XGBoost: A Scalable Tree Boosting System

Oct 9, 2023

Seungeun Lee

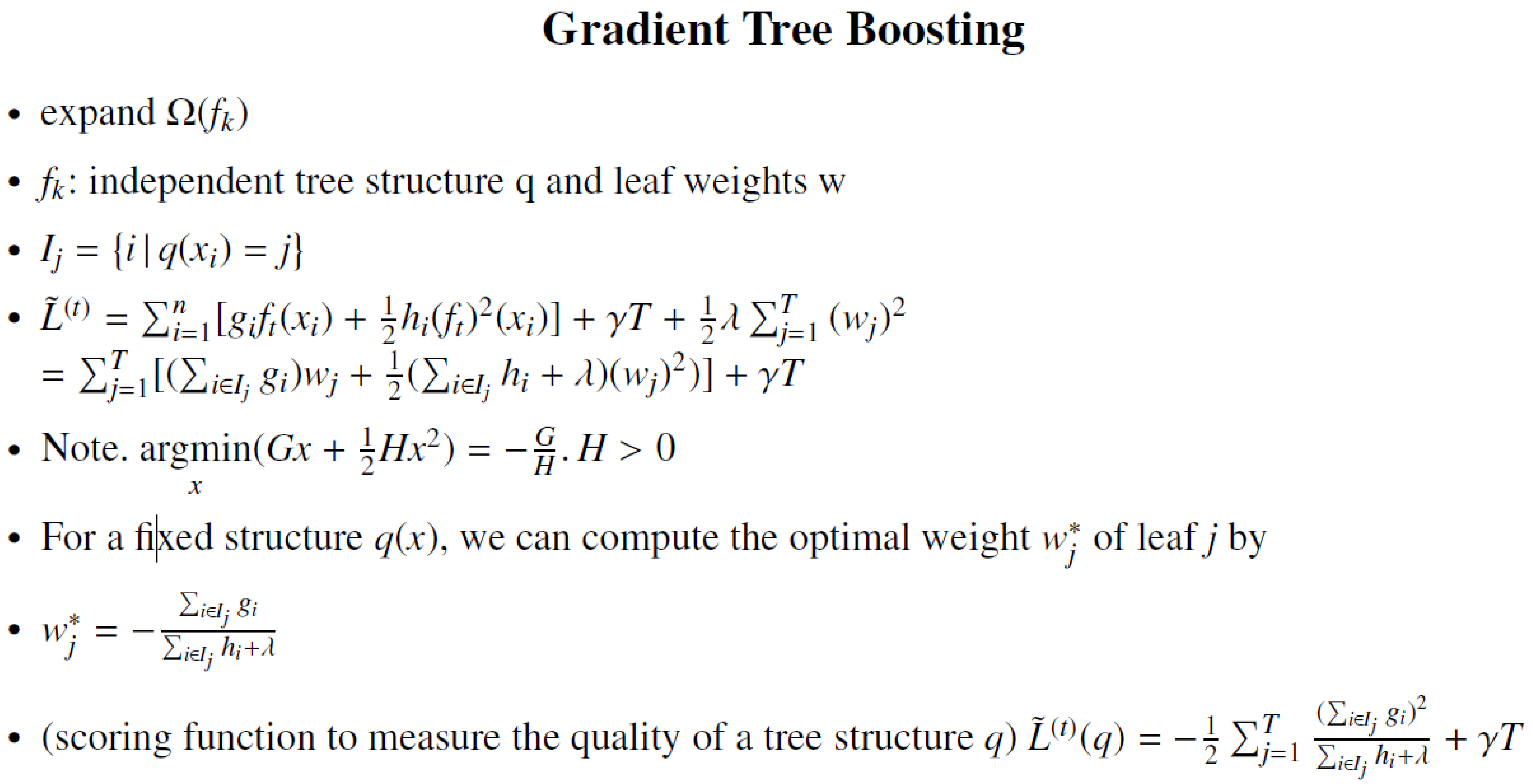
① | Recap

② | Split Finding Algorithms

③ | H/W Optimization

④ | Details

Recap

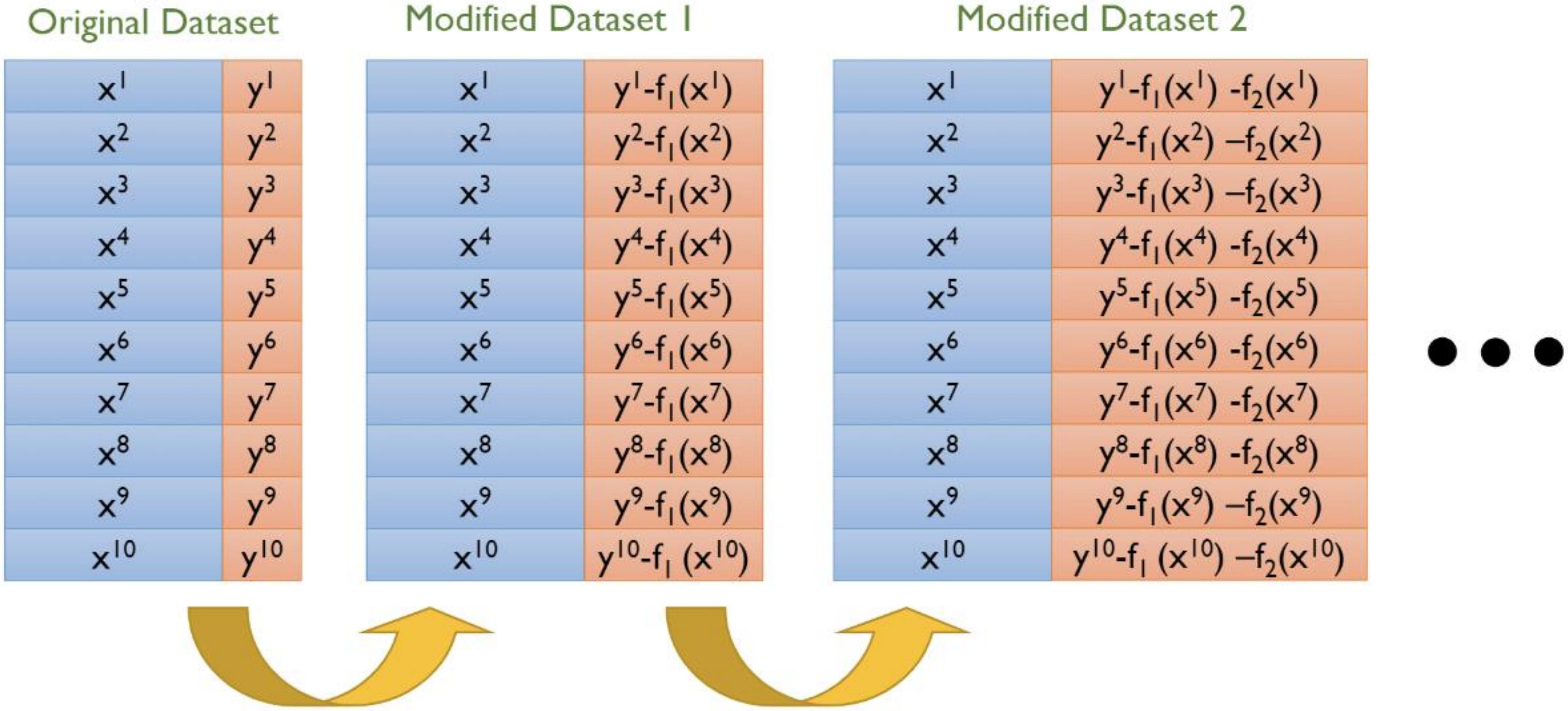


Gradient Tree Boosting

- expand $\Omega(f_k)$
- f_k: independent tree structure q and leaf weights w
- $I_j = \{i \,|\, q(x_i) = j\}$
- $\tilde{L}^{(t)} = \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i(f_t)^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T}(w_j)^2$
  $= \sum_{j=1}^{T}[(\sum_{i \in I_j} g_i)w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda)((w_j)^2)] + \gamma T$
- Note. $\operatorname*{argmin}_x(Gx + \frac{1}{2}Hx^2) = -\frac{G}{H}. H > 0$
- For a fixed structure $q(x)$, we can compute the optimal weight w_j^* of leaf j by
- $w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$
- (scoring function to measure the quality of a tree structure q) $\tilde{L}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T$

Recap

- Why?



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

Modified Dataset 2

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

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

Recap

- How is this idea related to the gradient?

✓ Loss function of the ordinary least square (OLS)

$$\min L = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2$$

✓ Gradient of the Loss function

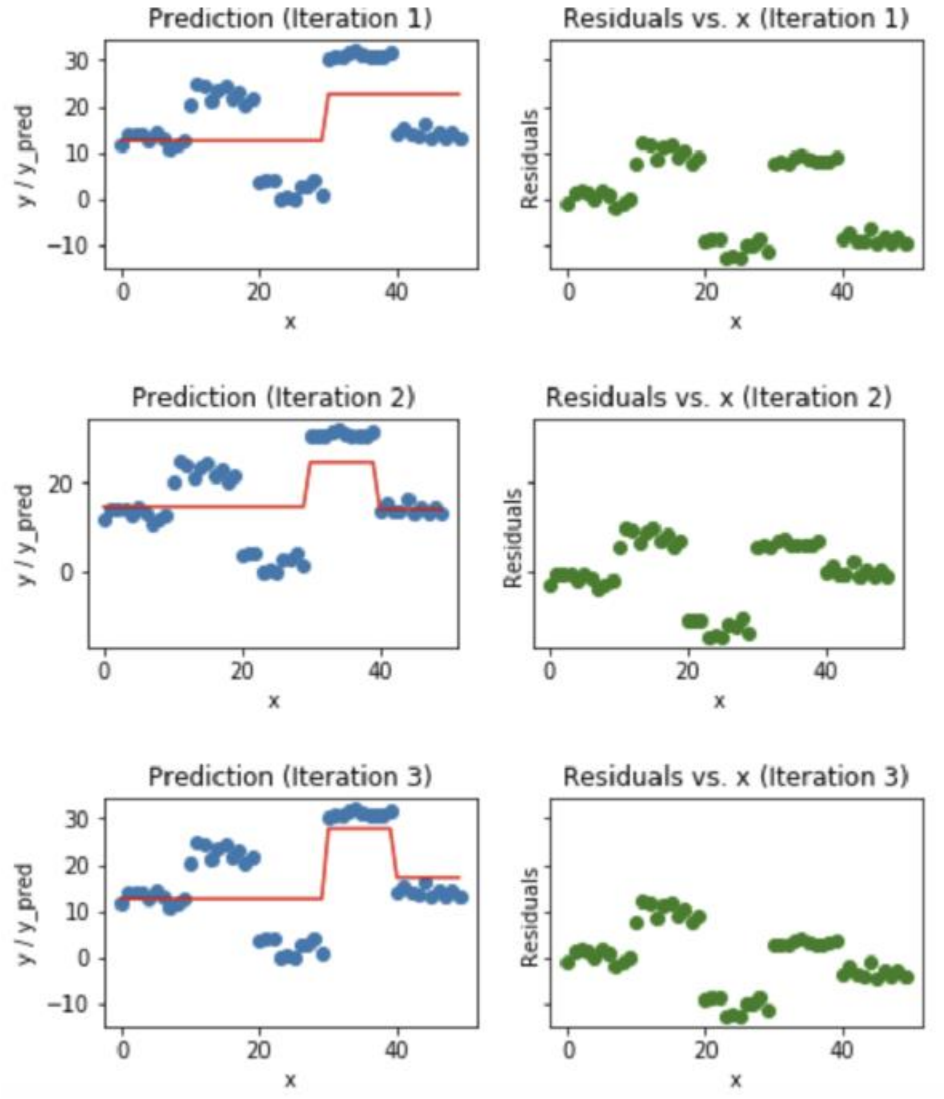
$$\frac{\partial L}{\partial f(\mathbf{x}_i)} = f(\mathbf{x}_i) - y_i$$

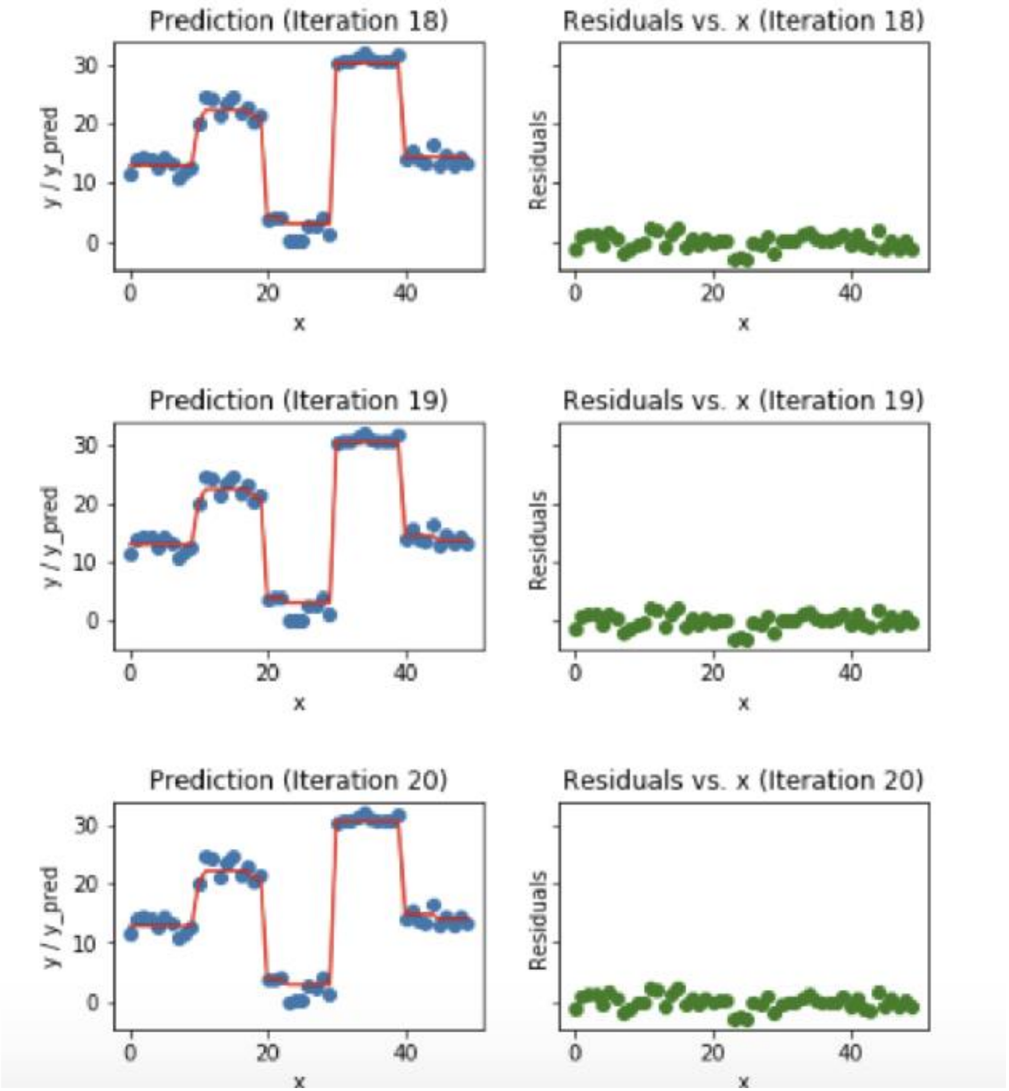
✓ Residuals are the negative gradient of the loss function

$$y_i - f(\mathbf{x}_i) = -\frac{\partial L}{\partial f(\mathbf{x}_i)}$$

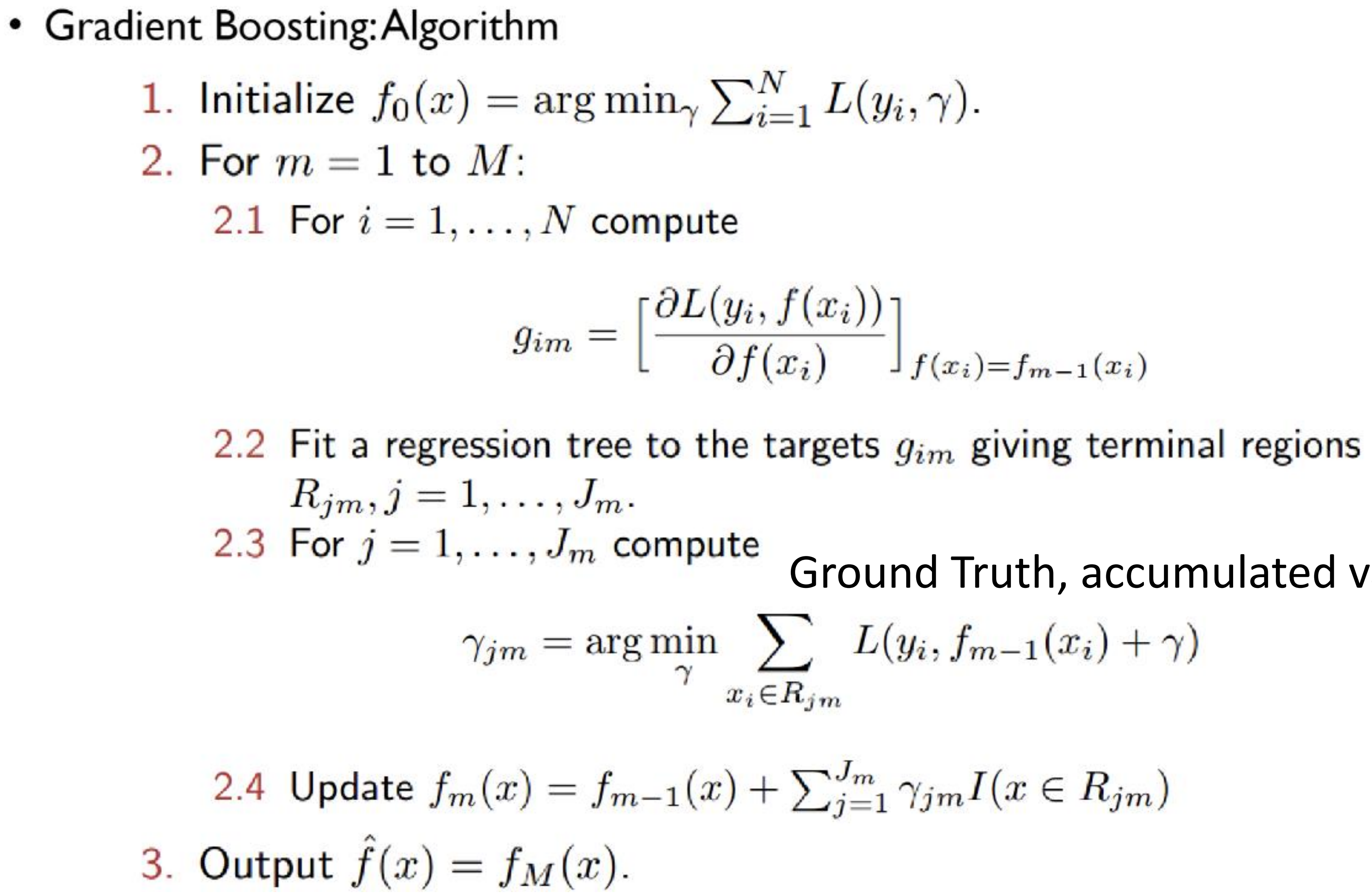
Recap

Tree depth





Recap



- Gradient Boosting: Algorithm

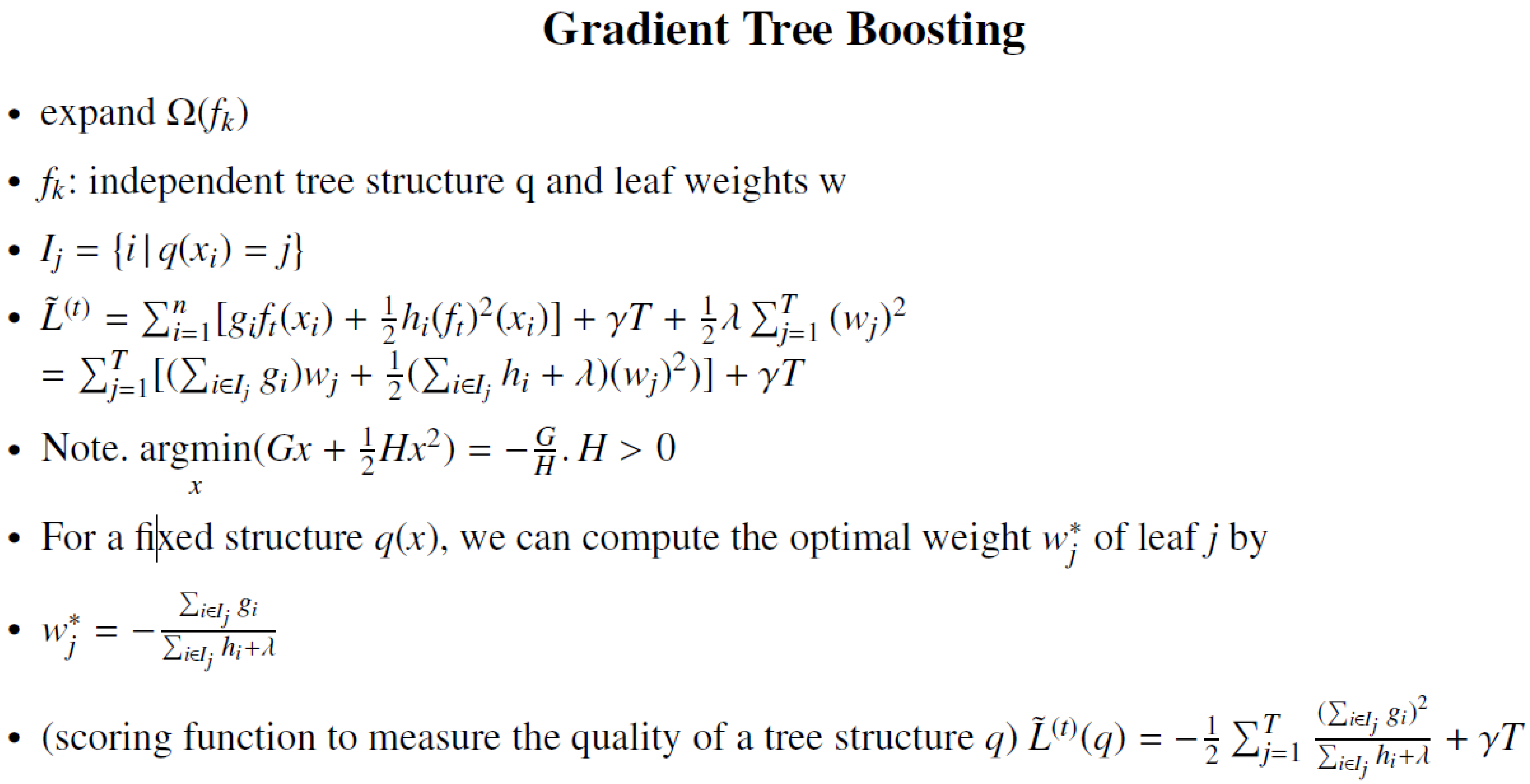
```
1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
    2.1 For $i = 1, \ldots, N$ compute
        $$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$$
    2.2 Fit a regression tree to the targets $g_{im}$ giving terminal regions
        $R_{jm}, j = 1, \ldots, J_m$.
    2.3 For $j = 1, \ldots, J_m$ compute
        $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$
    2.4 Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
3. Output $\hat{f}(x) = f_M(x)$.
```

Ground Truth, accumulated values

Recap

- However, Overfitting
- Memorizes the uncertainty or noise of the data
- Shrinkage (minimizes the power of ‘overfitted’ models), Subsampling (maintains the size of the dataset), Early Stopping …

Recap



Gradient Tree Boosting

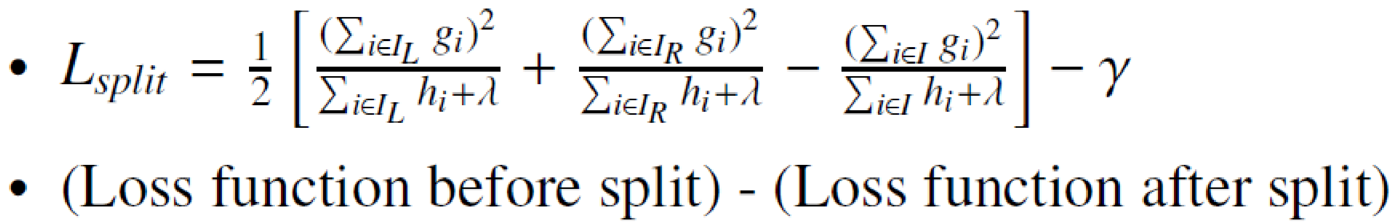
- expand $\Omega(f_k)$
- f_k: independent tree structure q and leaf weights w
- $I_j = \{i \,|\, q(x_i) = j\}$
- $\tilde{L}^{(t)} = \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i(f_t)^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} (w_j)^2$
  $= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)(w_j)^2)] + \gamma T$
- Note. $\underset{x}{\operatorname{argmin}}(Gx + \frac{1}{2}Hx^2) = -\frac{G}{H}. H > 0$
- For a fixed structure $q(x)$, we can compute the optimal weight w_j^* of leaf j by
- $w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda}$
- (scoring function to measure the quality of a tree structure q) $\tilde{L}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T$

Recap

- $w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$
- (scoring function to measure the quality of a tree structure q) $\tilde{L}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T$

- BUT we can make infinite number of trees
- We have to decide “when to SPLIT the trees” to make an optimal tree structure

Recap



- $L_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i+\lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i+\lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i+\lambda}\right] - \gamma$
- (Loss function before split) - (Loss function after split)

- (loss of left node after the split) + (loss of right node after the split) – (loss before the split)
- Choose the split with the maximized loss reduction
- Shrinkage, Column subsampling

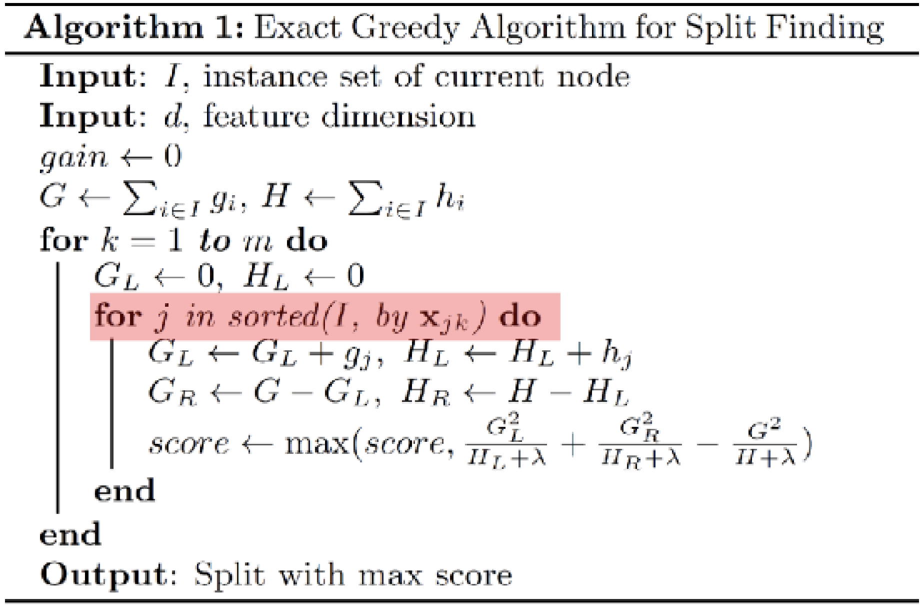
① | Recap

② | Split Finding Algorithms

③ | H/W Optimization

④ | Details

Split finding Algorithms

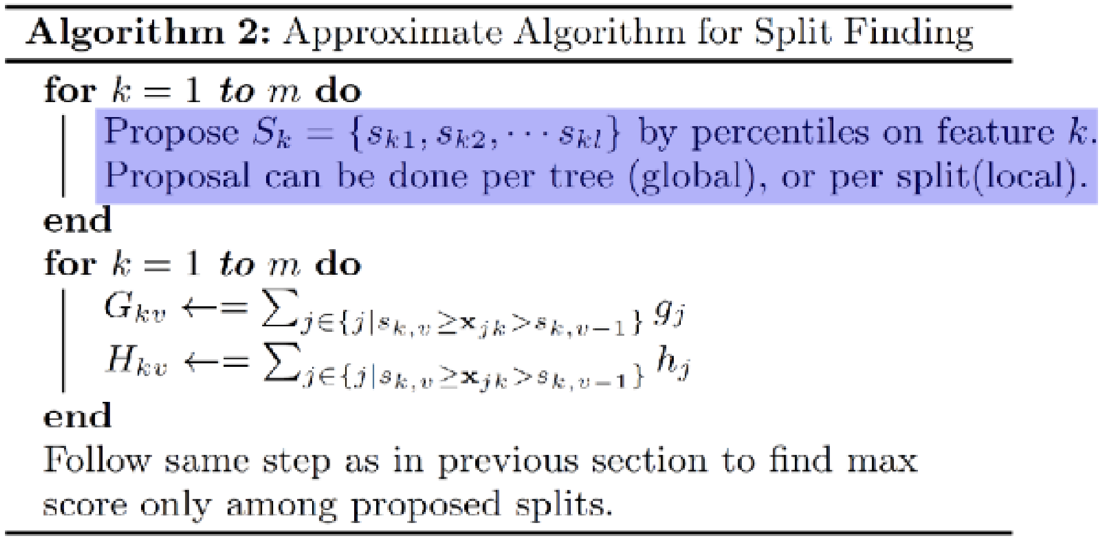


Algorithm 1: Exact Greedy Algorithm for Split Finding

```
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
```

- Exact Greedy Algorithm
- Find all possible split point greedily
- OOM Error, cannot be done under distributed settings

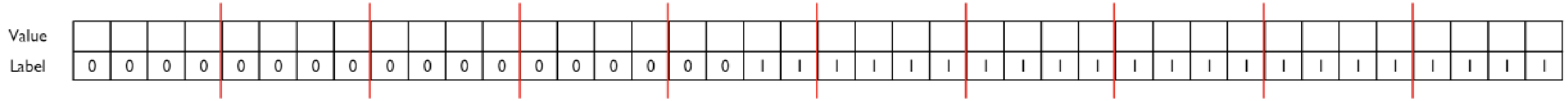
Split finding Algorithms



```
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.
```

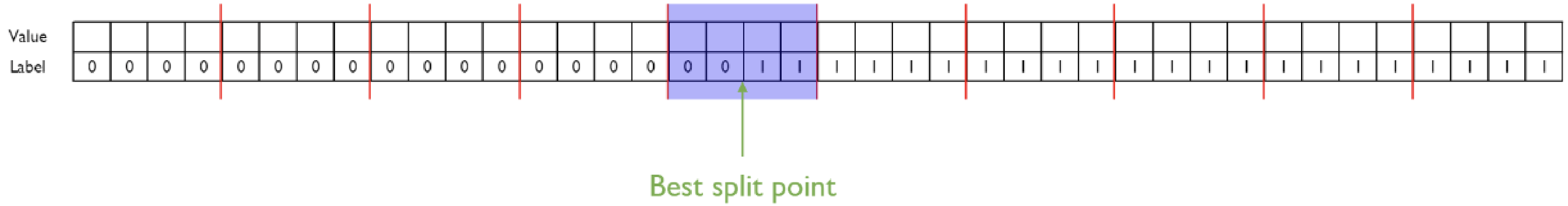
- Approximation Algorithm
- k: index of the variables (features), l: # of buckets
- 2 methods: per tree (global), per split (local)
- Epsilon as a hyperparameter

Split finding Algorithms



Value																																								
Label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

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



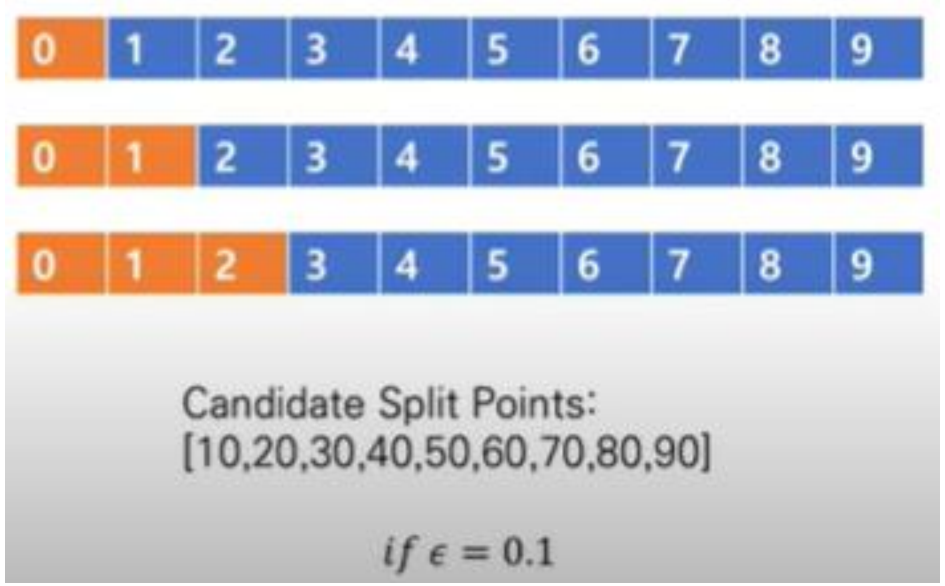
Value																																								
Label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

- Ascending order
- # of buckets = 10 i.e. epsilon = 0.1
- Exact greedy: 39 / approximation: 3 x 10 = 30

Split finding Algorithms

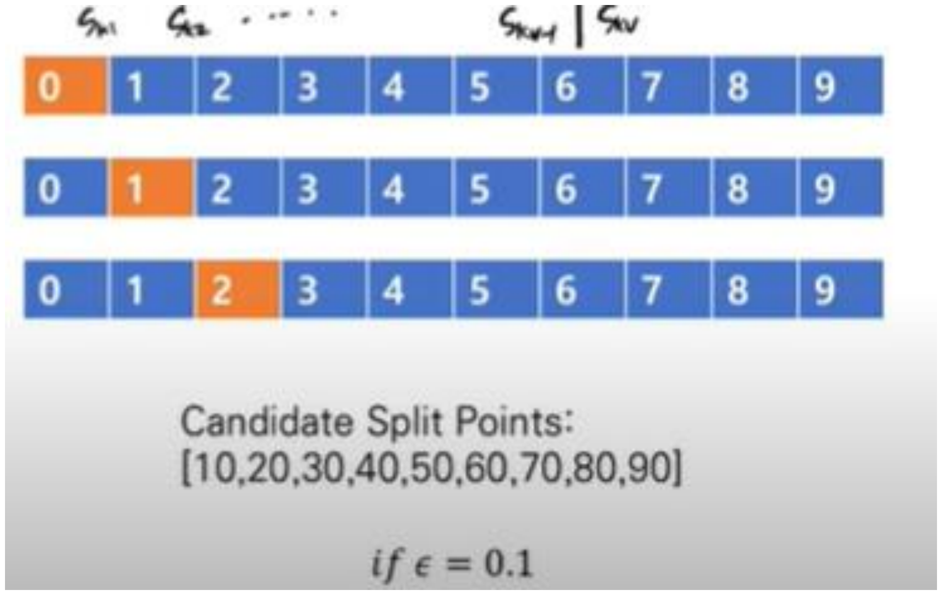
- Construct candidate split points w/ the percentile of feature distribution (epsilon)
- Update w/ G, H

Split finding Algorithms



Candidate Split Points:
[10,20,30,40,50,60,70,80,90]

$if\ \epsilon = 0.1$

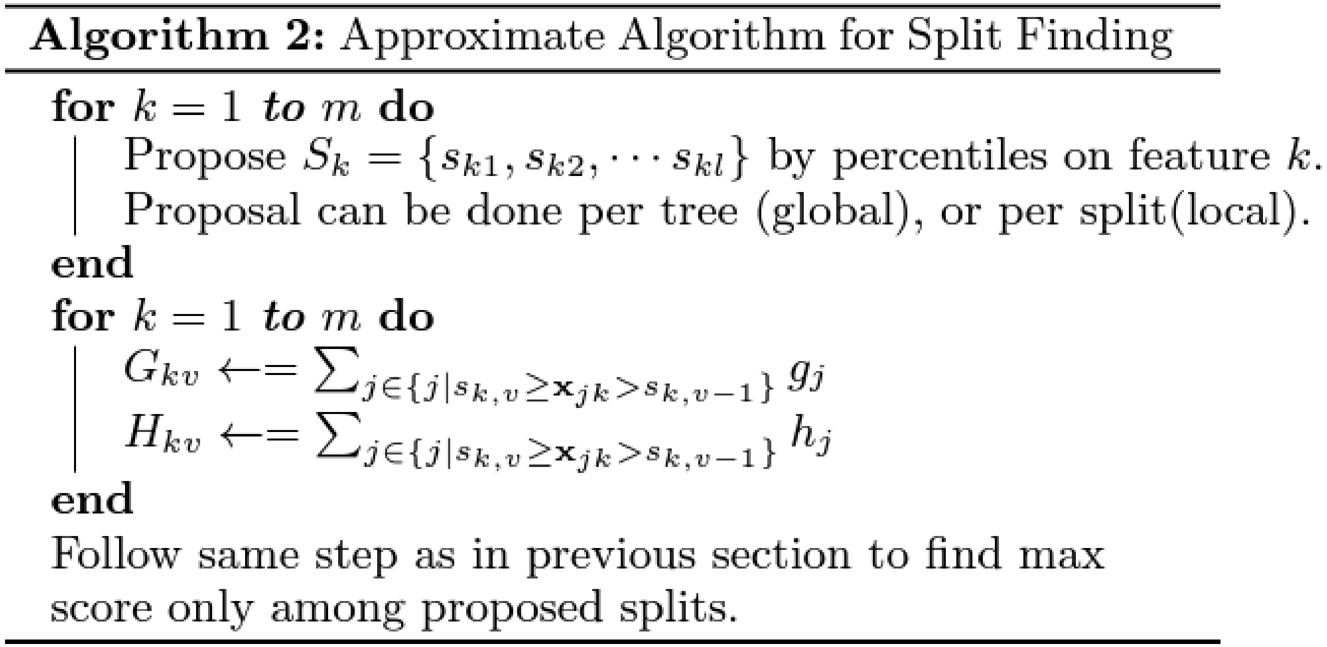


Candidate Split Points:
[10,20,30,40,50,60,70,80,90]

$if\ \epsilon = 0.1$

- For data points that are only “inside” the split points
- Enables “Parallelization”

Split finding Algorithms

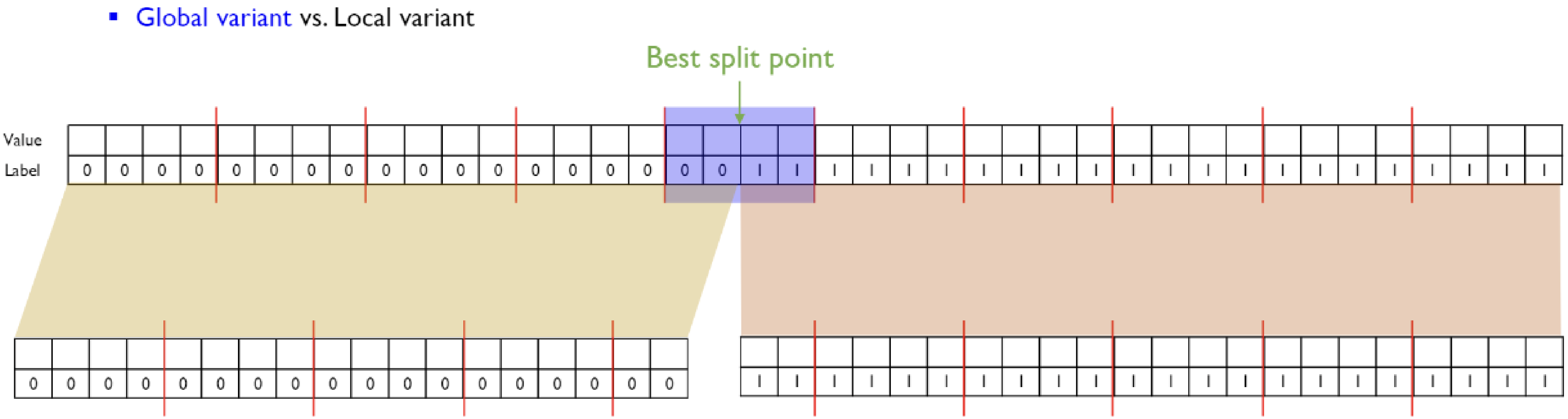


Algorithm 2: Approximate Algorithm for Split Finding

```
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.
```

Split finding Algorithms

- Global variant (per tree)

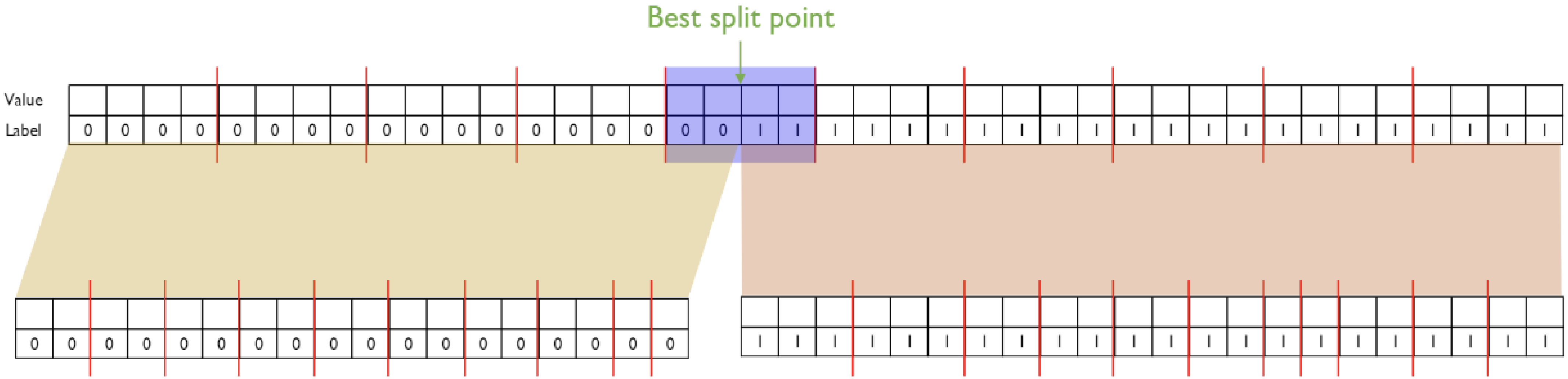


- Global variant vs. Local variant

Split finding Algorithms

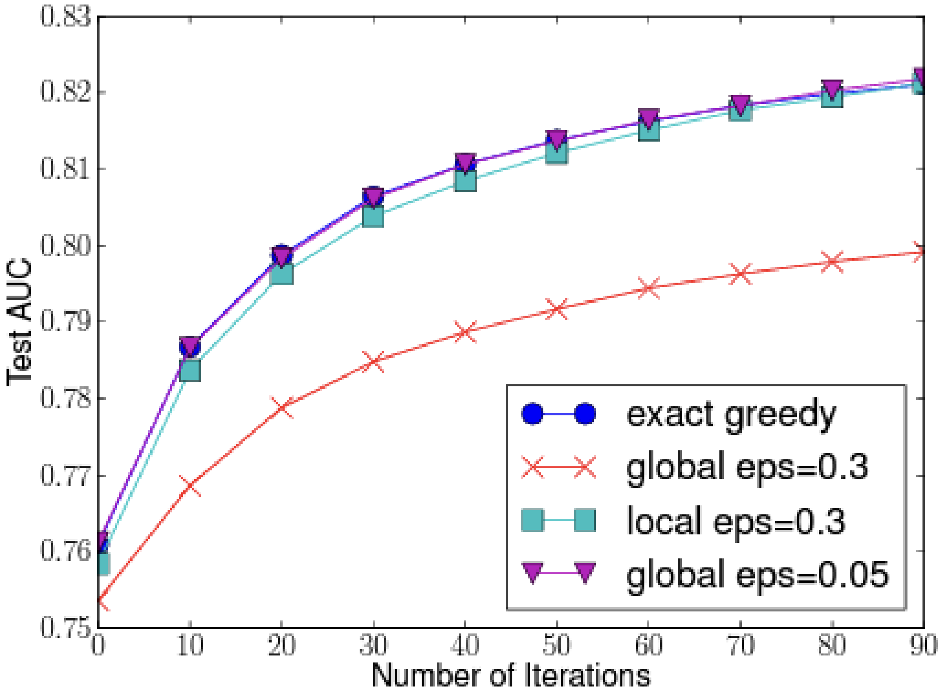
- Local variant (per split)
- Maintains the # buckets

  - Global variant vs. Local variant



Split finding Algorithms

- Does not mention which (global or local) algorithm is better
- Suggest the way to choose an appropriate epsilon
- Global: large epsilon does not work
- Local: large epsilon does work
- Epsilon: percentile parameter
- 1/epsilon ~ # of candidate split points



Split finding Algorithms

- Sketch Algorithm

Get a scheme of the Original Data Distribution w/ sample data sketch

- Quantile Sketch Algorithm

Get a scheme of the Original Data Distribution w/ sample data sketch & quantile

- Weighted Quantile Sketch Algorithm

Normal quantile: each quantile has the same # of data

Weighted quantile: each quantile has the same sum of weights (h_i)

Split finding Algorithms

- Sketch Algorithm, Quantile Sketch Algorithm, Weighted Quantile Sketch Algorithm
- These algorithms enable XGBoost to make a parallel computation

Sparsity-Aware Split Finding

- Missing values

① | Recap

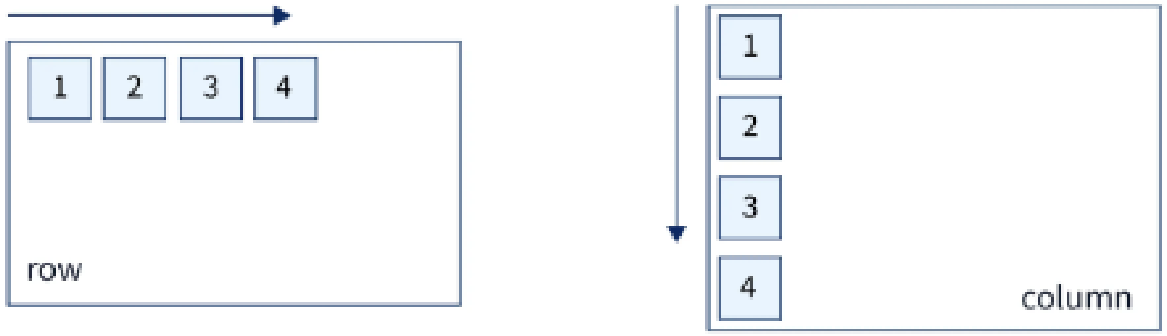
② | Split Finding Algorithms

③ | H/W Optimization

④ | Details

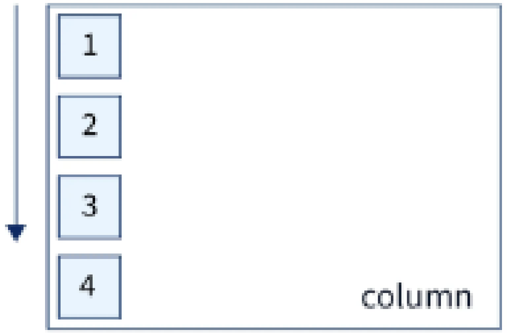
Hardware Optimization

- System Design for Efficient Computing
- Row Orientation vs. Column Orientation



- Row: Transactional Processing, ALL the columns are required
- Column: Only relevant columns are required

Hardware Optimization



- 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 “only once” before training and can be reused in later iterations. (by blocks)
- BLOCKS?

Hardware Optimization

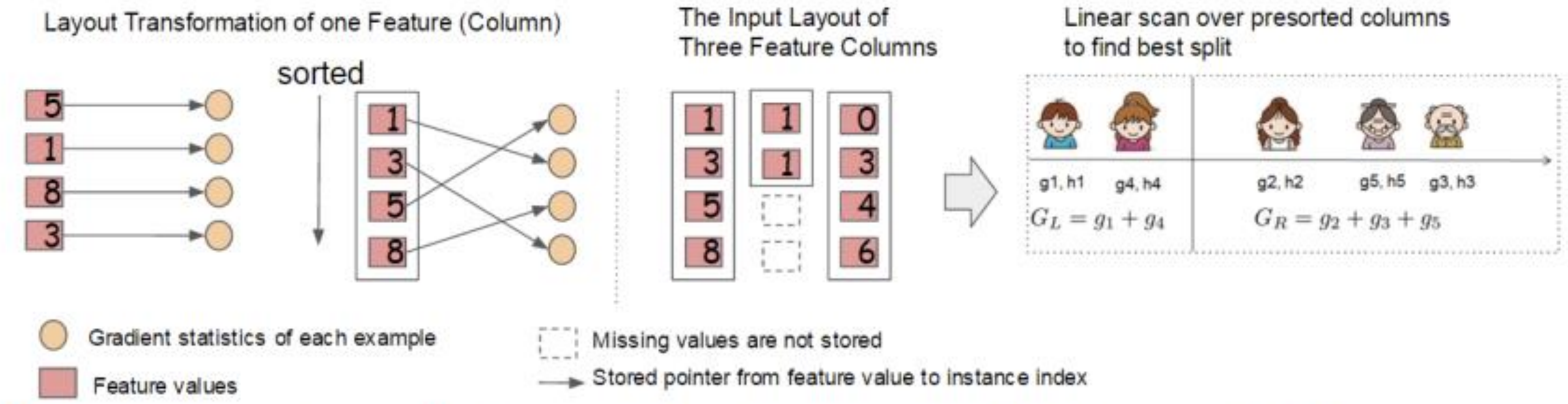
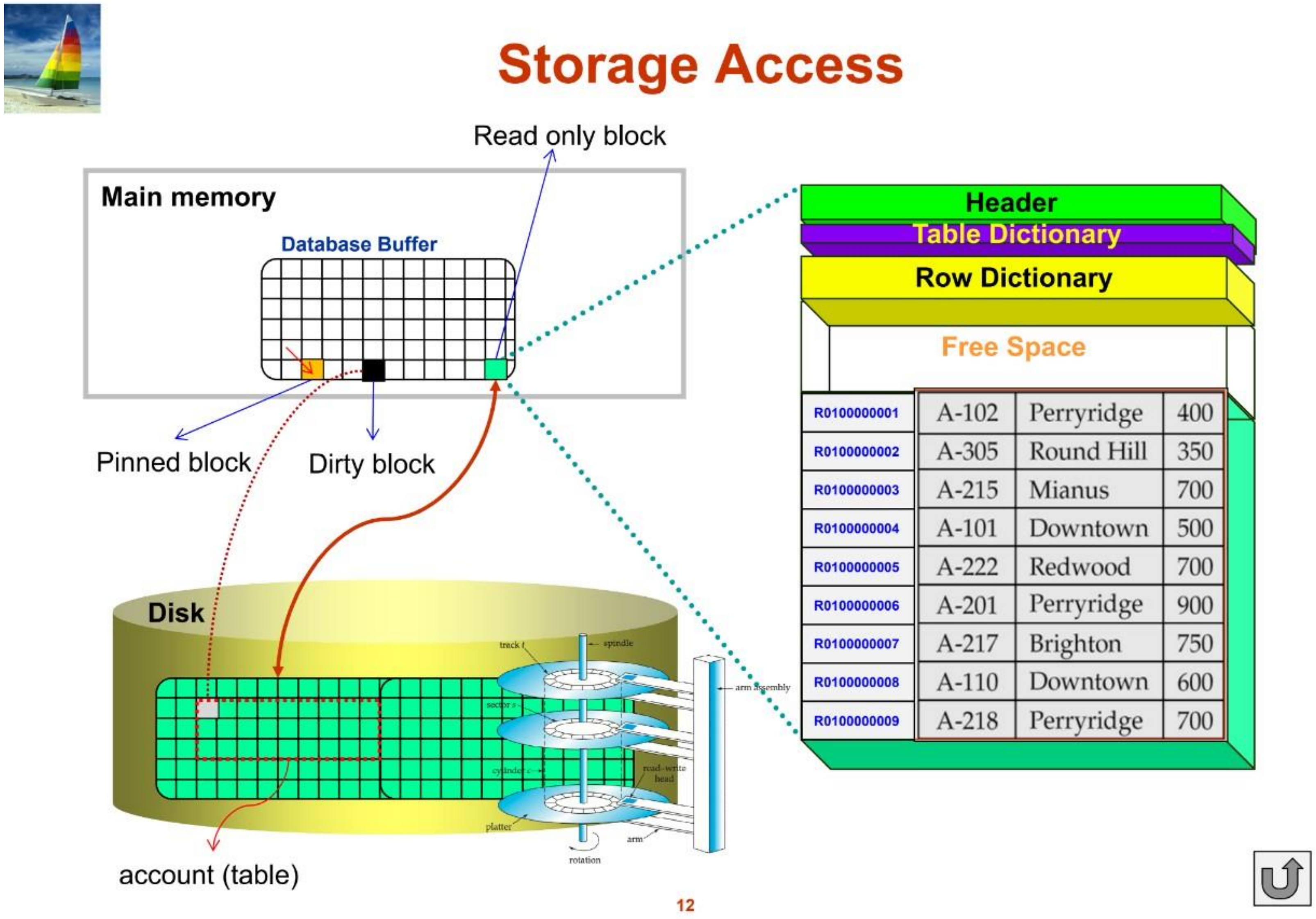

Layout Transformation of one Feature (Column)
sorted
5
1
8
3
1
3
5
8
The Input Layout of Three Feature Columns
1
3
5
8
1
1
0
3
4
6
Linear scan over presorted columns to find best split
g1, h1
g4, h4
g2, h2
g5, h5
g3, h3
$G_L = g_1 + g_4$
$G_R = g_2 + g_3 + g_5$
Gradient statistics of each example
Feature values
Missing values are not stored
Stored pointer from feature value to instance index

Figure 6: Block structure for parallel learning. Each column in a block is sorted by the corresponding feature value. A linear scan over one column in the block is sufficient to enumerate all the split points.

Hardware Optimization



Storage Access

R0100000001	A-102	Perryridge	400
R0100000002	A-305	Round Hill	350
R0100000003	A-215	Mianus	700
R0100000004	A-101	Downtown	500
R0100000005	A-222	Redwood	700
R0100000006	A-201	Perryridge	900
R0100000007	A-217	Brighton	750
R0100000008	A-110	Downtown	600
R0100000009	A-218	Perryridge	700

Hardware Optimization

- Cache-aware access

Cache -> Main Memory (M/M) -> Disk (SSD, HDD)

I/O Speed: Cache > M/M > Disk

- Block: a virtual unit of data
- It is important to choose the block size adequately
- Bigger the better -> No! (slow access)
- Smaller the better -> No! (slow computation in total)

Hardware Optimization

- Out-of-core computing

Utilize disk space to handle data that does not fit into M/M (block)

Reduce overhead and increase the throughput of disk I/O

- Block Compression (CSC)
- Block Sharding (partition)

When we can utilize more than 1 disks, save the data in each disk in different orders

Can increase the reading (throughput) of disk I/O

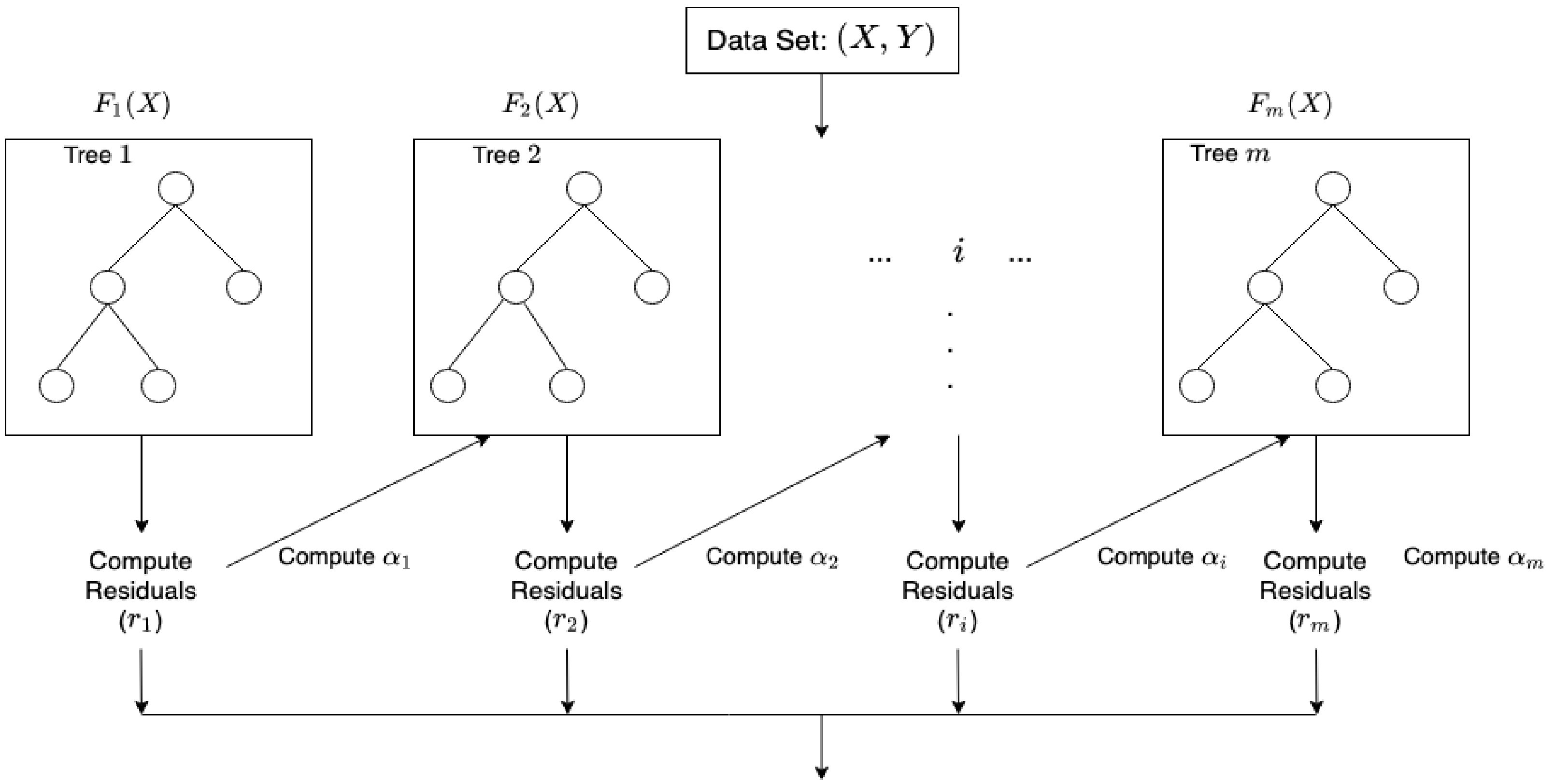
① | Recap

② | Split Finding Algorithms

③ | H/W Optimization

④ | Details

Details



$$F_m(X) \; = \; F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

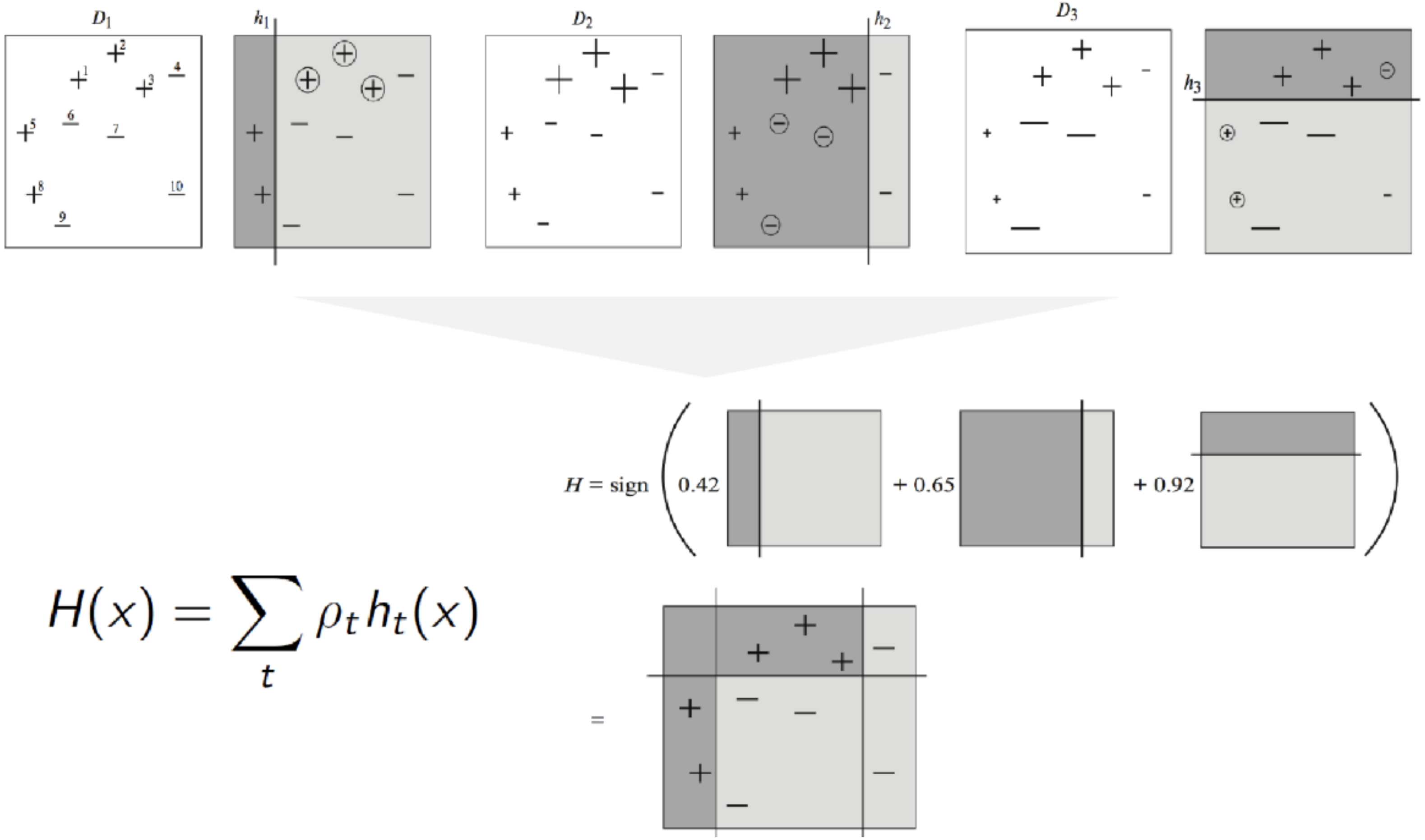
where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} \; = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

Details

- Models based on GBM: XGBoost, LightGBM (Microsoft), CatBoost (Categorical dataset)
- AdaBoost – w/o stacked trees, make a complex hyperplane

Details

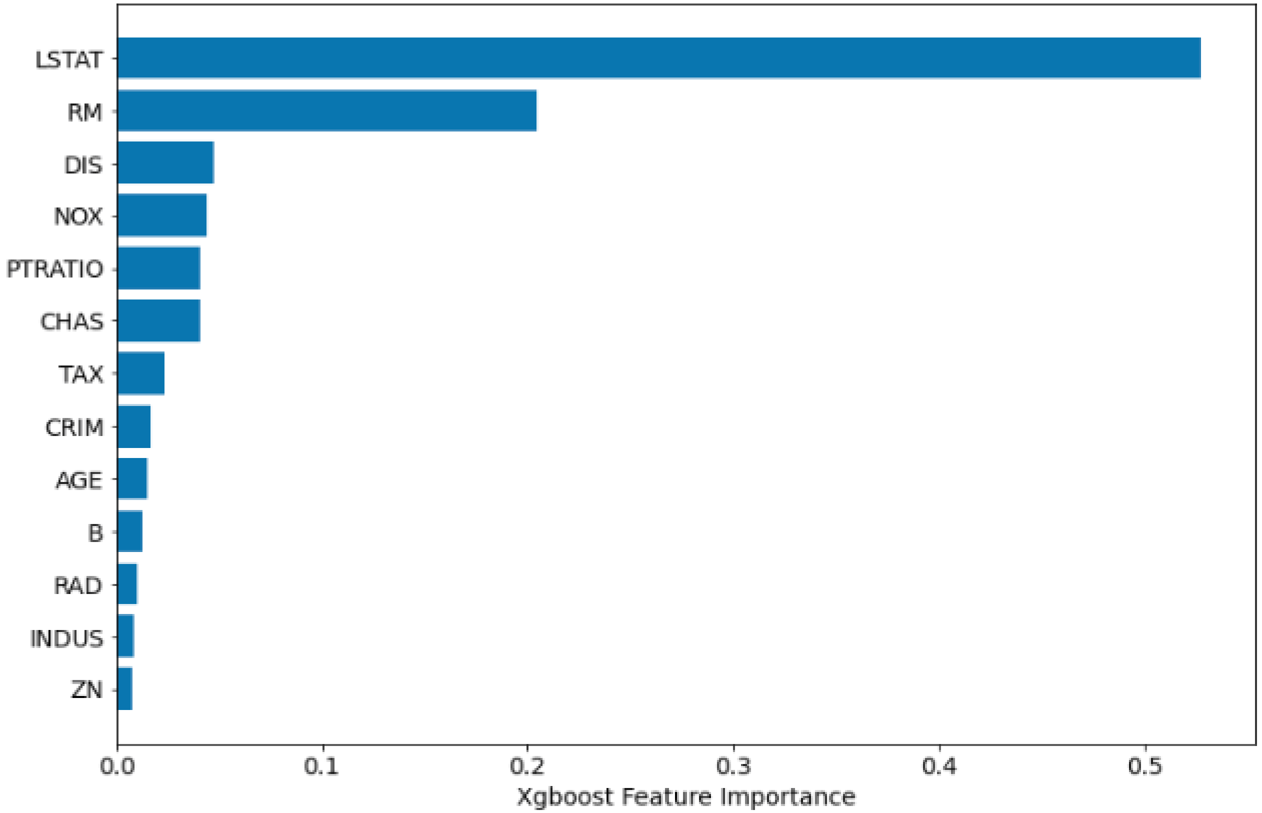
- Adaboost



$$H(x) = \sum_t \rho_t h_t(x)$$

Details

- Plot_importance



Details

- Plot_importance
- Information gain (default):

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

(Loss function before split) - (Loss function after split)

- get_score: # of the advent of variables (F score)
- Num_rounds: # of boosting
- Max_depth: depth of a tree
- Subsample [0, 1], ETA (learning rate), gamma (regularization, avoid overfitting)

Reference

- https://github.com/pilsung-kang/Business-Analytics-IME654-/tree/master/04%20Ensemble%20Learning
- XGBoost: A Scalable Tree Boosting System

Future Work

- Deep Learning for Tabular Data
- Machine Learning based models (in general), usually outperforms the deep learning model ‘only’ in terms of tabular data analysis
- But what if we need Deep Learning in tabular data analysis?

Happens when multi-modal data analysis is held

- Then let’s make Deep Tabular Learning model that utilizes the pros of Machine Learning techniques

Future Work

- But just simple Deep Learning? (DL usually requires a bunch of time to optimize)
- Then Meta Learning (Few-shot, fast meta-learner)

- When Meta Learning is combined w/ Tree-boosted ML techniques…?
- How to make ML techniques differentiable? … Gumbel-(soft)max techniques?