GBM Family

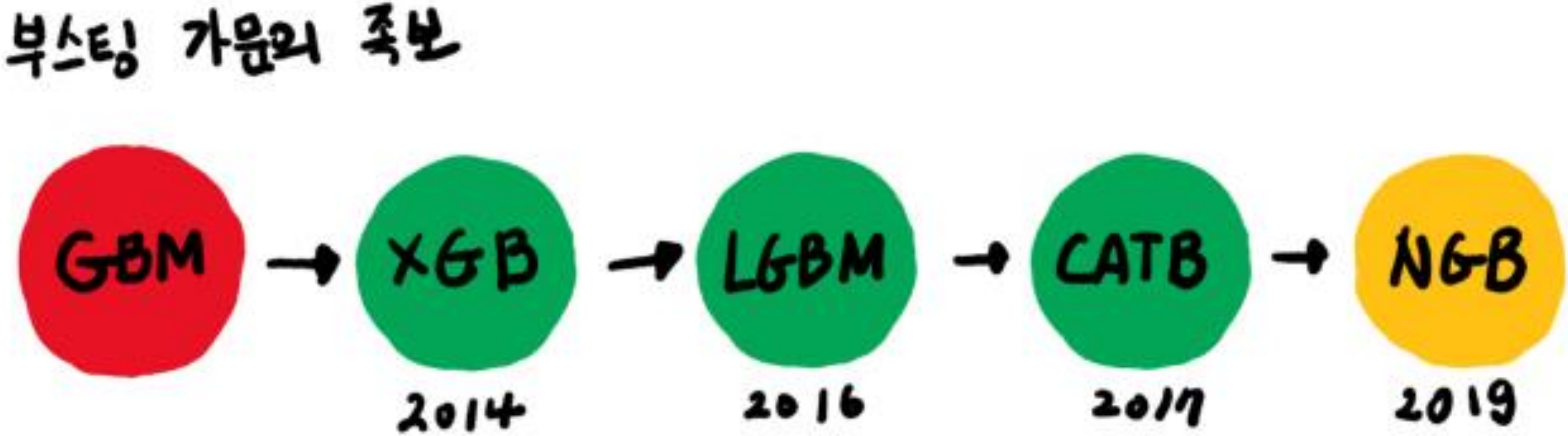
Data Scientist
안건이

목차

- Boosting
  - ~~AdaBoost~~
  - Gradient Boosting Machine (GBM)
  - XGBoost
  - LightGBM

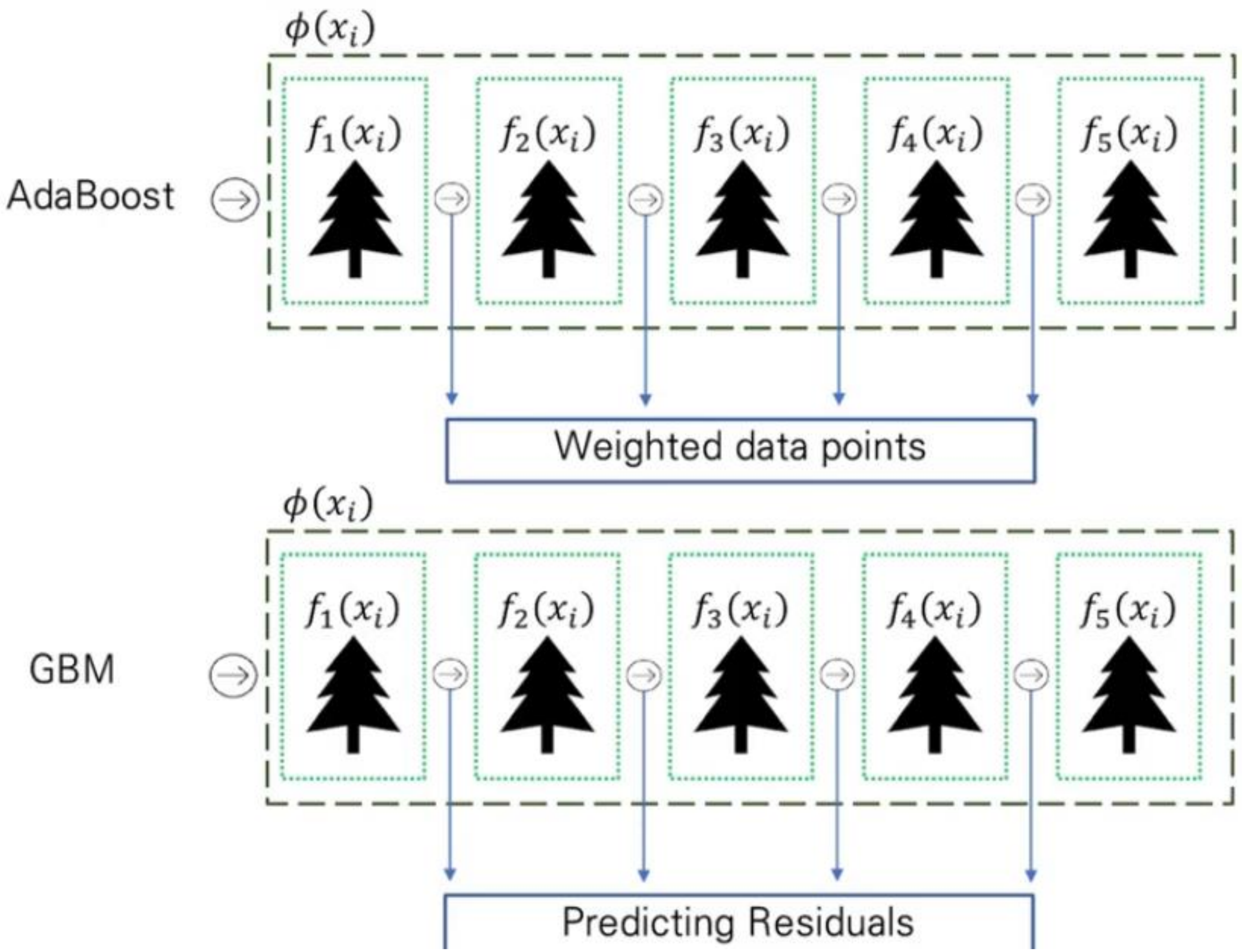
- 데이터 실습

GBM Family



AdaBoost vs Gradient Boosting Machine

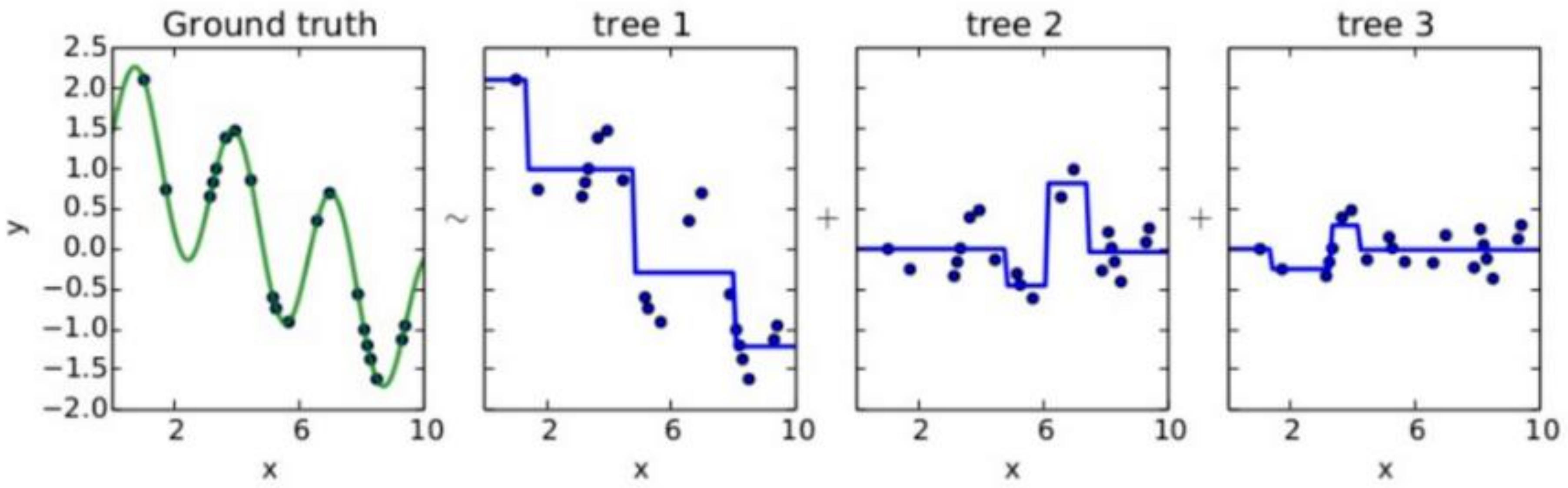
- GBM : Gradient Boosting Machine
  - Adaboost : 하나의 Tree에서 발생한 Error가 다음 Tree에 영향을 줌
    - 여러 Tree가 순차적으로 연결되어 최종 결과를 도출함



잔차를 학습한다

Gradient Boosting Machine

- GBM : Gradient Boosting Machine
  - Adaboost : 하나의 Tree에서 발생한 Error가 다음 Tree에 영향을 줌
    - 여러 Tree가 순차적으로 연결되어 최종 결과를 도출함



$$j(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$$

$$\frac{\partial j(y_i, f(x_i))}{\partial f(x_i)} = \frac{\partial\left[\frac{1}{2}(y_i - f(x_i))^2\right]}{\partial f(x_i)} = f(x_i) - y_i$$

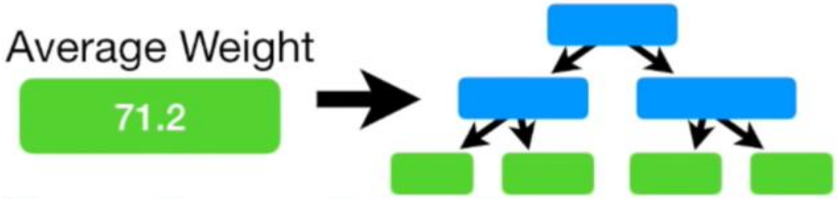
Gradient Boosting Machine

- GBM : Gradient Boosting Machine
  - Step1
  - Tree가 아닌 하나의 leaf (Single leaf) 부터 시작함 → 이 leaf는 Target 값에 대한 초기 추정 값을 나타냄
    - GBM은 single leaf 부터 시작하며, 그 single leaf 모델이 예측하는 Target 값 추정 값은 모든 Target의 평균

Step 1

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

Single leaf value : (88+76+56+73+77+57)/6 = 71.2

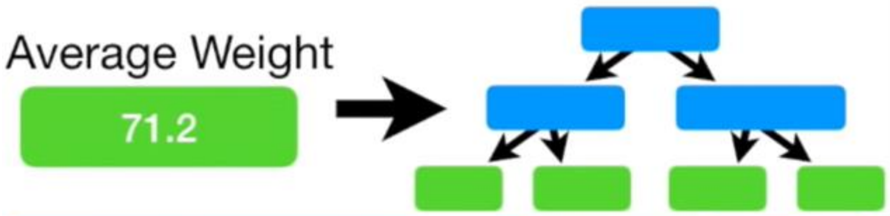


Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

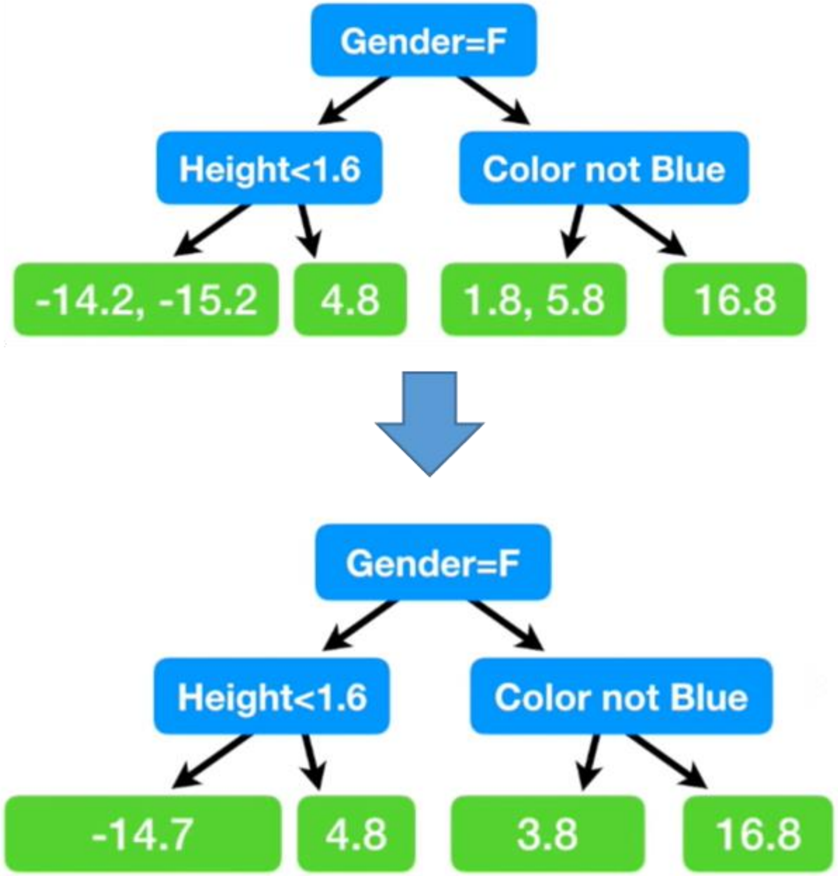
Gradient Boosting Machine

- GBM : Gradient Boosting Machine
  - Step 2
  - Residual(잔차)을 예측하는 Tree를 학습함
  - Terminal Node에 두 개 이상의 Residual 값이 있는 경우 평균으로 치환해서 넣어주게 됨

Step 2



Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

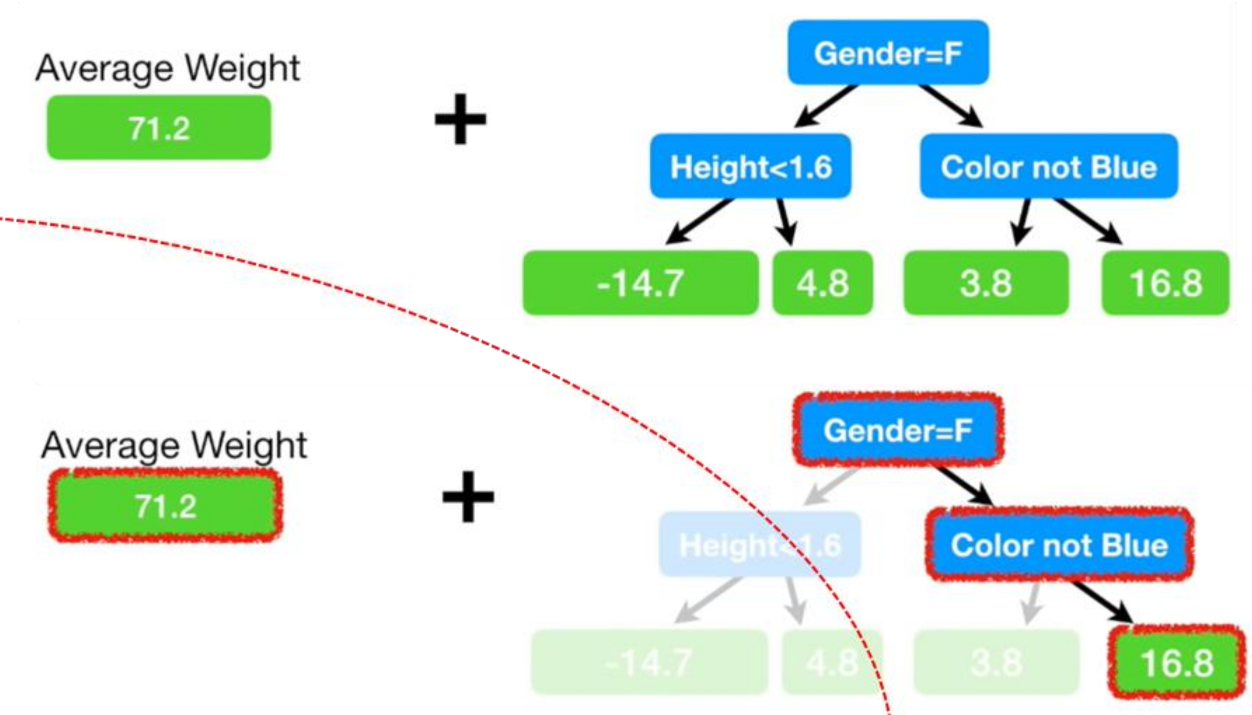


Gradient Boosting Machine

- GBM : Gradient Boosting Machine
  - Step 3
  - Average Weight (Single leaf) + Predicted Residual

Step 3

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

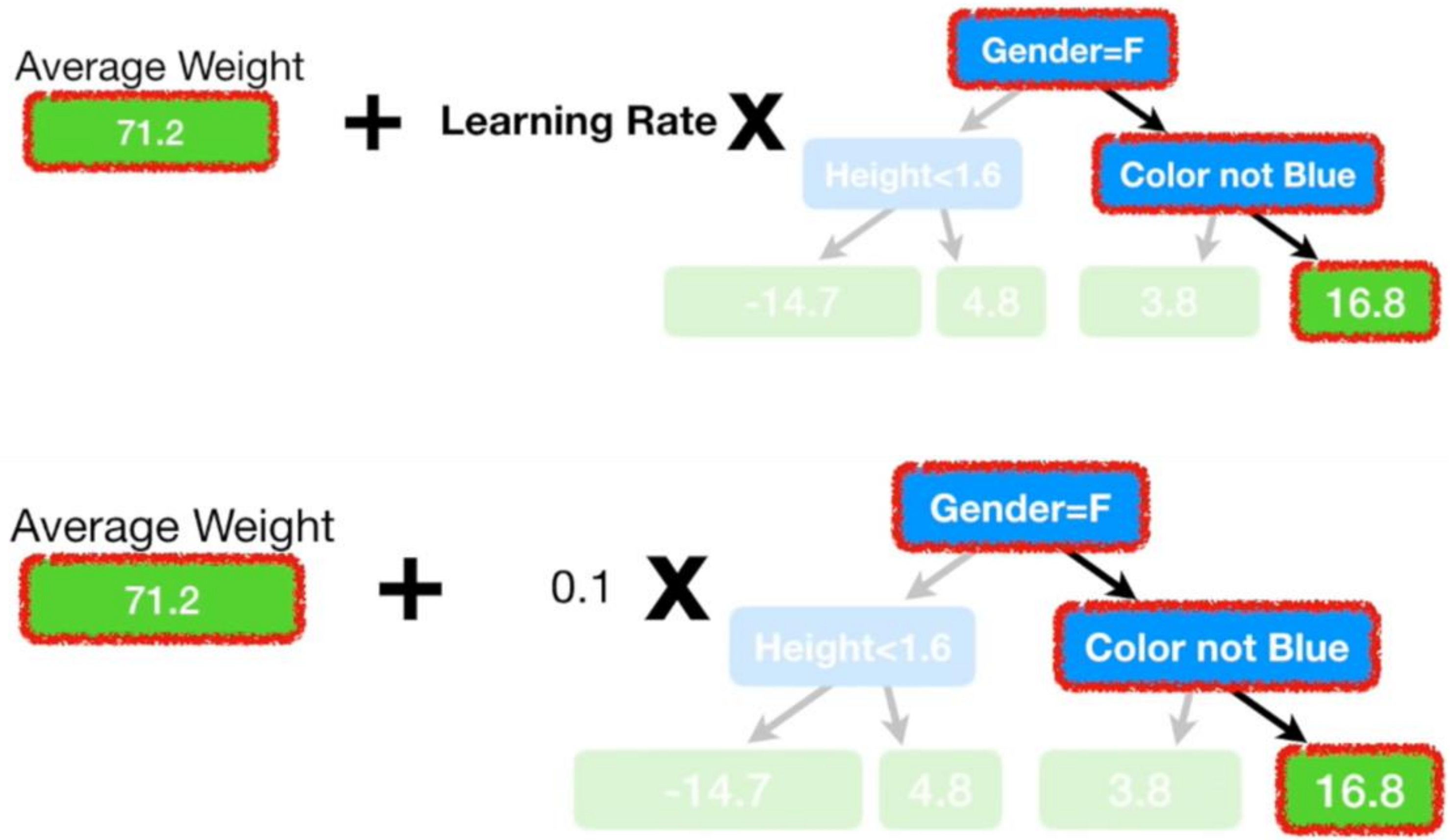


…so the **Predicted Weight** = 71.2 + 16.8 = 88

Gradient Boosting Machine

- GBM : Gradient Boosting Machine
  - Step 4
  - Overfitting을 방지하기 위해 Learning Rate을 사용함
  - Learning Rate = 0 ~ 1

Step 4

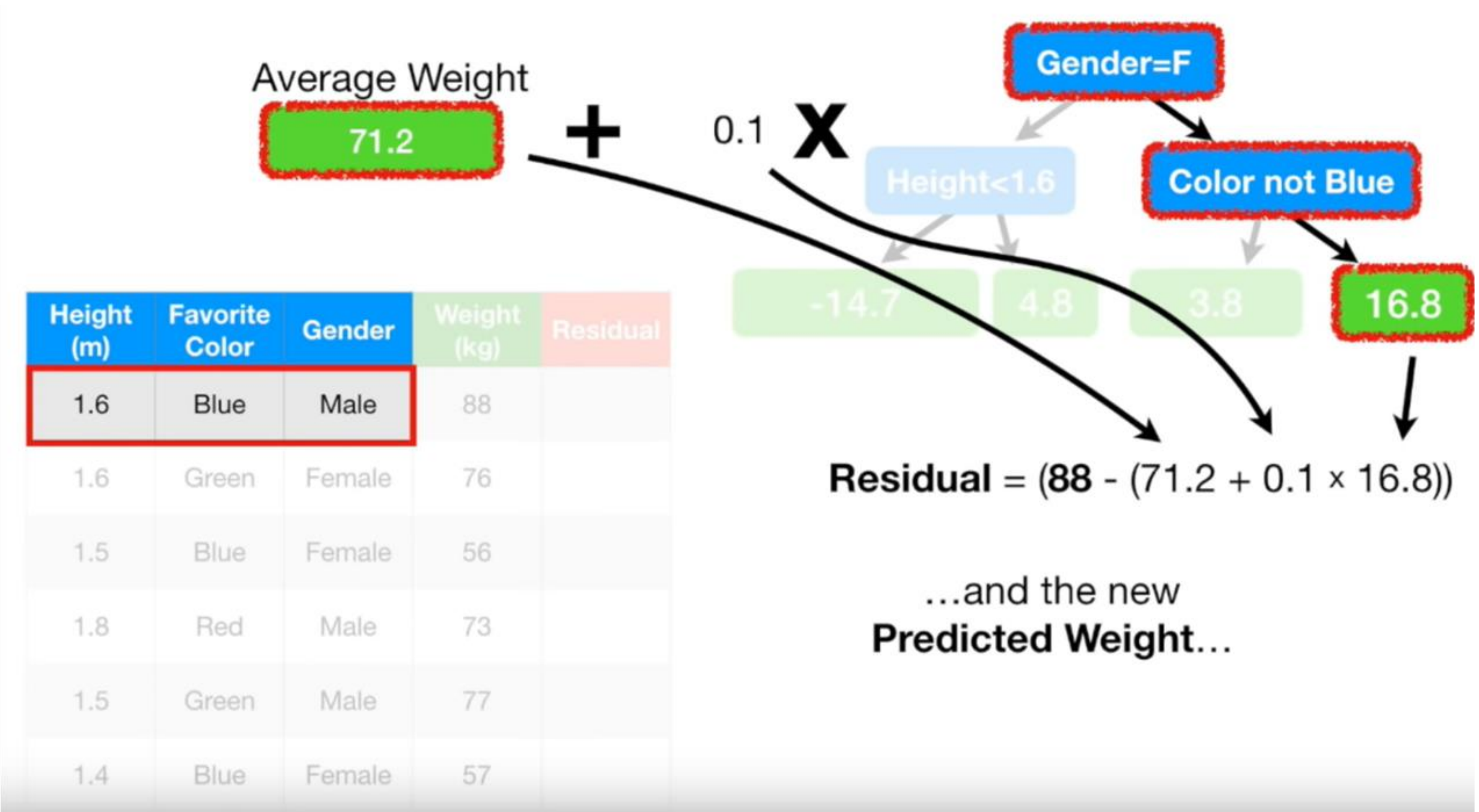


Now the **Predicted Weight** = 71.2 + (0.1 × 16.8) = 72.9

Gradient Boosting Machine

- GBM : Gradient Boosting Machine
  - Step 4
  - Overfitting을 방지하기 위해 Learning Rate을 사용함
  - Learning Rate = 0 ~ 1

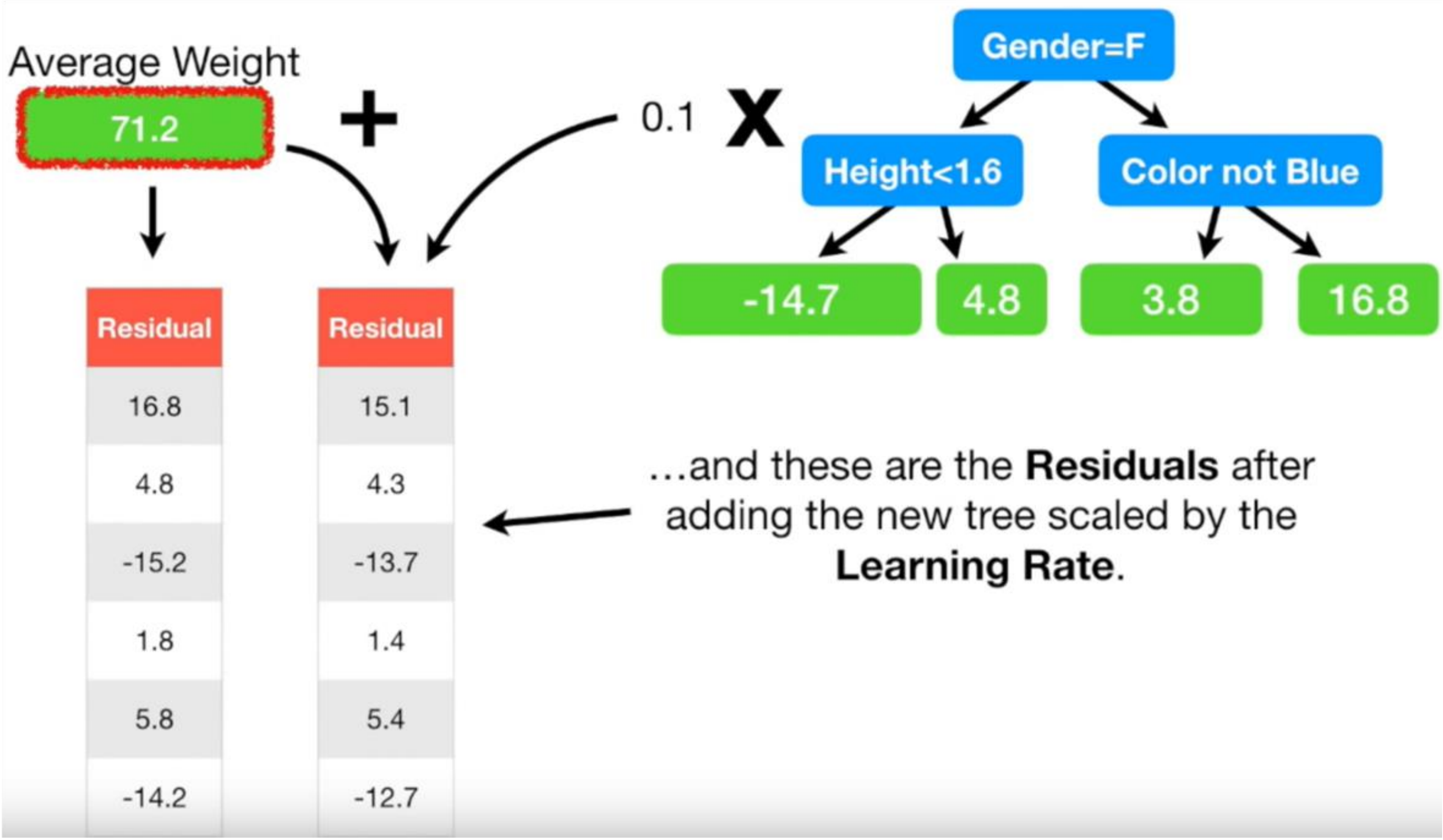
Step 4



Gradient Boosting Machine

- GBM : Gradient Boosting Machine
  - Step 5
  - Residual Update

Step 5



Gradient Boosting Machine

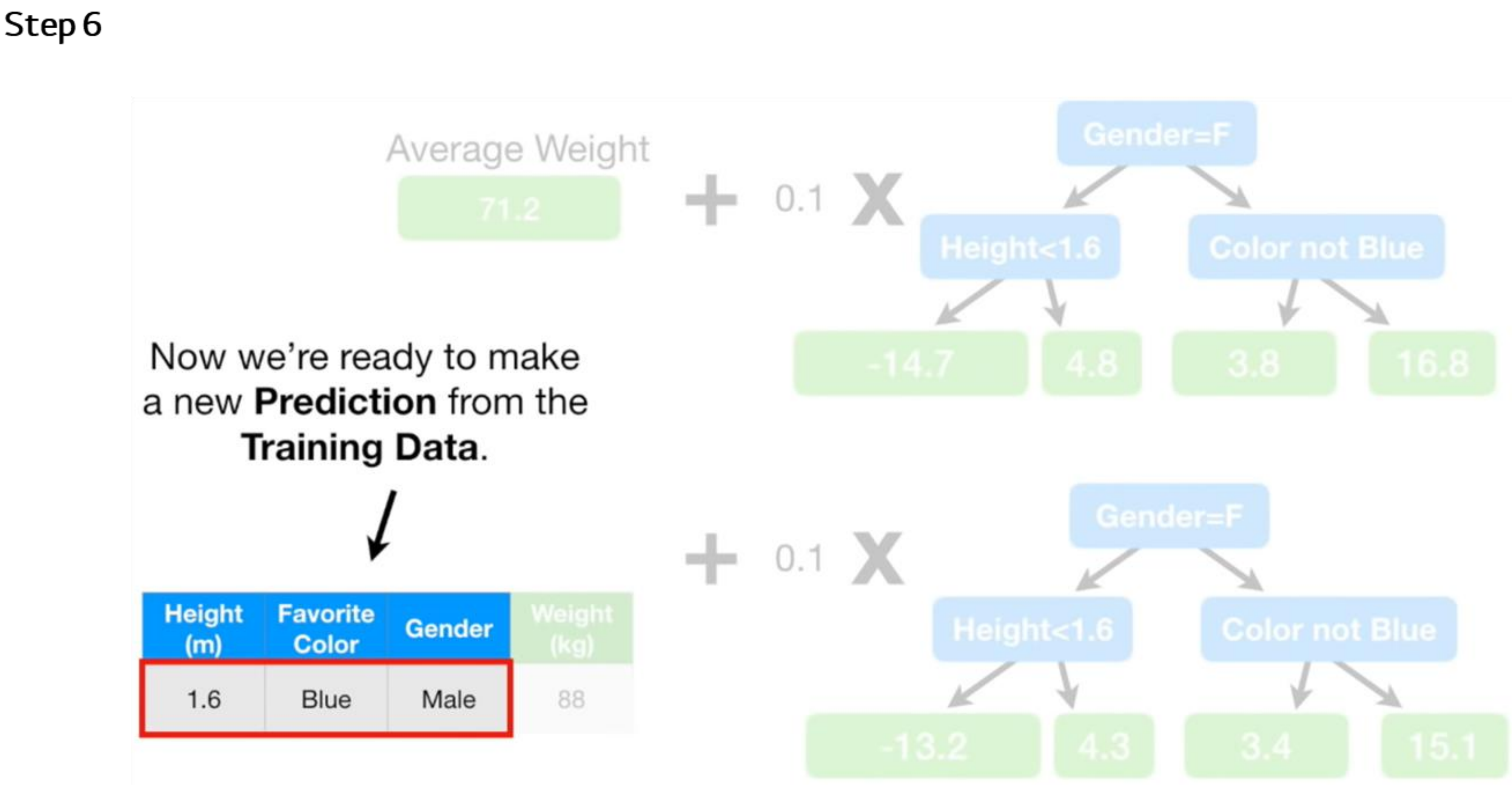
- GBM : Gradient Boosting Machine
  - Step 6
  - Set up New Tree

Step 6

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7

Gradient Boosting Machine

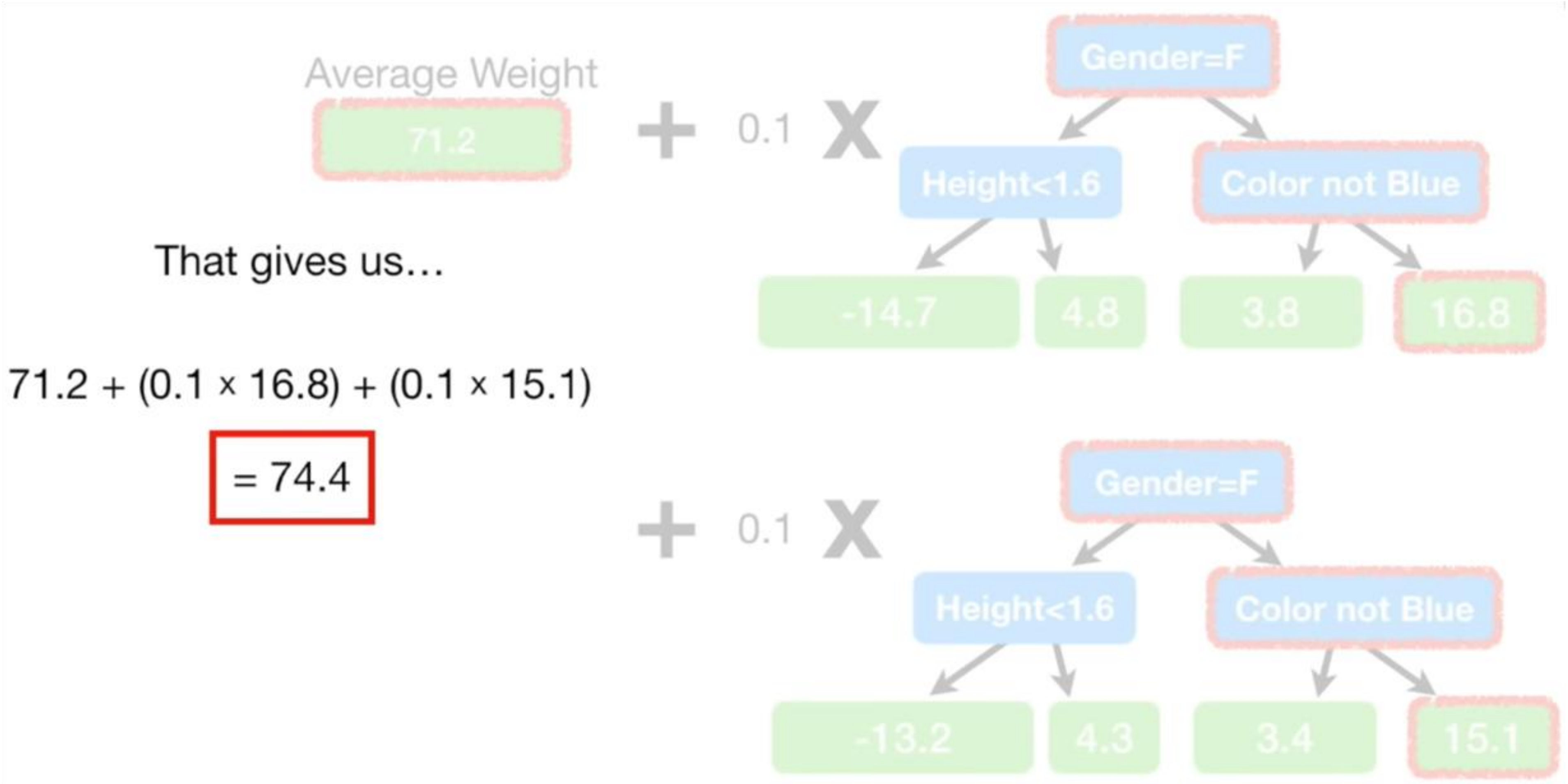
- GBM : Gradient Boosting Machine
  - Step 6
  - Set up New Tree



Gradient Boosting Machine

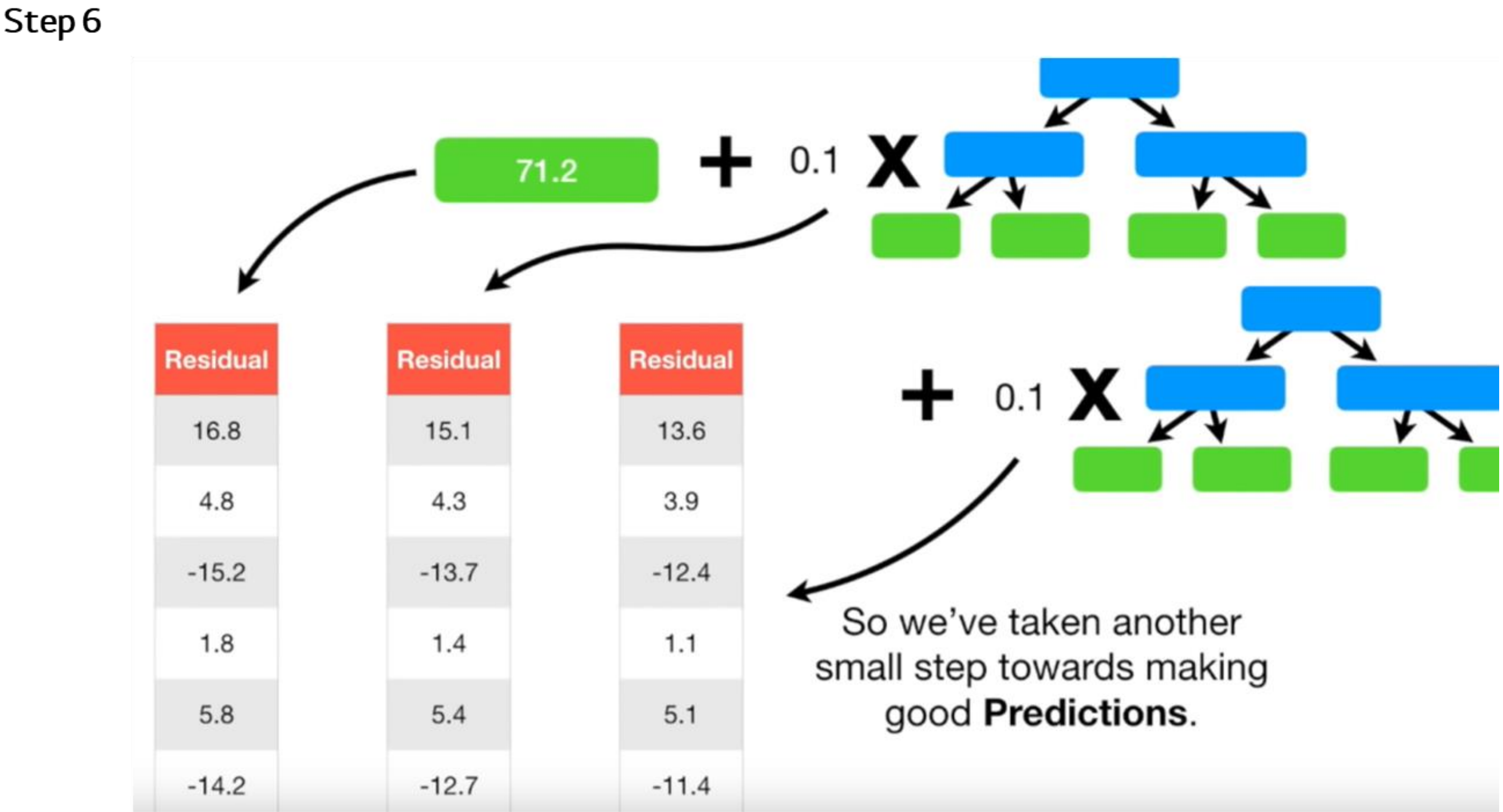
- GBM : Gradient Boosting Machine
  - Step 6
  - Set up New Tree

Step 6



Gradient Boosting Machine

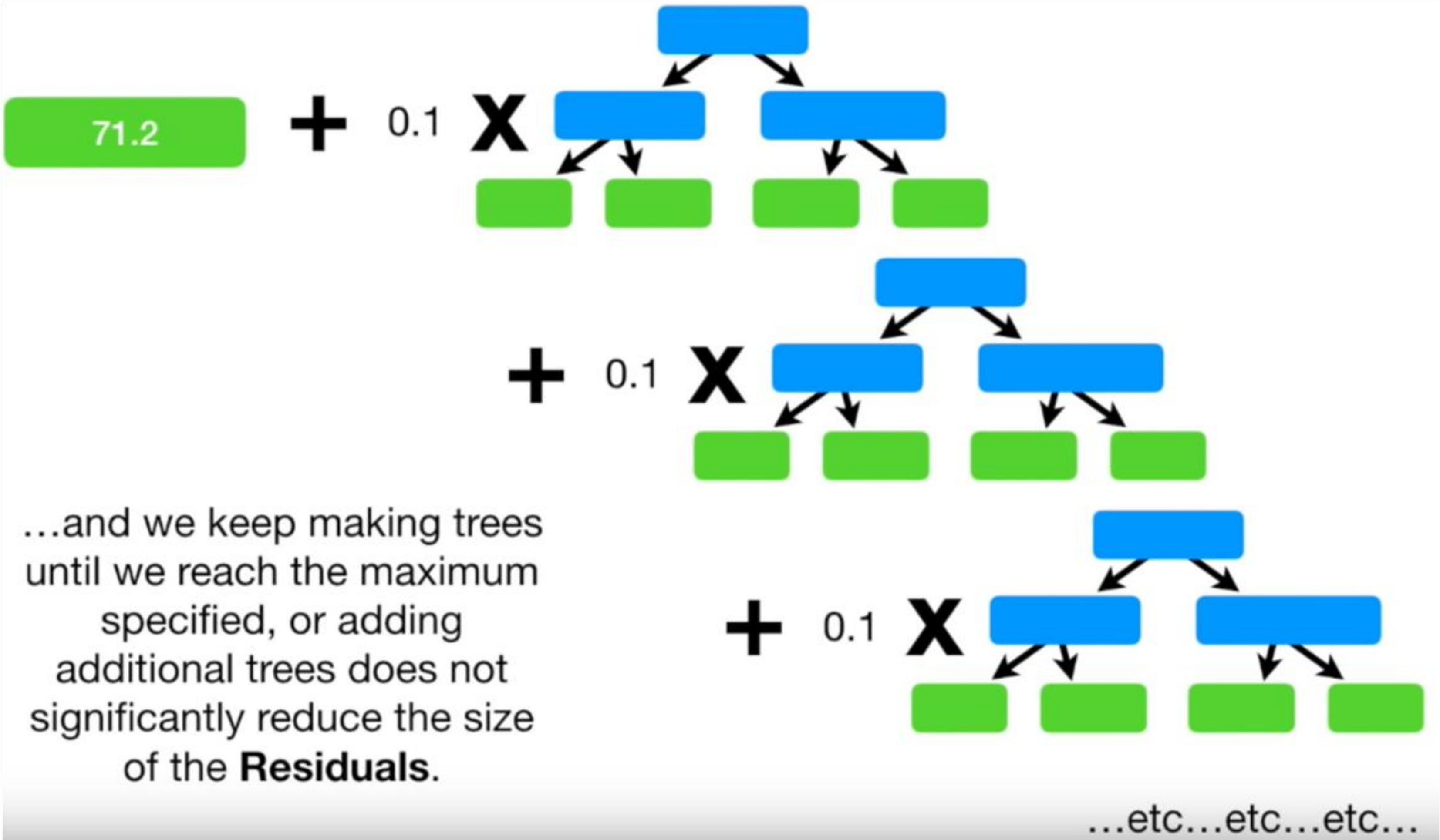
- GBM : Gradient Boosting Machine
  - Step 6
  - Set up New Tree



Gradient Boosting Machine

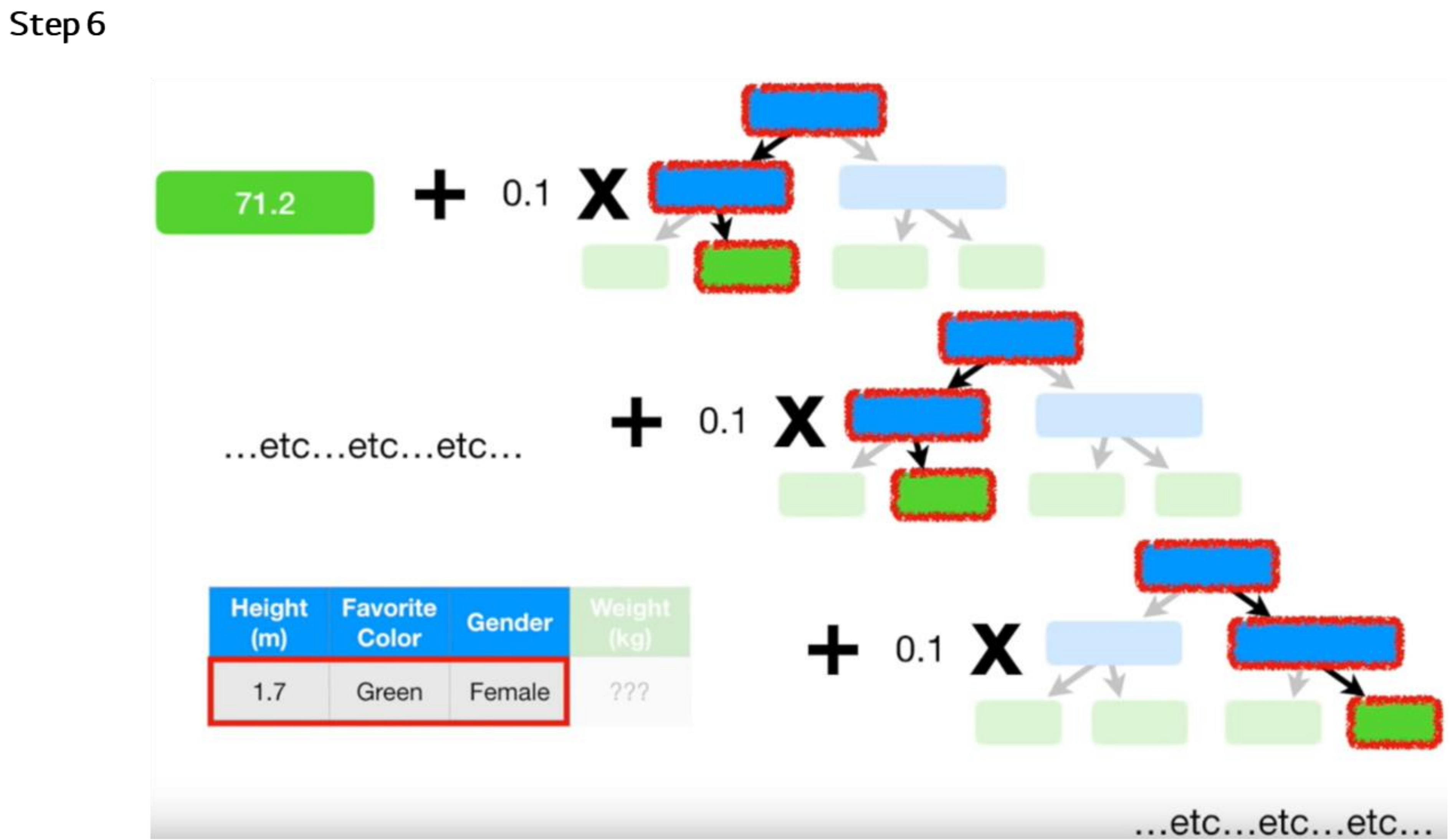
- GBM : Gradient Boosting Machine
  - Step 6
  - Set up New Tree

Step 6



Gradient Boosting Machine

- GBM : Gradient Boosting Machine
  - Step 6
  - Set up New Tree



Gradient Boosting Machine

- GBM : Gradient Boosting Machine

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i),\ f_k \in F$$ 모든 CART Tree들을 담고 있는 함수 공간

$$= f_1(x_i) + f_2(x_i) + f_3(x_i) + \cdots + f_K(x_i)$$

새로운 함수, 어떤 기준으로 뽑을까?

기존의 함수 집합에 더해졌을 때, Loss Function이 최소가 되는 함수를 찾는다.

$$L(\phi) = \underbrace{\sum_{i=1}^{n} l(y_i, \hat{y}_i)}_{\text{Training Loss}} + \underbrace{\boxed{\sum_k \Omega(f_k)}}_{\text{Regularization: Complexity of the Trees}} \rightarrow \Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2$$

Solution: Additive Training (Boosting)

$$\hat{y}_i^0 = 0$$

$$\hat{y}_i^1 = f_1(x_i) = \hat{y}_i^0 + f_1(x_i)$$

$$\hat{y}_i^2 = f_1(x_i) + f_2(x_i) = \hat{y}_i^1 + f_2(x_i)$$

$$\underbrace{\hat{y}_i^t}_{\text{t번째 Prediction}} = \sum_{k=1}^{t} f_k(x_i) = \underbrace{\hat{y}_i^{t-1}}_{\text{t-1번째 Prediction}} + \underbrace{f_t(x_i)}_{\text{Added Function}}$$

$$L(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_k \Omega(f_k)$$

$$\Downarrow \boxed{\hat{y}_i = \hat{y}^{t-1} + f_t(x_i)}$$

$$L(\phi) = \sum_{i=1}^{n} l\left(y_i, \hat{y}^{t-1} + f_t(x_i)\right) + \Omega(f_t)$$

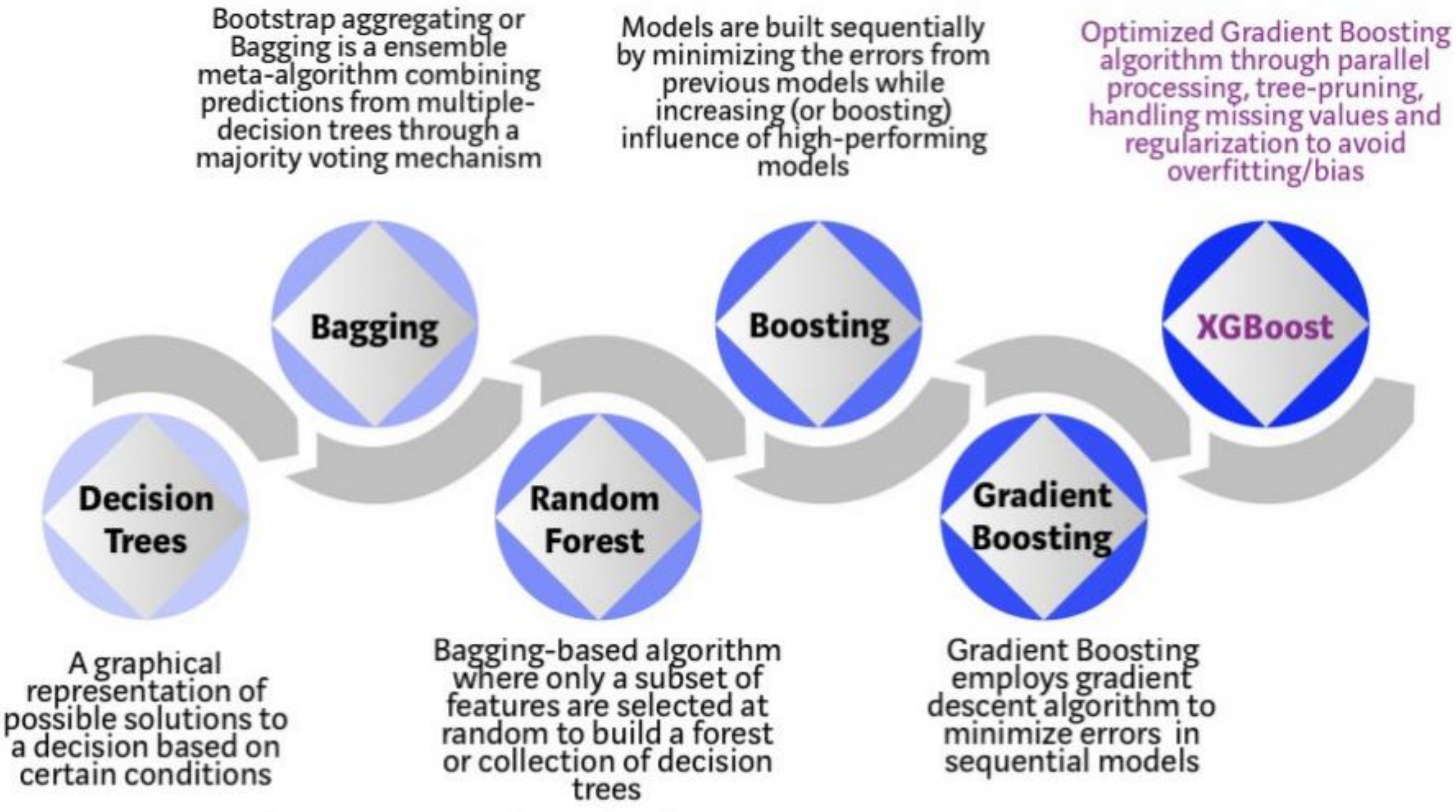
$$\Downarrow \boxed{\textit{Taylor Expansion}}$$

$$l(y_i, \hat{y}^{t-1} + f_t(x_i)) \ \xrightarrow{\text{Taylor Expansion}}\ l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)$$

XGBoost & LightGBM

XGBoost

- XGBoost : A Scalable Tree Boosting System

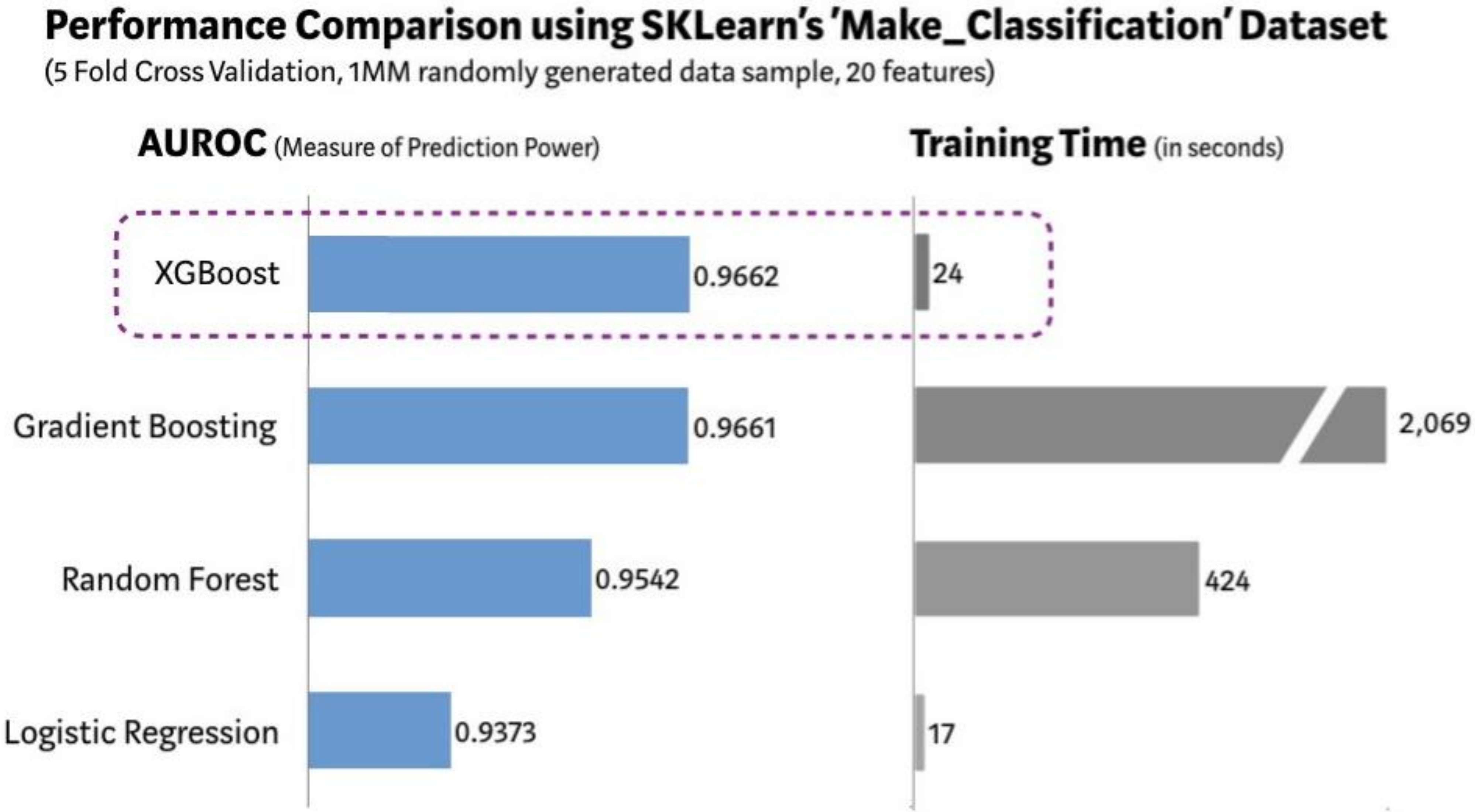


XGBoost

- XGBoost : An optimized version of GBM enabling

XGBoost

- XGBoost : An optimized version of GBM enabling
    - Kaggle No.1 (2015) Algorithm



XGBoost

- XGBoost : An optimized version of GBM enabling
  - Kaggle No.1 (2015) Algorithm

2015 Kaggle Winning Solution

17/29 Used XGBoost

11/29 Used Deep Neural Nets

2015 KDDcup

Top 10 all used XGBoost

XGBoost

- XGBoost : An optimized version of GBM enabling
  - Kaggle No.1 (2015) Algorithm

Innovations

Algorithmatic

- Tree Boosting
- Split Finding Algorithms

Systematic

- System Design

XGBoost

- XGBoost : An optimized version of GBM enabling
  - Kaggle No.1 (2015) Algorithm

Dense Data를 사용하면 좋지만…

현실에는 Sparse Data가 가득합니다

XGBoost

1. Efficiency
2. **Scalability**

XGBoost

- XGBoost : An optimized version of GBM enabling
  - Kaggle No.1 (2015) Algorithm

Algorithm 3: Sparsity-aware Split Finding

Input: I, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
Input: d, feature dimension
Also applies to the approximate setting, only collect statistics of non-missing entries into buckets
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I}, g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ *to* m **do**
  // enumerate missing value goto right
  $G_L \leftarrow 0, \ H_L \leftarrow 0$
  for j *in sorted*(I_k, *ascent order by* $\mathbf{x}_{jk}$) **do**
    $G_L \leftarrow G_L + g_j, \ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L, \ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
  // enumerate missing value goto left
  $G_R \leftarrow 0, \ H_R \leftarrow 0$
  for j *in sorted*(I_k, *descent order by* $\mathbf{x}_{jk}$) **do**
    $G_R \leftarrow G_R + g_j, \ H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R, \ H_L \leftarrow H - H_R$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split and default directions with max gain

Value	1.3		1.1	0.2		1.9	0.5		1.5	1.8
Class	1	0	1	0	0	1	0	0	1	1

Value	0.2	0.5	0.8	1.1	1.3	1.5	1.9			
Class	0	0	1	1	1	1	1	0	0	0

Value				0.2	0.5	0.8	1.1	1.3	1.5	1.9
Class	0	0	0	0	0	1	1	1	1	1

Best split, default direction = left

XGBoost

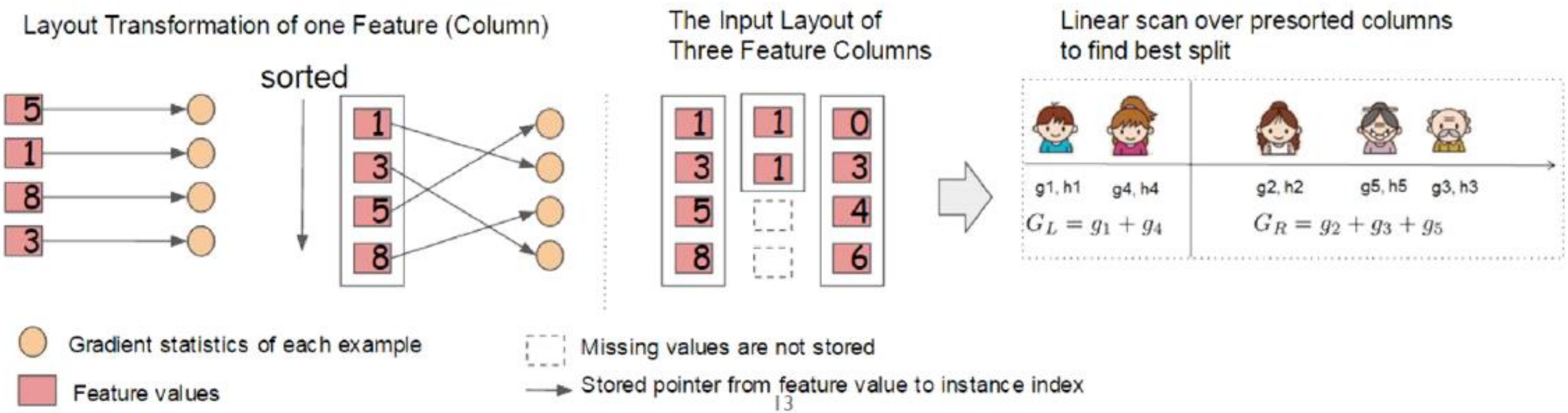
- XGBoost : An optimized version of GBM enabling
  - Kaggle No.1 (2015) Algorithm

✓ The most time-consuming part of tree learning

- to get the data into sorted order

✓ XGBoost propose to store the data in in-memory units called block

- Data in each block is stored in the compressed column (CSC) format, with each column sorted by the corresponding feature value
- This input data layout only needs to be computed once before training and can be reused in later iterations.



XGBoost

- XGBoost : An optimized version of GBM enabling
  - Kaggle No.1 (2015) Algorithm

XGBOOST

Gradient Boosting

Additive Optimization in Functional Space

Regularization

Prevent Overfitting

Column Sampling

Random Forest

Sparsity-Aware Learning

Parallel Tree Learning

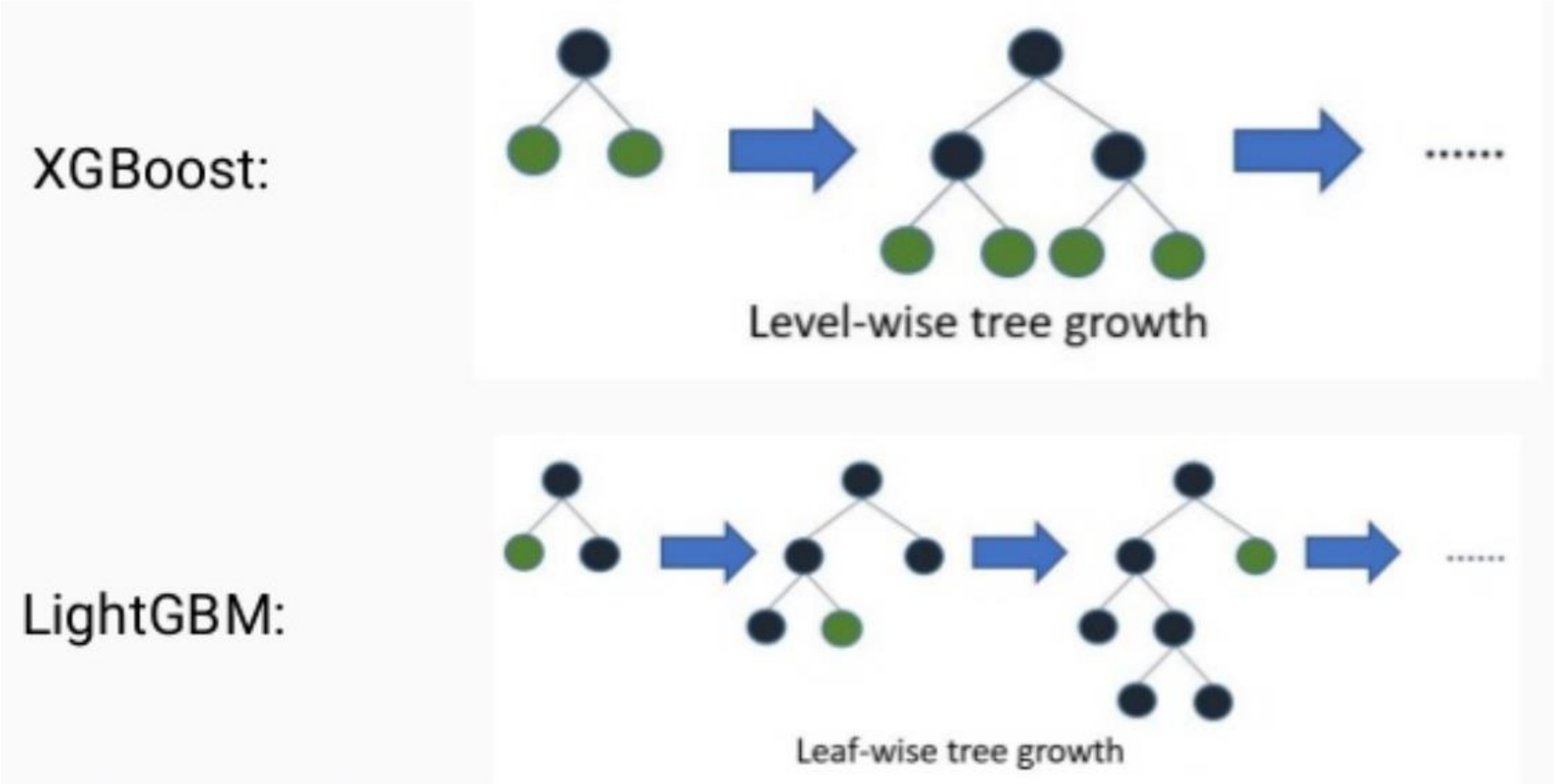
Systematic Improvement

Table 1: Comparison of major tree boosting systems.

System	exact greedy	approximate global	approximate local	out-of-core	sparsity aware	parallel
XGBoost	yes	yes	yes	yes	yes	yes
pGBRT	no	no	yes	no	no	yes
Spark MLLib	no	yes	no	no	partially	yes
H2O	no	yes	no	no	partially	yes
scikit-learn	yes	no	no	no	no	no
R GBM	yes	no	no	no	partially	no

LightGBM

- LightGBM
  - Level-wise Tree는 균형을 잡아주어야 하기 때문에 Tree Depth가 줄어 듬
    - 균형을 잡아주기 위한 연산이 추가 되어 시간이 조금 더 걸림
  - Leaf-wise Tree는 비대칭적이고 깊은 Tree가 생성됨
    - 동일한 leaf를 생성할 때 leaf-wise는 level-wise 보다 손실을 더 줄일 수 있음
    - 하지만 Overfitting 가능성이 있으며, 데이터 양이 적을 때 비효율 적임
    - 논문에서는 10,000개 데이터 미만일 때 너무 쉽게 Overfitting 될 수 있다고 기재되어 있음



XGBoost vs LightGBM vs CatBoost

	XGBoost	Light BGM		CatBoost	
Parameters Used	max_depth: 50 learning_rate: 0.16 min_child_weight: 1 n_estimators: 200	max_depth: 50 learning_rate: 0.1 num_leaves: 900 n_estimators: 300		depth: 10 learning_rate: 0.15 l2_leaf_reg= 9 iterations: 500 one_hot_max_size = 50	
Training AUC Score	0.999	Without passing indices of categorical features	Passing indices of categorical features	Without passing indices of categorical features	Passing indices of categorical features
		0.992	0.999	0.842	0.887
Test AUC Score	0.789	0.785	0.772	0.752	0.816
Training Time	970 secs	153 secs	326 secs	180 secs	390 secs
Prediction Time	184 secs	40 secs	156 secs	2 secs	14 secs
Parameter Tuning Time (for 81 fits, 200 iteration)	500 minutes	200 minutes		120 minutes	

Q & A