XGBOOST 논문 리뷰

한양대학교 산업공학과
황예준

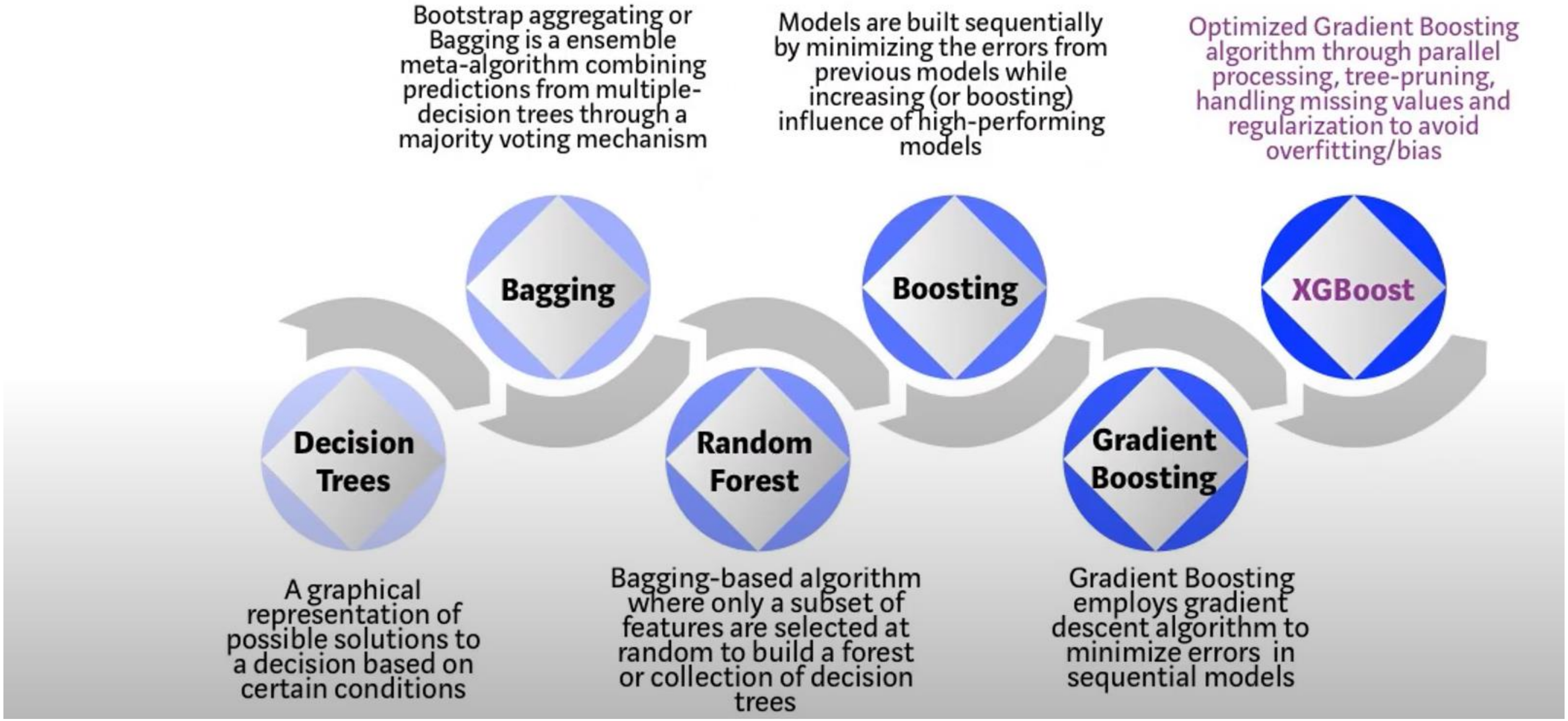
목차

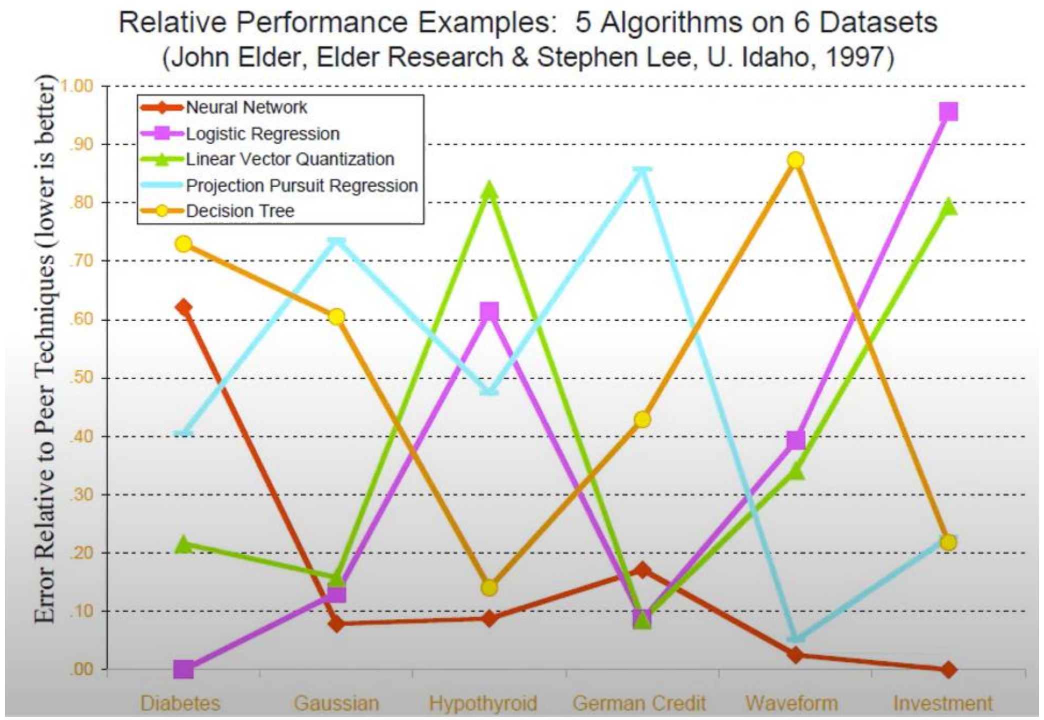
- Introduction

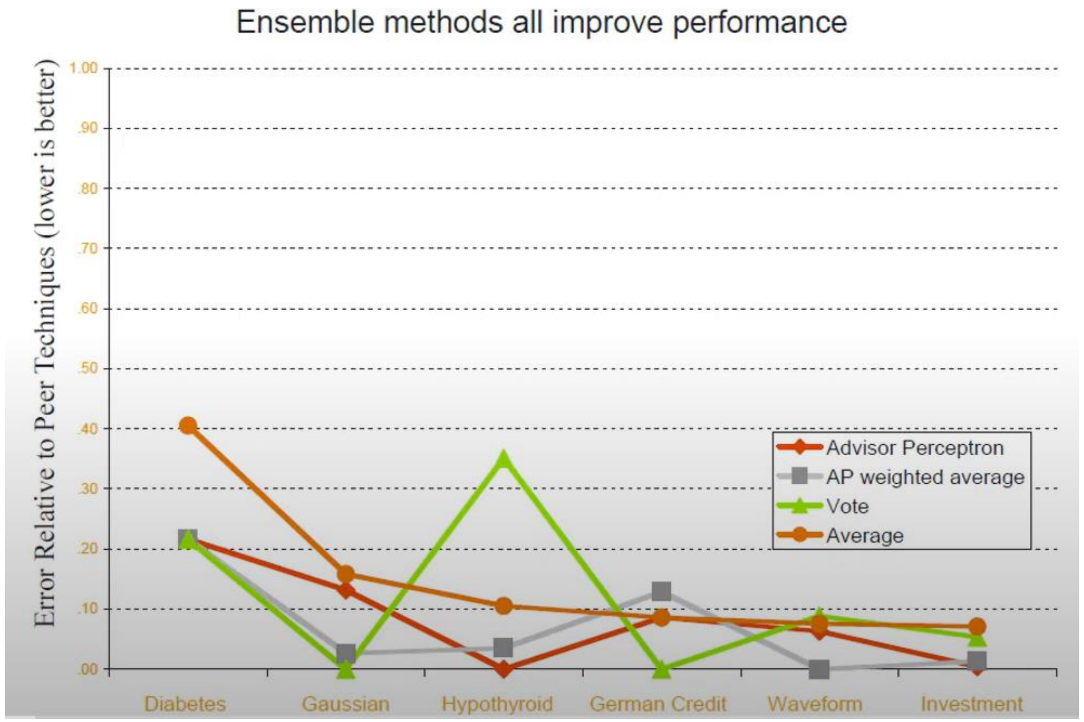
- 앙상블(Ensemble)
  - Bagging
  - Boosting

- XGBOOST

- XGBoost: A Scalable Tree Boosting System



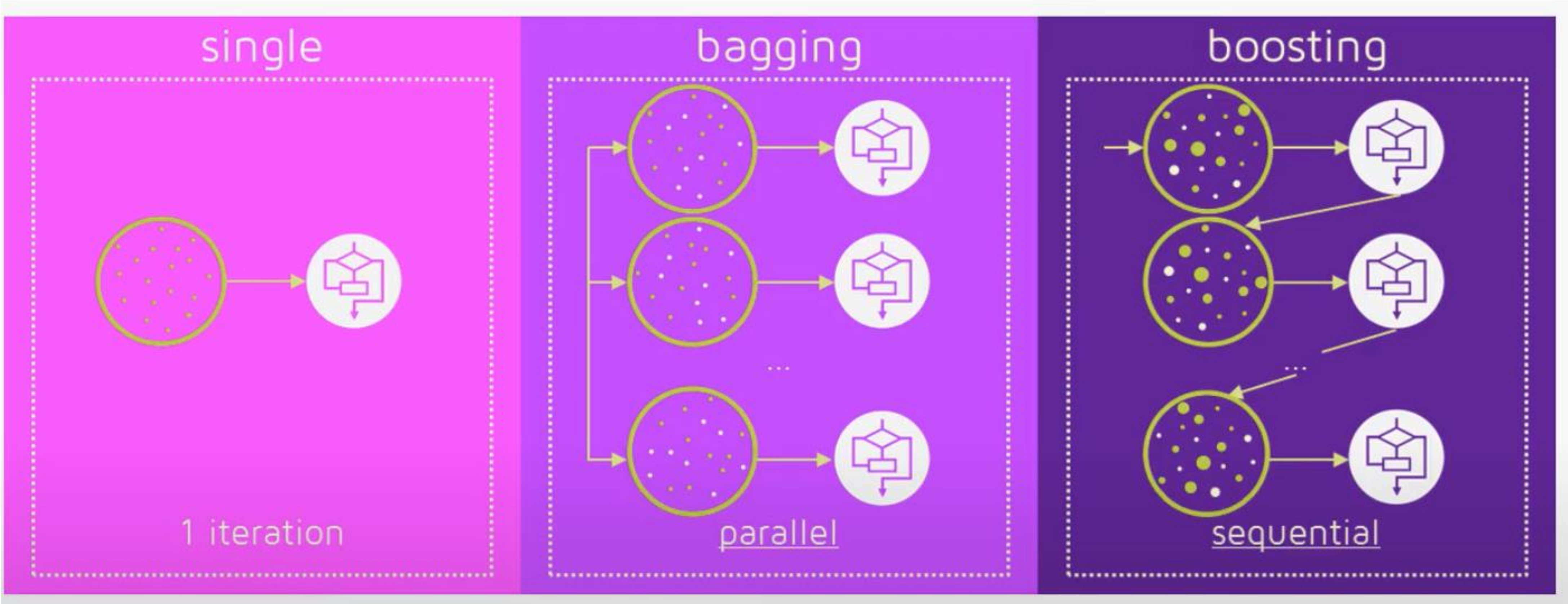




개별 모델들의 성능

앙상블된 모델들의 성능

앙상블 하는 방법



- Final output $\hat{y} = \delta\Big(f_1(\mathbf{x}), f_2(\mathbf{x}), \cdots, f_{k-1}(\mathbf{x}), f_k(\mathbf{x})\Big)$

  ✓ $\delta(\cdot)$: An aggregation function of individual outputs (ex: simple average)

앙상블의 효과(수식)

앙상블의 개별 모델: $y_m(\mathbf{x}) = f(\mathbf{x}) + \epsilon_m(\mathbf{x}).$

개별 모델의 Error: $\mathbb{E}_{\mathbf{x}}\left[\{y_m(\mathbf{x}) - f(\mathbf{x})\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$

M개의 모델의 Error 평균: $E_{Avg} = \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$

앙상블의 Error: $E_{Ensemble} = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x}) - f(\mathbf{x})\right\}^2\right]$

$$= \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right\}^2\right]$$

1)이상적인 상황($E(\epsilon_m(x))=0$, $\epsilon_m(x)$은 uncorrelated)

$$E_{Ensemble} = \frac{1}{M} E_{Avg}$$

2)각 모델간 Correlation이 존재하는 현실

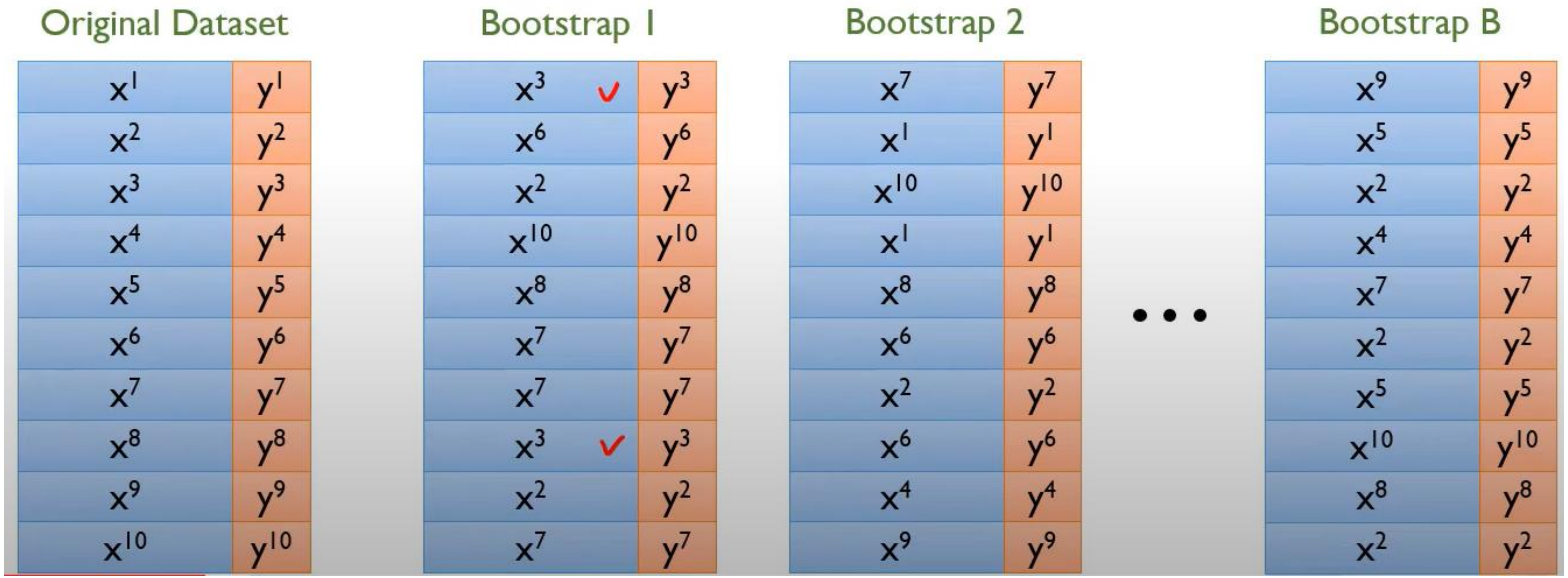
By 코시슈바르츠 방정식

$$\left[\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right]^2 \leq M \sum_{m=1}^{M} \epsilon_m(\mathbf{x})^2 \Rightarrow \left[\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right]^2 \leq \frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})^2$$

$$E_{Ensemble} \leq E_{Avg}$$

즉, 최소한 Ensemble의 효과가 개별 모델보다는 좋다는 것을 알 수 있다.

Bagging



- 다른 data set을 만들 때 "복원추출"함
- 각 데이터별로 한번도 선택 안되거나 여러 번 선택될 수 있음

단순한 복원추출의 Bootstrap이 무슨 의미?

:데이터가 가지는 분포를 왜곡하는 효과

$$Y=f(x)+\varepsilon$$

OOB(Out Of Bag)데이터

:복원 추출시 단 한번도 선택되지 않은 데이터

$$p=\left(1-\frac{1}{N}\right)^N \rightarrow \lim_{N\to\infty}\left(1-\frac{1}{N}\right)^N = e^{-1} = 0.368$$ (p는 단 한번도 선택되지 않을 확률)

- N이 일정 수준 이상이면 1/3에 해당하는 데이터가 OOB 데이터가 됨
- Deterministic한 게 아니라 Probalistic하게 검증용 데이터를 얻을 수 있다는 장점

Boosting(Adaboost)

Weak model:Random Gauess보다 조금 더 성능이 좋은 모델

Boosting방법

1)학습 데이터 준비

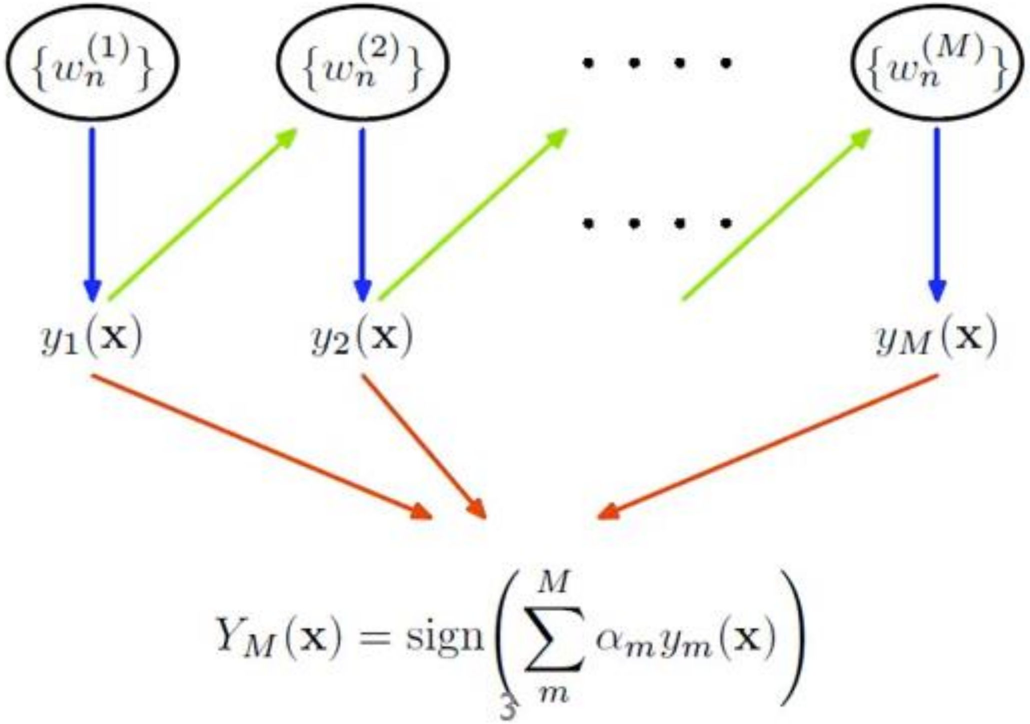
2)Week model 준비

3)기존의 Week model로 맞추지 못한 hard case에서

더 많이 samplin되도록 학습 데이터의 weight를 새롭게 부과

4)새로운 weight를 가지는 학습 데이터로 다음 week model생

5)만들어진 개별 week model을 취합하여 최종값으로 사용



- AdaBoosting: Algorithm

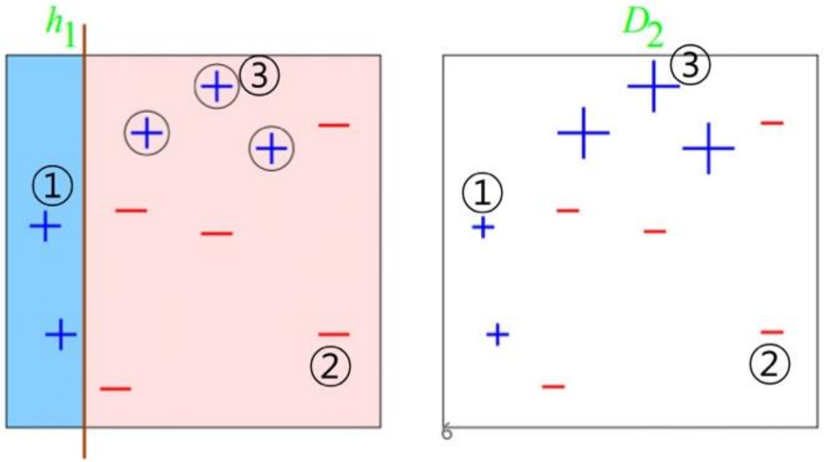
Algorithm 2 Adaboost

Input: Required ensemble size T
Input: Training set $S = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, where $y_i \in \{-1, +1\}$
Define a uniform distribution $D_1(i)$ over elements of S.
for $t = 1$ to T **do**
  Train a model h_t using distribution D_t.
  Calculate $\epsilon_t = P_{D_t}(h_t(x) \neq y)$
  If $\epsilon_t \geq 0.5$ break
  Set $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  Update $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
  where Z_t is a normalization factor so that D_{t+1} is a valid distribution.
end for
For a new testing point (x', y'),
$H(x') = sign\Big(\sum_{t=1}^{T} \alpha_t h_t(x')\Big)$

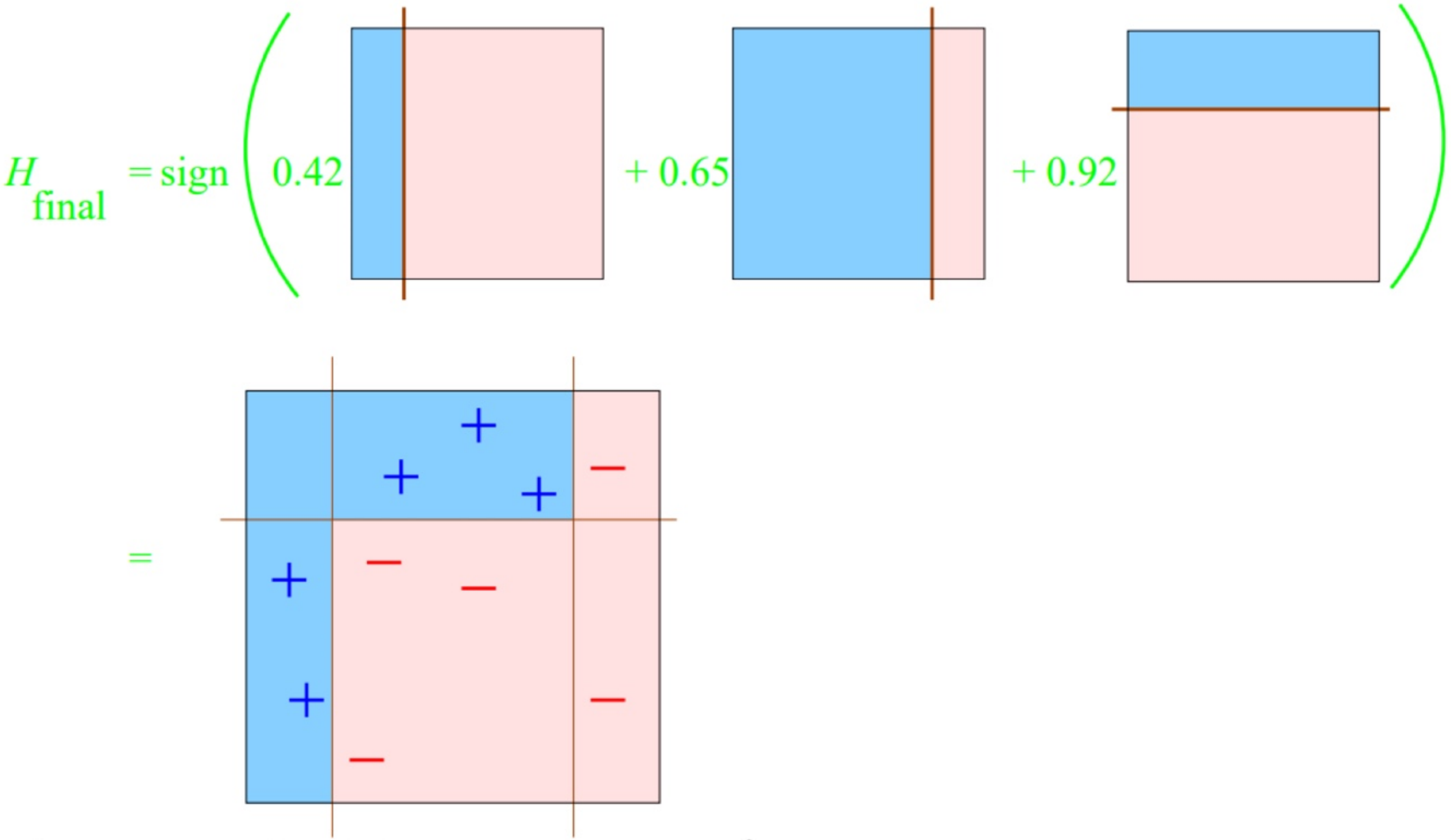
- AdaBoost Example
  - ✓ The selection probability of x_i for the next training dataset

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

  - ✓ Case 1: $y_i = 1, h_t(x_i) = 1 \rightarrow y_i h_t(x_i) = 1 \rightarrow -\alpha_t y_i h_t(x_i) < 0 \rightarrow$ decrease p
  - ✓ Case 2: $y_i = -1, h_t(x_i) = -1 \rightarrow y_i h_t(x_i) = 1 \rightarrow -\alpha_t y_i h_t(x_i) < 0 \rightarrow$ decrease p
  - ✓ Case 3: $y_i = 1, h_t(x_i) = -1 \rightarrow y_i h_t(x_i) = -1 \rightarrow -\alpha_t y_i h_t(x_i) > 0 \rightarrow$ increase p
  - ✓ α_t is the confidence of the current model that controls the magnitude of change



✓ Final classifier

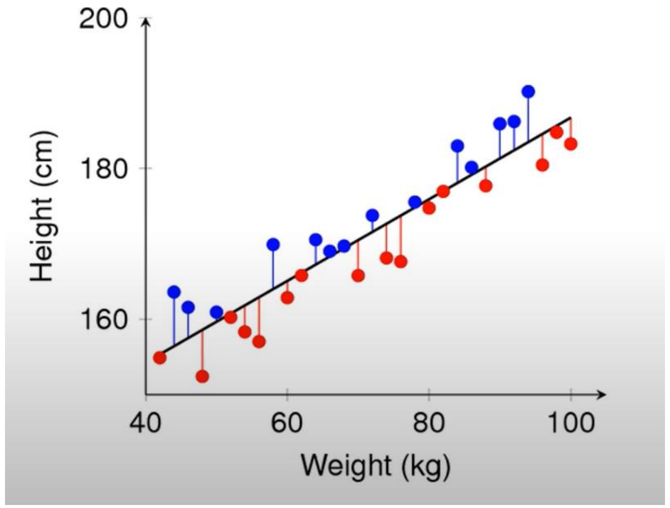


Boosting(Gradient Boost)

기존 week model를 보완 하는 방법

AdaBoost
:예측 실패한 데이터에 대해 high weight를 줌으로써 보완된 weak model을 만듬

GradientBoost
:예측 실패한 데이터에 대해 Gradient를 이용하여 보완된 weak model을 만듬



선형 회귀 모형 ŷ=f1(x)

완벽하게 데이터를 추정하지 못하는 것을 볼 수 있음

첫번째 모델이 추정하지 못했던 오차(y-f1(x))만큼 추정하는
두번째 모델을 만든다면?

즉, f1(x)는 y를 추정하기 위한 모델

f2(x)는 y-f1(x)를 추정하기 위한 모델

- Main idea

Original Dataset

x^1	y^1
x^2	y^2
x^3	y^3
x^4	y^4
x^5	y^5
x^6	y^6
x^7	y^7
x^8	y^8
x^9	y^9
x^{10}	y^{10}

Modified Dataset 1

x^1	$y^1\text{-}f_1(x^1)$
x^2	$y^2\text{-}f_1(x^2)$
x^3	$y^3\text{-}f_1(x^3)$
x^4	$y^4\text{-}f_1(x^4)$
x^5	$y^5\text{-}f_1(x^5)$
x^6	$y^6\text{-}f_1(x^6)$
x^7	$y^7\text{-}f_1(x^7)$
x^8	$y^8\text{-}f_1(x^8)$
x^9	$y^9\text{-}f_1(x^9)$
x^{10}	$y^{10}\text{-}f_1\,(x^{10})$

Modified Dataset 2

x^1	$y^1\text{-}f_1(x^1)\ \text{-}f_2(x^1)$
x^2	$y^2\text{-}f_1(x^2)\ –f_2(x^2)$
x^3	$y^3\text{-}f_1(x^3)\ –f_2(x^3)$
x^4	$y^4\text{-}f_1(x^4)\ \text{-}f_2(x^4)$
x^5	$y^5\text{-}f_1(x^5)\ \text{-}f_2(x^5)$
x^6	$y^6\text{-}f_1(x^6)\ \text{-}f_2(x^6)$
x^7	$y^7\text{-}f_1(x^7)\ \text{-}f_2(x^7)$
x^8	$y^8\text{-}f_1(x^8)\ \text{-}f_2(x^8)$
x^9	$y^9\text{-}f_1(x^9)\ –f_2(x^9)$
x^{10}	$y^{10}\text{-}f_1\,(x^{10})\ –f_2(x^{10})$

• • •

$$y = f_1(\mathbf{x}) \qquad y - f_1(\mathbf{x}) = f_2(\mathbf{x}) \qquad y - f_1(\mathbf{x}) - f_2(\mathbf{x}) = f_3(\mathbf{x})$$

왜 Gradient하고 연관이 되는가?

회귀모형에서 Loss함수로 많이 쓰는 OLS(Ordinary Least Square) 최소자승법

$$\min L = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(x_i))^2.$$

여기서 $f(x_i)$에 대해 Loss를 미분해보면

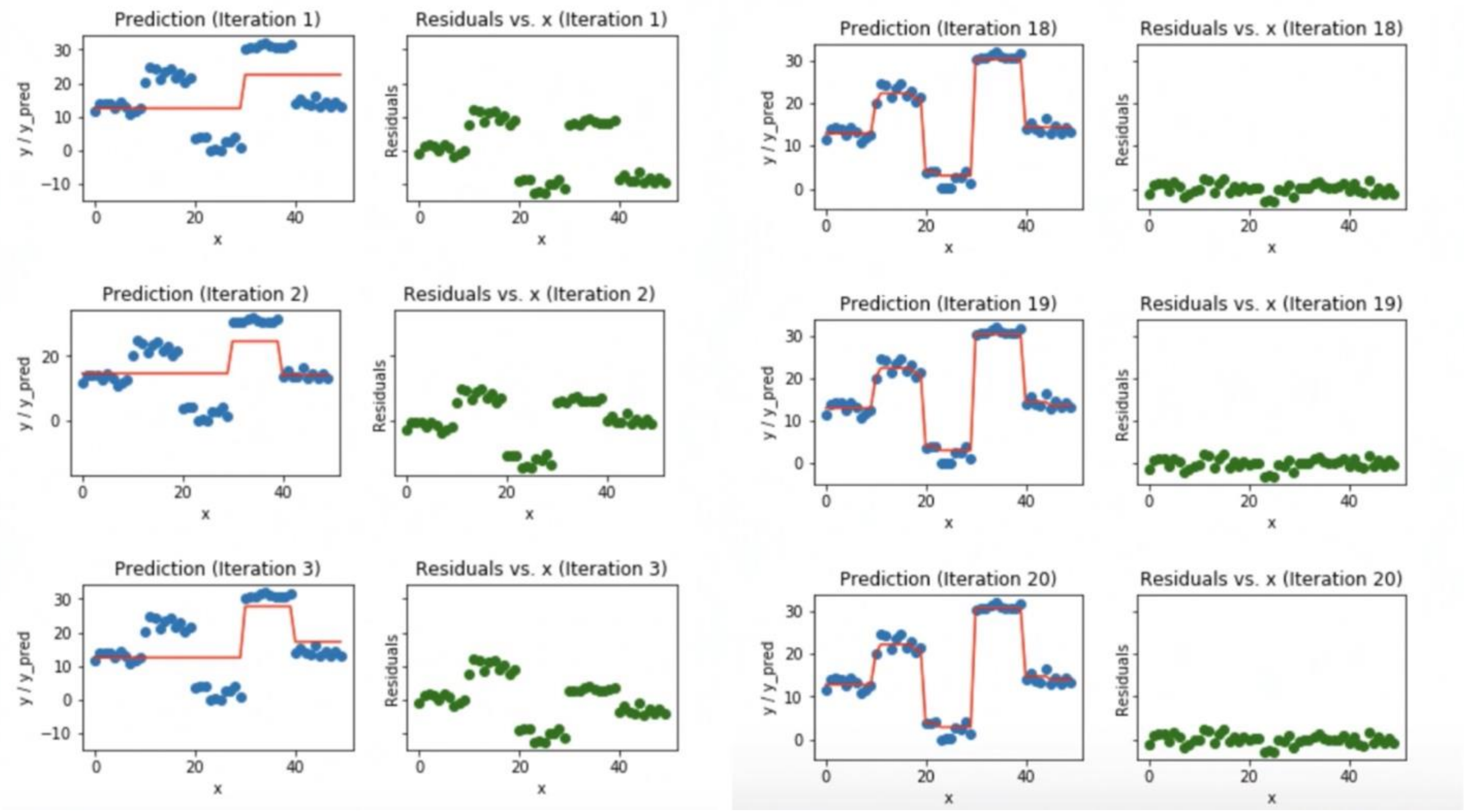
$$\frac{\partial L}{\partial f(x_i)} = f(x_i) - y_i$$ 가 됨.

즉, $$y_i - f(x_i) = -\frac{\partial L}{\partial f(x_i)}$$ (마이너스 gradient)

즉, 잔차만큼 학습하는 것은 gradient의 반대 방향으로 학습하는 것과 같은 의미

Gradient Boosting Machine: GBM 개별모델:tree

- GBM Regression Example 1



XGBOOST

:Gradient Boost 방법론을 사용하되,

조금 더 빠르고 대용량의 데이터에 대해 사용하도록 변형시킴

Algorithmatic

- Split Finding Algorithm
- Spare-Aware Split Finding

Systematic

- CSC(Compressed Column format)
- Cache-aware access

1)Split Finding Algorithm

① Basic exact greedy algorithm

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i,\ H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I,$ *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
end
Output: Split with max score

장점:언제나 최적의 split point를 찾을 수 있다.

단점:데이터가 memory에 한번에 로딩 되지 않을 때 알고리즘 자체를 수행할 수 없다.
모든 경우의 수를 탐색해야 하여 분산환경에서 처리가 불가능.

1)Split Finding Algorithm

② Approximate algorithm

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**

    Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.

    Proposal can be done per tree (global), or per split(local).

end

for $k = 1$ ***to*** m **do**

    $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$

    $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$

end

Follow same step as in previous section to find max score only among proposed splits.

전체 데이터를
l개의 bucket으로 분할을 함.

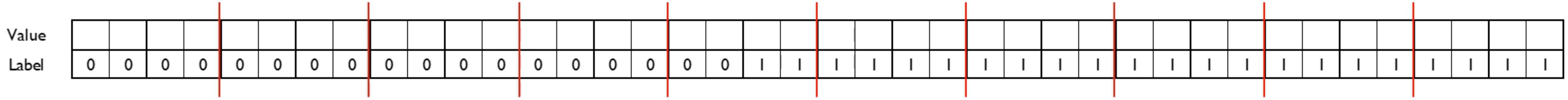
k는 변수.

1)Split Finding Algorithm

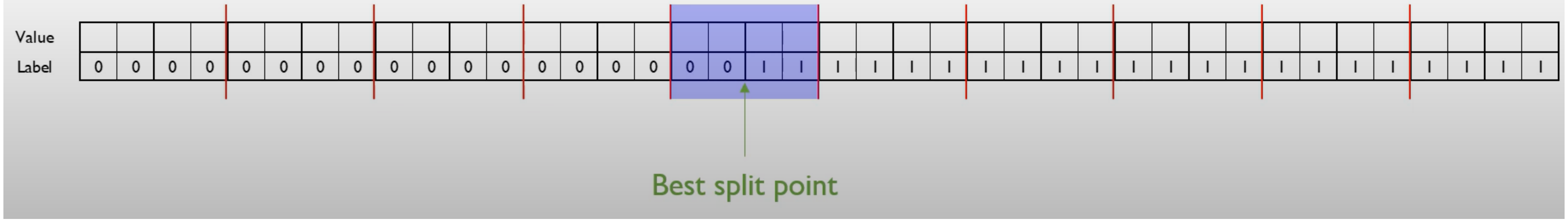
② Approximate algorithm

✓ Approximate algorithm: an illustrative example

- Assume that the value is sorted in an ascending order (left: small, right: large)
- Divide the dataset into 10 buckets



- Compute the gradient for each bucket and find the best split



Exact algorithm:존재하는 모든 split point, 모든 데이터에 대해 gradient계산

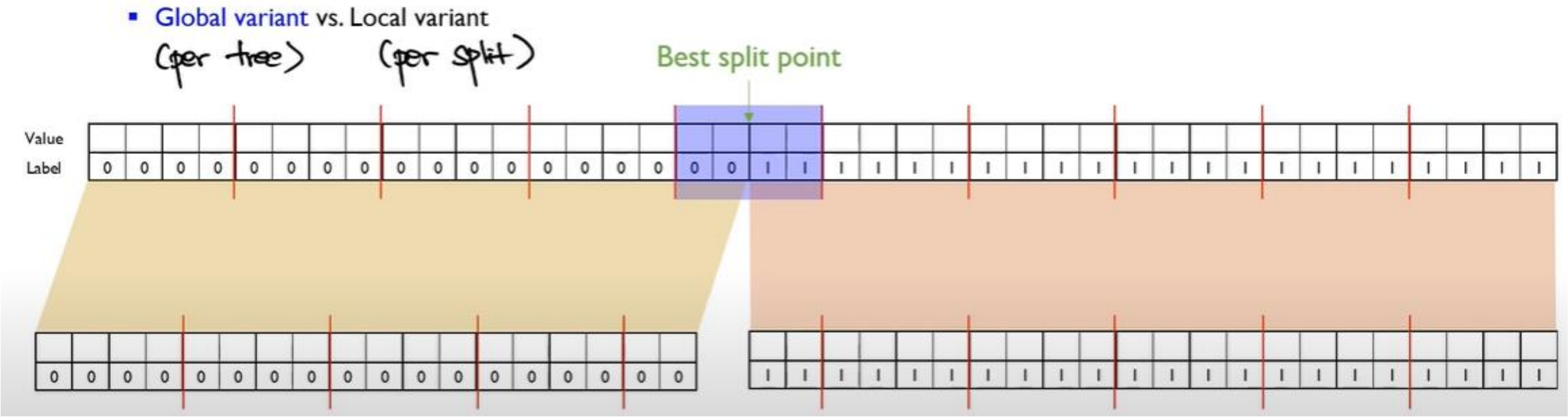
Aprroximate :Bucket을 나눈 후 해당 Bucket 내에서 split point와 데이터에 대해 gradient 계산

1)Split Finding Algorithm

② Approximate algorithm

Bucket을 나누는 2가지 방법. Global variant 와 Local variant

Global variant
: Tree에서 Reculsive 하게 plunning함에 있어 처음에 정한 Bucket 기준을 유지

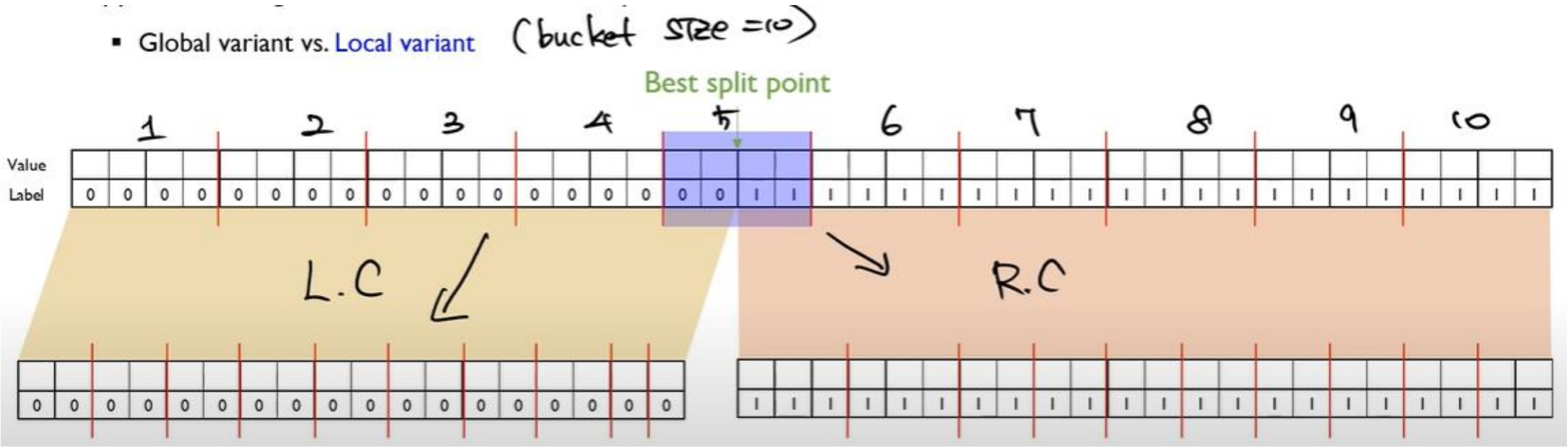


1)Split Finding Algorithm

② Approximate algorithm

Bucket을 나누는 2가지 방법. Global variant 와 Local variant

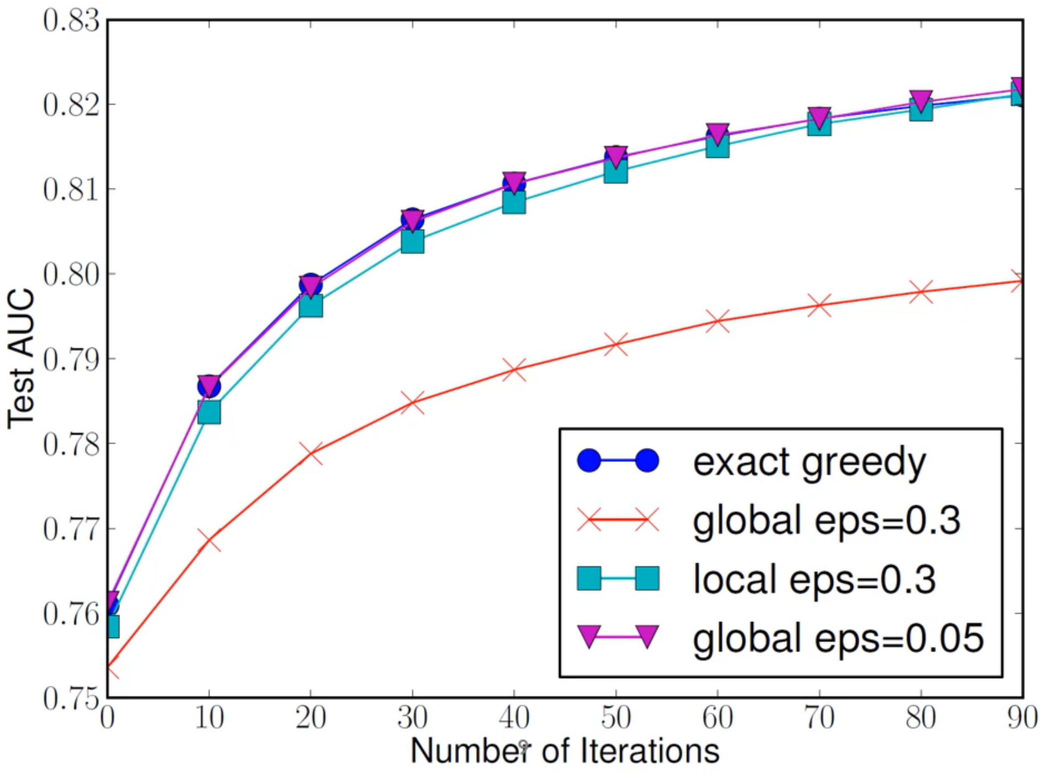
Local variant
: Tree에서 Reculsive 하게 plunning함에 있어 Parent Node와 Child Node가 Bucket개수가 같도록 함



1)Split Finding Algorithm

② Approximate algorithm

Split Finidng을 approximate하게 함으로 Split candidat도 줄이고 병렬화도 가능해진다.



1/eps 개의 bucket 만들어짐

Global variant방법은 eps이 작을때
성능이 더 좋음을 볼 수 있음

2)Sparse-Aware Split Finding

:결측치(Missing data)를 효율적으로 처리하고자 하는 관점

각 split마다 default direction을 학습과정에서 정하고

새로운 데이터가 들어왔을 때 값이 missing이면 정의했던 default direction으로 보내버리는 방법

- **Sparsity-Aware Split Finding**

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

결측 데이터 전부 우측으로

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

1 2 3 4 5 6 7

결측 데이터 전부 좌측으로

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

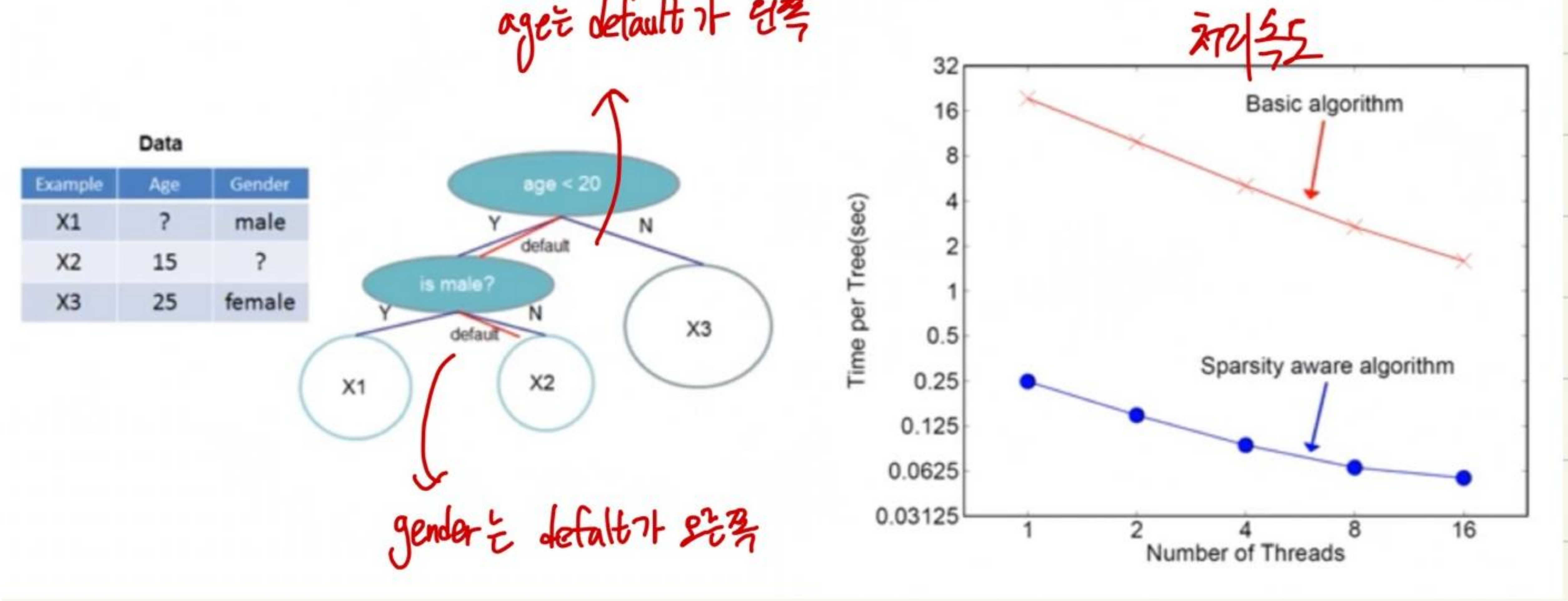
1 2 3 4 5 6 7

Best split, default direction = left

좌측으로 보냈을때 best split이 gradien에 의한 효과가 더 크므로 default direction은 왼쪽

2)Sparse-Aware Split Finding

- Sparsity-Aware Split Finding



XGBoost Exact Greedy Algorithm **VS** pGBRT (Parallel boosted regression trees)

Exact Greedy Algorithm

Approximate algorithm(Only Available)

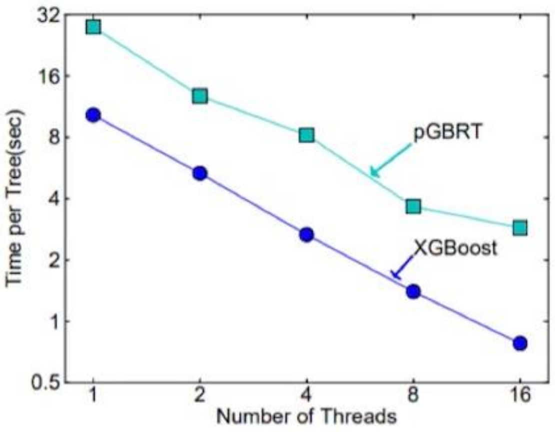


Figure 10: Comparison between XGBoost and pG-BRT on Yahoo LTRC dataset.

Exact greedy algorithm을 사용한 xgboost가 성능이 더 좋은 것은 xgboost의 하드웨어적 부분 덕분

감사합니다.