Boosting Algorithms

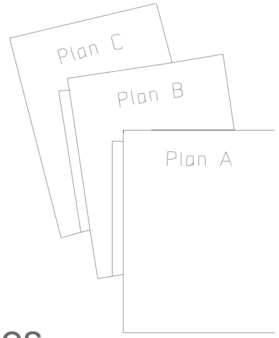
서울대학교 컴퓨터공학부 이 영 기

Human-Centered Computer Systems Lab

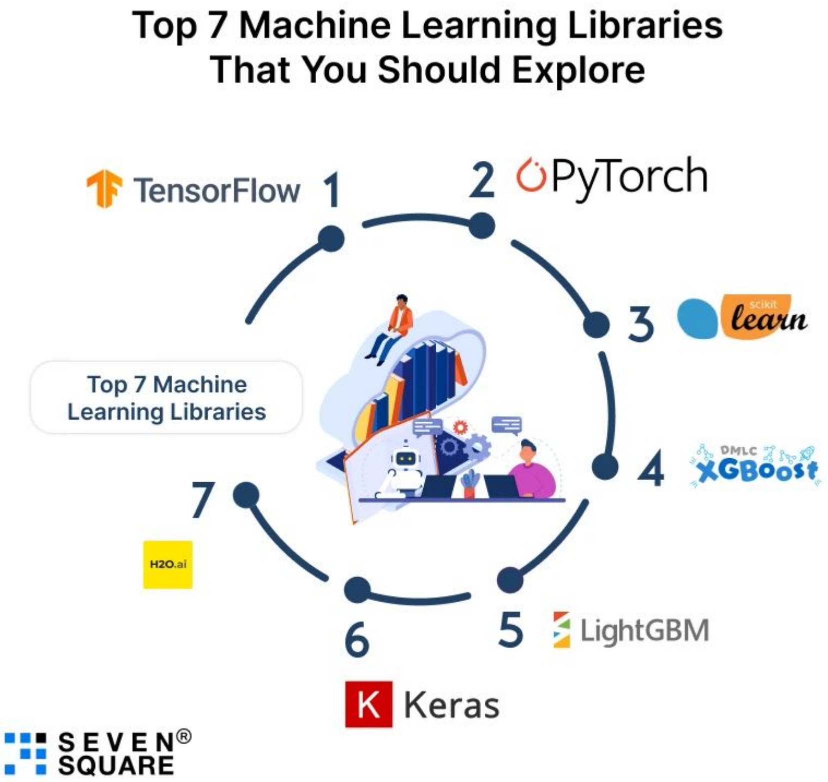
SEOUL NATIONAL UNIVERSITY

Overview

- Objective
  - Understand boosting algorithms
- Content
  - GBM(Gradient Boosting Machine)
  - XGBoost(eXtreme Gradient Boost)
  - LightGBM(Light Gradient-Boosting Machine)
- After this module, you should be able to
  - Understand various types of boosting and their advantages and disadvantages



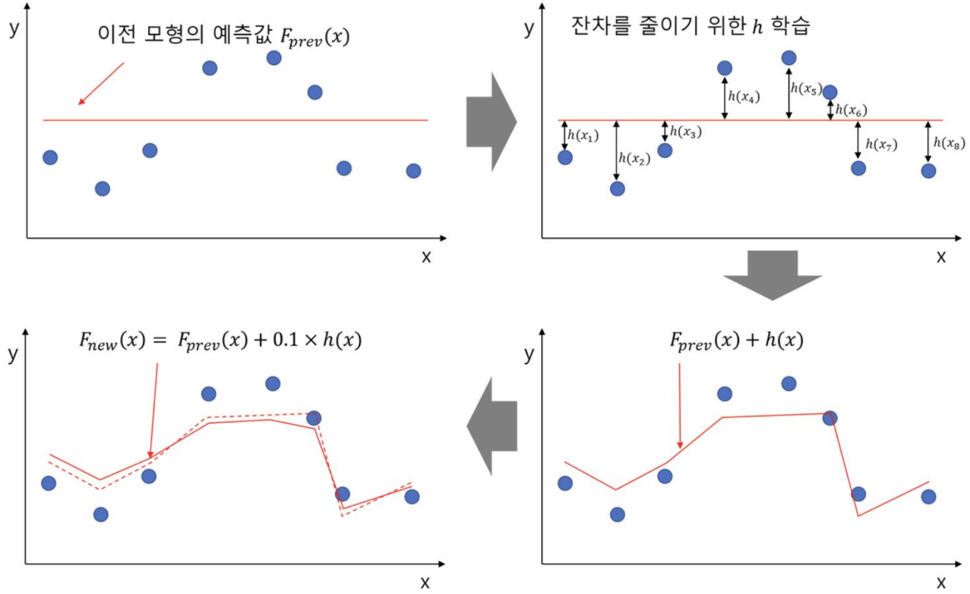
Introduction



https://www.sevensquaretech.com/top-machine-learning-libraries-in-2025/

GBM(Gradient Boosting Machine)

- A **sequential** ensemble technique that builds strong predictive models by iteratively adding weak learners to correct the *residuals* of previous steps



$$F_m(x) = h_0(x) + l h_1(x) + \cdots + l h_m(x)$$

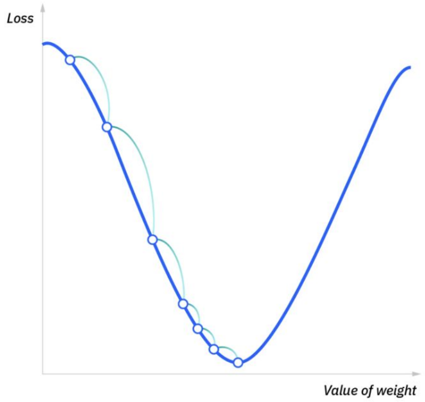
GBM(Gradient Boosting Machine)

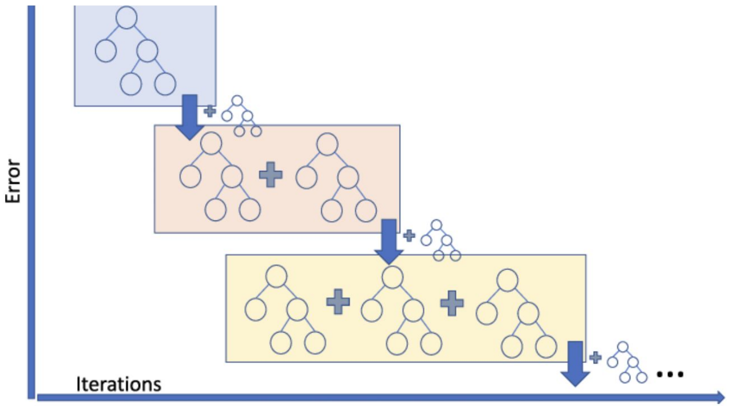
(-) ***residual***

- Gradient serves as a mathematical generalization of the *residual*

$$L(y_i, f(x_i)) = \frac{1}{2}(y_i - F(x_i))^2 \qquad \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial[\frac{1}{2}(y_i - F(x_i))^2]}{\partial F(x_i)} = -(y_i - F(x_i))$$

- It performs **Gradient Descent**, following the steepest descent direction to minimize the overall loss function
- However, it is slow(working sequentially) and vulnerable to overfitting

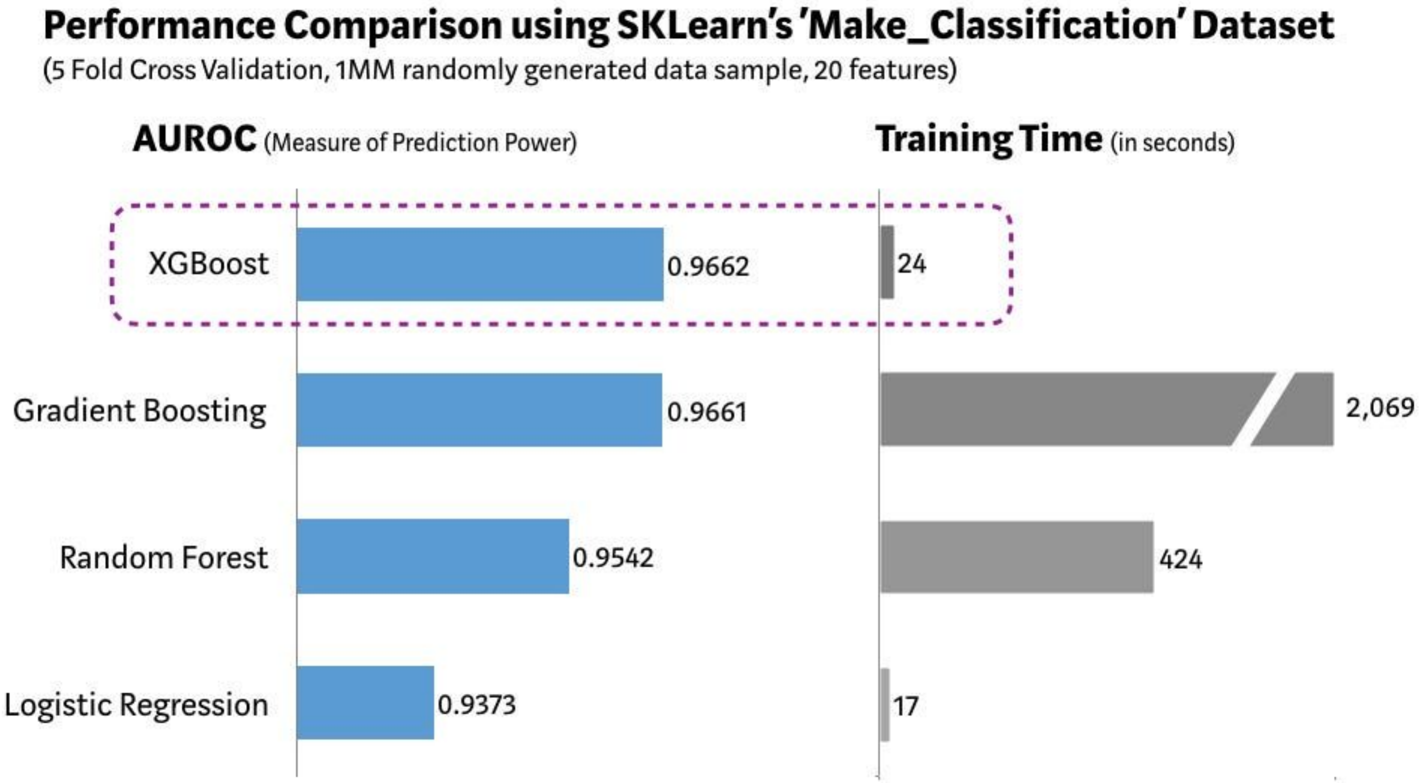




XGBoost(eXtreme Gradient Boost)

- A highly **scalable** and **regularized** gradient boosting framework to maximize computational speed and predictive accuracy through 2nd-order optimization
- Advantages
  - **Accuracy**: Uses 2nd-order Taylor Expansion (Hessian) for more accurate convergence compared to GBM's 1st-order gradients
  - **Regularization**: Prevents the model from becoming overfitting by using L2-Regularization
  - **Speed**: Maximizes speed by parallelizing split finding via pre-sorted column blocks and hardware-aware optimizations

XGBoost(eXtreme Gradient Boost)

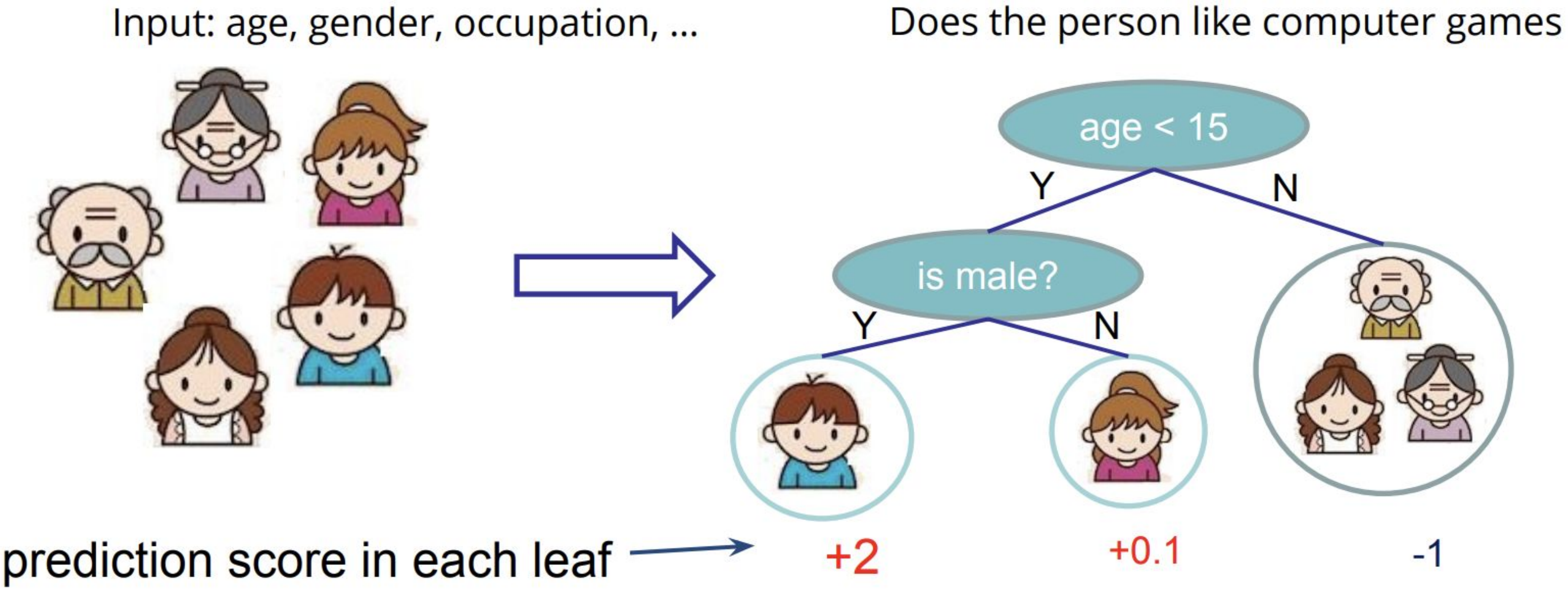


CART(Classification And Regression Tree) Model

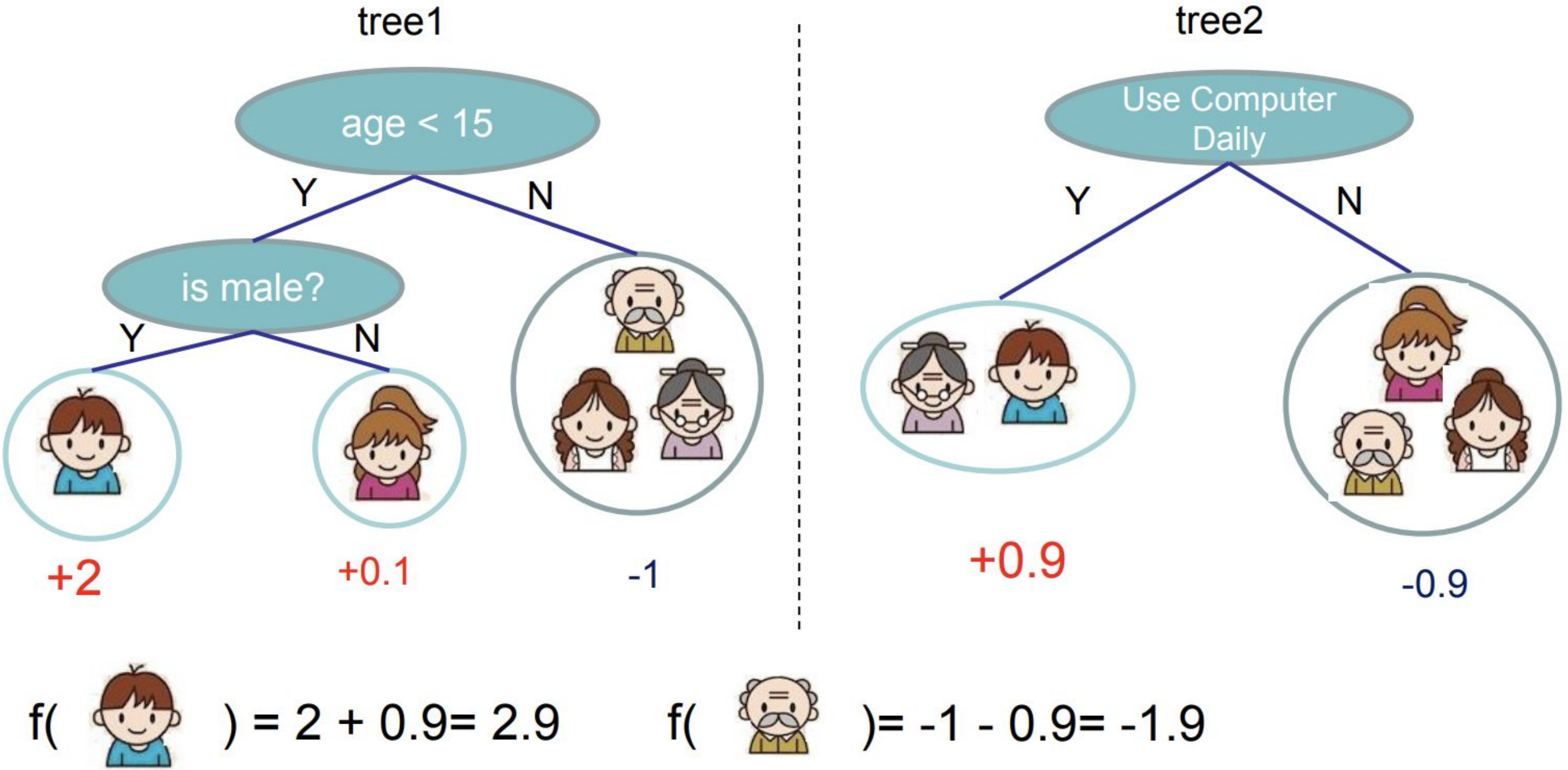
- A **binary decision tree** that assigns a real-valued score (w) to each leaf node, regardless of whether the task is classification or regression
- Key Attributes: Binary Splitting and Numerical Leaf Scores
- Essential for **Additive Learning**: These numerical scores act as correction values that are summed across all trees to form the final ensemble prediction

$$Y' = a \times \text{tree A} + b \times \text{tree B} + c \times \text{tree C} + \dots$$
$$\text{where } a, b, c, \cdots = \text{weights for each tree}$$

CART(Classification And Regression Tree) Model



CART(Classification And Regression Tree) Model



XGBoost in a Nutshell

- Input: $n \times m$ (n samples, m features) $X = \{(\mathbf{x}_i, y_i)\}, \quad (|X| = n, \mathbf{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R})$
- Model: $\mathcal{F}$ (consists of B CART trees)

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{b=1}^{B} f_b(\mathbf{x}_i), f_b \in \mathcal{F}$$

$$\text{where } \mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\} \quad (q : \mathbb{R}^m \to |T|, w \in \mathbb{R}^{|T|})$$

- Loss Term

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_b \Omega(f_b), \text{ where } \Omega(f) = \gamma|T| + \frac{1}{2}\lambda\|w\|^2$$

- Training
    - Incremental branch expansion: Each tree grows by iteratively adding one branch at a time
    - Post-pruning effect: By evaluating the Gain against γ, XGBoost effectively prunes unnecessary branches that do not contribute significantly to the model's predictive power

XGBoost: Regularization

- In XGBoost loss, we can see the regularization term (Ω)

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_b \Omega(f_b), \ \text{where } \Omega(f) = \gamma|T| + \frac{1}{2}\lambda\|w\|^2$$

- Unlike traditional GBM, regularization penalizes the number of leaves ($|T|$) and the magnitude of leaf weights (w, represented as L2-norm) to prevent overfitting
- Setting γ and λ to 0 reverts XGBoost to the standard Gradient Tree Boosting

XGBoost: Algorithm Details

Goal: find f_b to minimize loss($\mathcal{L}^{(b)}$)

- Loss Function

$$\mathcal{L}^{(b)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(b-1)} + f_b(\mathbf{x}_i)) + \Omega(f_b) + \text{Constant}$$

- It is easy to express loss when it is MSE, but what if we want to use other loss?
    - (i) Function can be complicated, (ii) Library should recalculate when loss term changes
- In XGBoost, we use **second-order approximation** by Taylor Expansion

$$\mathcal{L}^{(b)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(b-1)}) + g_i f_b(\mathbf{x}_i) + \frac{1}{2} h_i f_b^2(\mathbf{x}_i)] + \Omega(f_t)$$

$$g_i = \frac{\partial}{\partial \hat{y}_i^{(b-1)}} l(y_i, \hat{y}_i^{(b-1)}), \quad h_i = \frac{\partial^2}{\partial (\hat{y}_i^{(b-1)})^2} l(y_i, \hat{y}_i^{(b-1)})$$

- Optimization depends only g_i & h_i, regardless of loss type, achieving better accuracy using Hessian (Newton's method)

XGBoost: Algorithm Details

- By applying the regularization term(Ω),

$$\tilde{\mathcal{L}}^{(b)} = \sum_{i=1}^{n}[g_i f_b(x_i) + \frac{1}{2}h_i f_b^2(x_i)] + \gamma|T| + \frac{1}{2}\lambda\sum_{j=1}^{|T|} w_j^2$$

- Using a new set $I_j = \{i|q(x_i) = j\}$

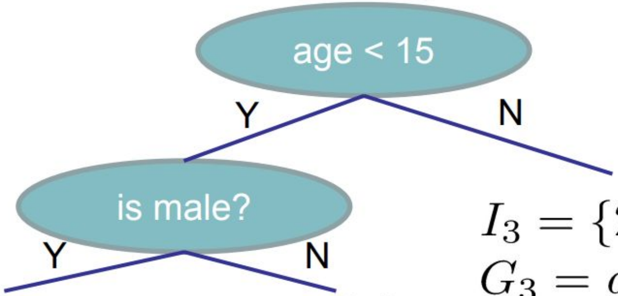
$$\tilde{\mathcal{L}}^{(b)} = \sum_{j=1}^{|T|}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma|T|$$

- By differentiating the objective function with respect to w_j, we obtain the optimal weight w_j^* that minimizes the loss for each leaf, and calculate the optimized loss

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda} \qquad \tilde{\mathcal{L}}^{(b)}(q) = -\frac{1}{2}\sum_{j=1}^{|T|}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma|T|$$

XGBoost: Example

Instance index	gradient statistics
1	g1, h1
2	g2, h2
3	g3, h3
4	g4, h4
5	g5, h5



$I_1 = \{1\}$
$G_1 = g_1$
$H_1 = h_1$

$I_2 = \{4\}$
$G_2 = g_4$
$H_4 = h_4$

$I_3 = \{2, 3, 5\}$
$G_3 = g_2 + g_3 + g_5$
$H_3 = h_2 + h_3 + h_5$

$$Obj = -\sum_j \frac{G_j^2}{H_j+\lambda} + 3\gamma$$

The smaller the score is, the better the structure is

XGBoost: Greedy Update

- Evaluating the gain of a potential split
  - Compare the loss(org - split) when split happens: $I = I_L \cup I_R$

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma$$

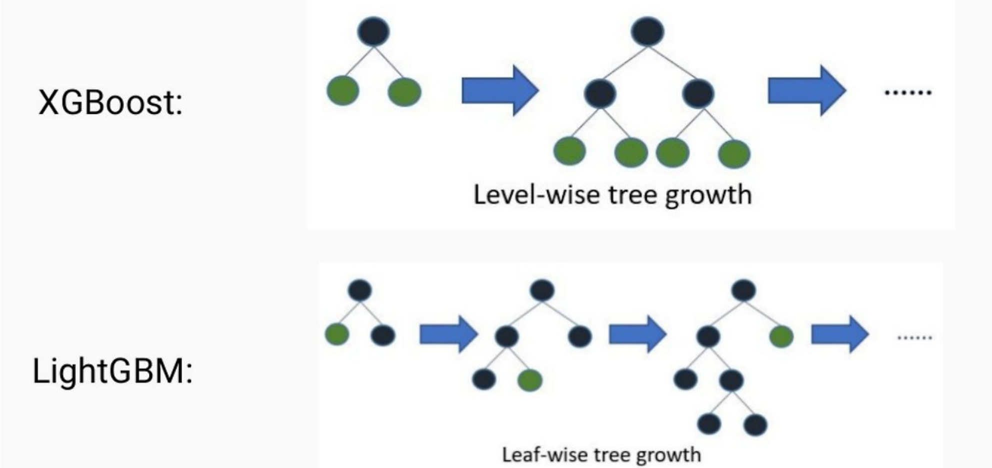
- If $\mathcal{L}_{split} > 0$, splitting benefits the optimization -> Keep splitting
- γ, λ: regularization parameter
  - If they get larger, $\mathcal{L}_{split} < 0$
  - Prevents overfitting(complex tree)

LightGBM: Overcoming the Scalability Bottleneck

- The most time-consuming part of GBM is **finding the optimal split point**, which typically requires scanning every data point (N) for every feature (D)
  - $O(N \times D)$ complexity becomes a massive burden for massive dataset
- Reducing N: GOSS (Gradient-based One-Side Sampling)
  - Prioritize high-gradient instances as they represent **under-fitted data** with the most information gain
- Reducing D: EFB (Exclusive Feature Bundling)
  - **Bundle mutually exclusive** sparse features(rarely non-zero simultaneously) into single dense feature

LightGBM: Leaf-wise Tree Growth

- LightGBM grows the tree by **splitting the leaf node** that results in the greatest loss reduction(max gain), regardless of its current depth
  - Efficient(achieve lower loss faster), but forms asymmetric tree(overfitting)
- Tuning parameters
  - *`num_leaves, max_depth, min_data_in_leaf`*



LightGBM vs. XGBoost

- LightGBM achieves target accuracy **much faster** than other GBM algorithms