activation

$$= \sigma \left( 0 + (0.1)(1) \right)$$

$$= \frac{1}{1 + e^{-0.1}} = \underline{\underline{0.52}}$$

Second Record

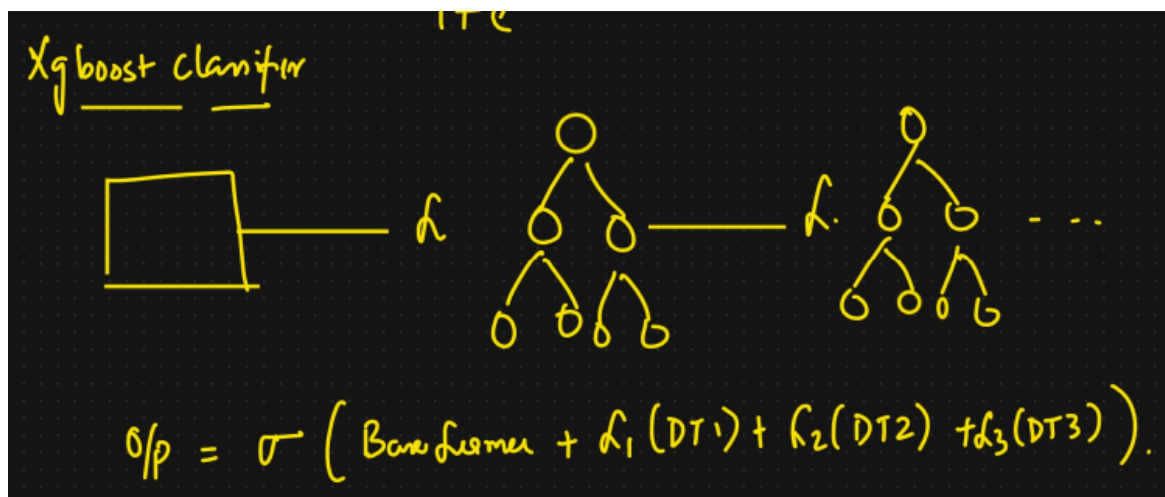
$$o/p = \sigma \left( 0 + \alpha (0.33) \right)$$

$$= \sigma \left( 0 + 0.1(0.33) \right)$$

$$= \frac{1}{1 + e^{-0.033}} = \underline{\underline{0.508}}$$



Salary	Credit	Approval	R1	y	R2
≤ 50K	B	0	-0.5	0.52	-0.48
≤ 50K	G	1	0.5	0.58	0.42
≤ 50K	G	1	0.5	—	—
> 50K	B	0	-0.5	—	—
> 50K	G	1	0.5	—	—
> 50K	N	1	0.5	—	—
≤ 50K	N	0	-0.5	—	—




## 6. Advantages of XGBoost

- **Speed & Performance:** Generally faster and more efficient than standard Gradient Boosting due to parallelization and optimizations.
- **Accuracy:** Often yields higher accuracy due to regularization and optimized algorithms.
- **Regularization:** Built-in L1/L2 regularization helps prevent overfitting.
- **Missing Value Handling:** Intrinsic ability to handle missing data simplifies preprocessing.
- **Flexibility:** Supports custom loss functions and evaluation metrics.

## 7. Disadvantages/Considerations

- **Hyperparameter Tuning:** Still requires careful tuning of numerous hyperparameters, which can be complex.
- **Interpretability:** Like other tree ensembles, can be harder to interpret than simpler models.
- **Computational Cost:** While optimized, can still be computationally intensive, especially on very large datasets or with many boosting rounds.
- **Potential for Bias:** Like any ML model, can learn and potentially amplify biases present in the training data.

Xgboost Regressor Mh Algorithm

Dataset {Regressor} → 

Exp	Gap	Salary	R <sub>1</sub>	$\hat{y}$	R <sub>2</sub>
→ 2	Yes	40K	-11	49.9	-9.9
→ 2.5	Yes	42K	-9	49.9	-7.9
→ 3	No	52K	1	51.5	0.5
4	No	60K	9	51.5	8.5
4.5	Yes	62K	11	52.1	9.9

≈ 51K

Similarity weight =  $\frac{\sum (\text{Residual})^2}{\sum p_r(1-p_r)}$

Gain

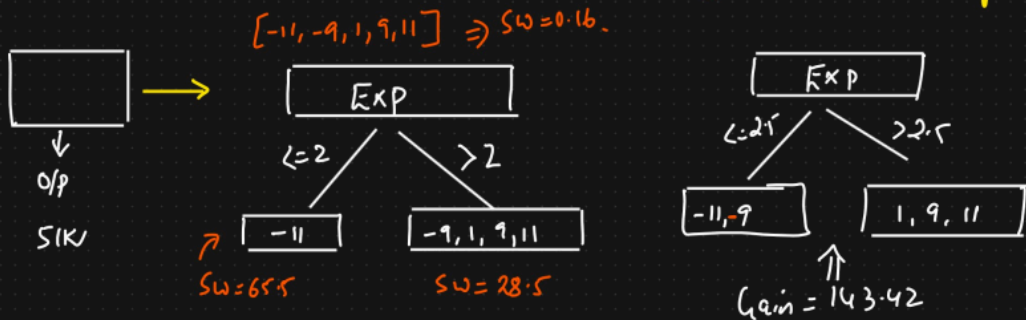
$[51 + (0.1)(-10)] = 51 - 0.1 = 49.9$

$[51 + (0.1)(5)] = 51.5$

$[51 + 0.1(11)] = 51 + 1.1 = 52.1$

## Steps

- ① Create a Base Model    ② Residual Computation    ③ Construct DT1 using  $\{x_i, R_i\}$



$$\text{Similarity Weight} = \frac{\sum (\text{Residual})^2}{\lambda + 1}$$

$$\lambda = 1$$

No. of Residuals +  $\lambda \rightarrow$  Hyperparameter

$$SW(\text{Left child}) = \frac{121}{1+1}$$

$$= 121/2 = 60.5$$

$$\boxed{\lambda \uparrow \quad SW \downarrow}$$

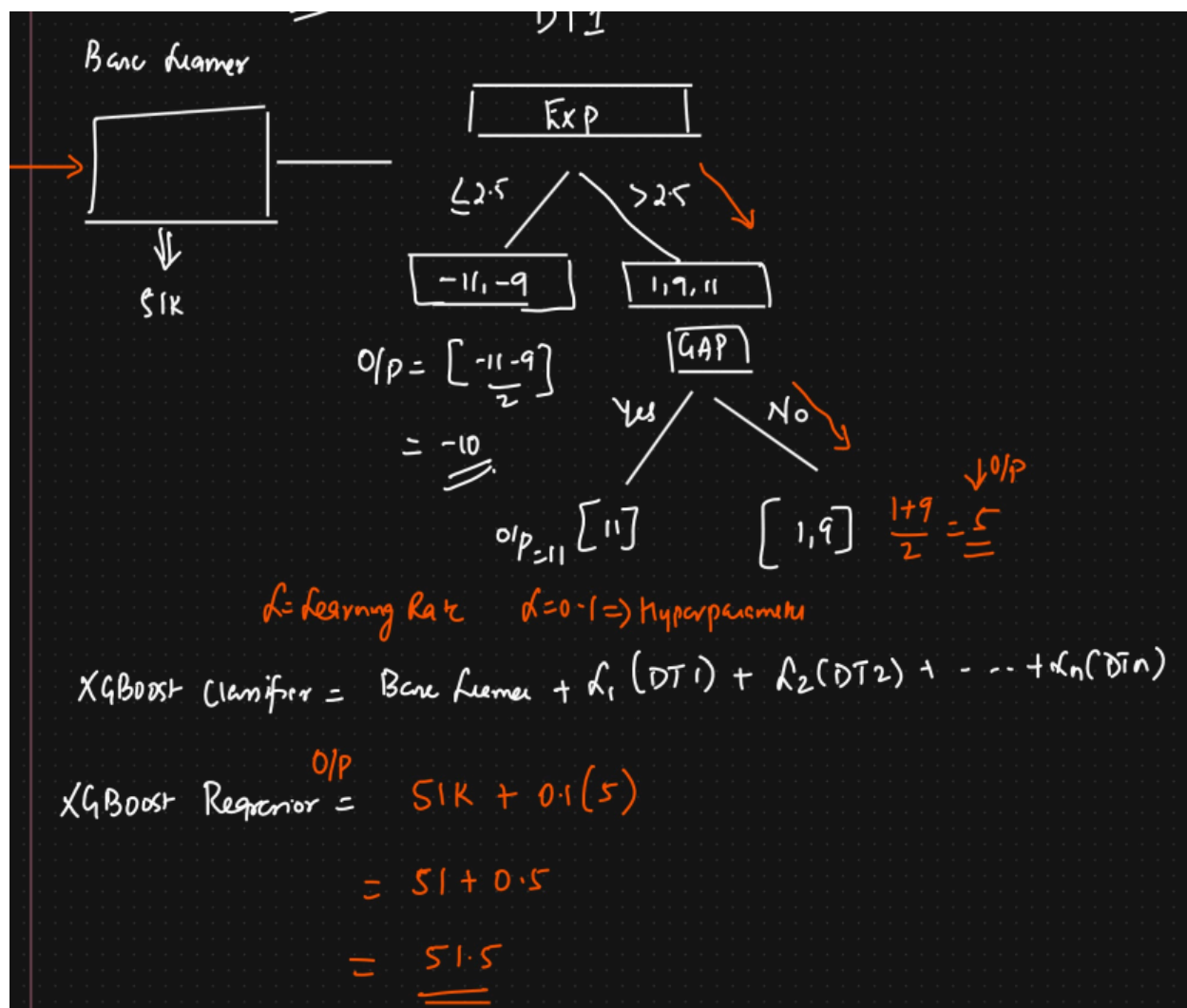
$$SW(\text{Right child}) = \frac{(-9+1+9+11)}{4+1}$$

$$= \frac{144}{5} = 28.8$$

## ⑤ Calculate Gain

$$\text{Gain} = 60.5 + 28.8 - 0.16$$

$$= 89.14$$



$$\text{Similarity weight} = \frac{\sum (\text{Residual})^2}{\text{No. of Residuals} + \lambda}$$

{ Regression }

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