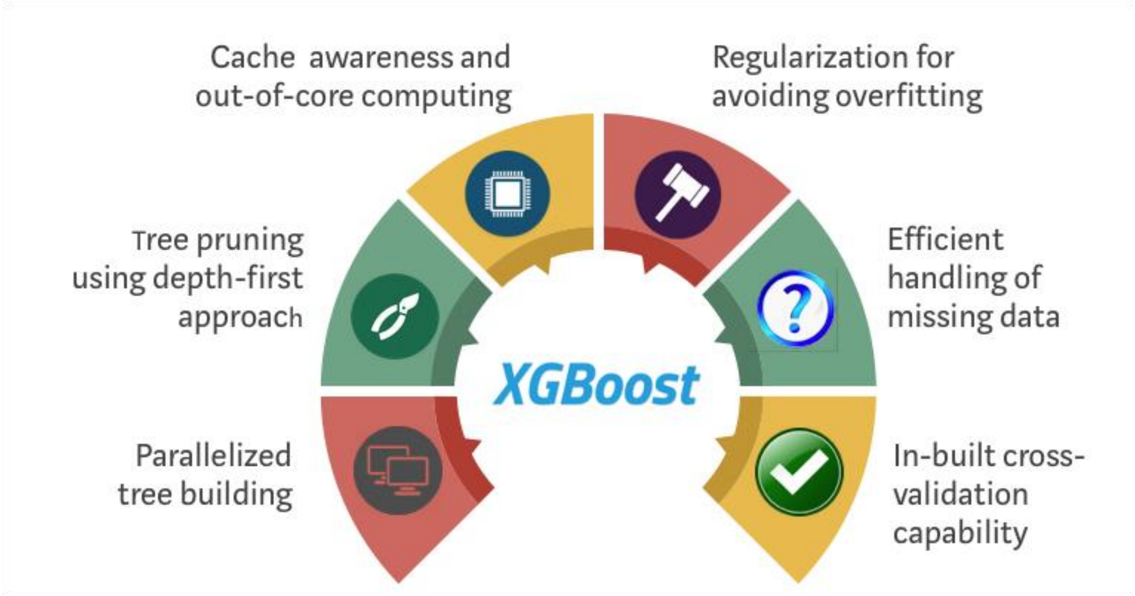


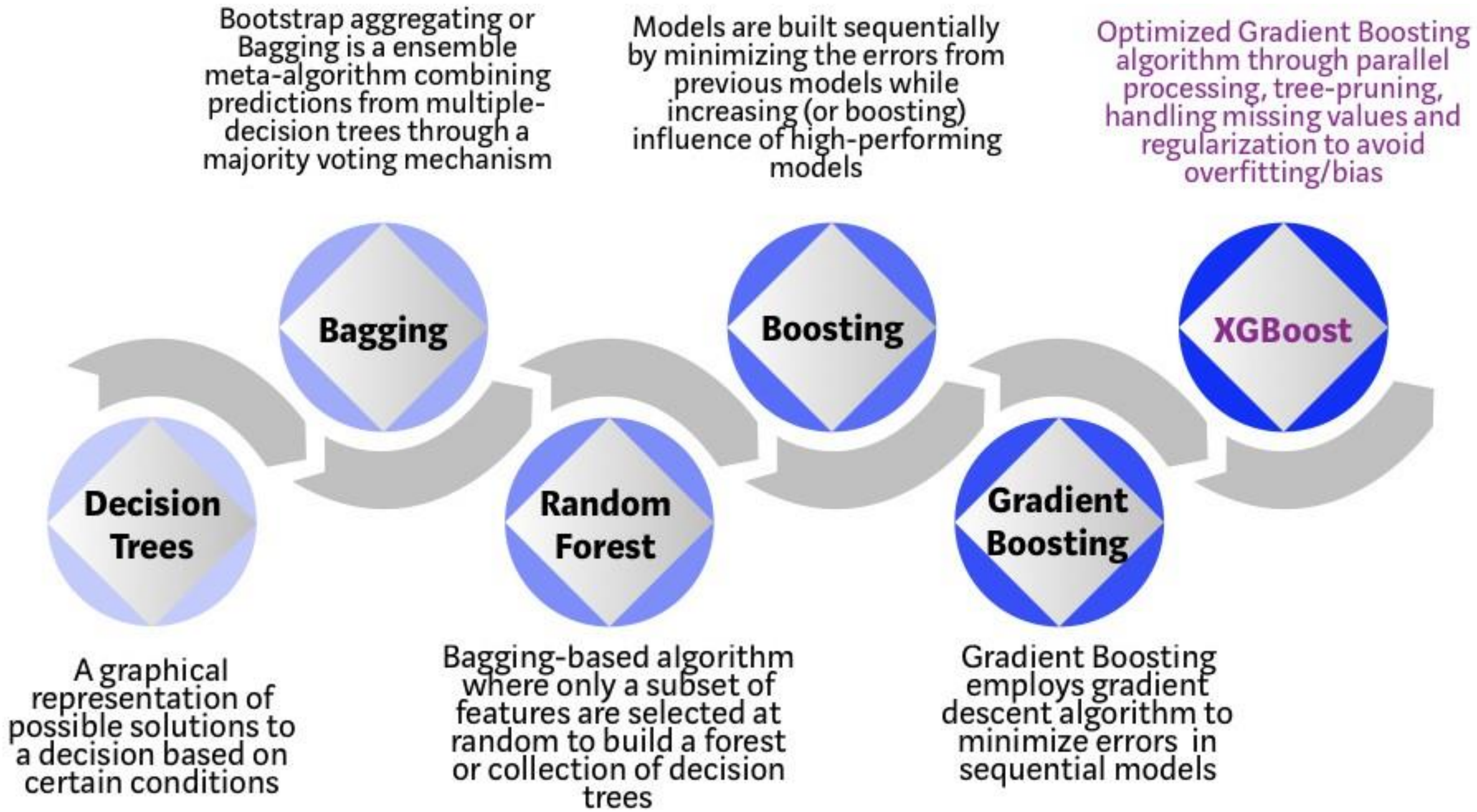
Ensemble Learning: XGBoost: A Scalable Tree Boosting System

Pilsung Kang
School of Industrial Management Engineering
Korea University

XGBoost

Chen, T., & Guestrin (2016)

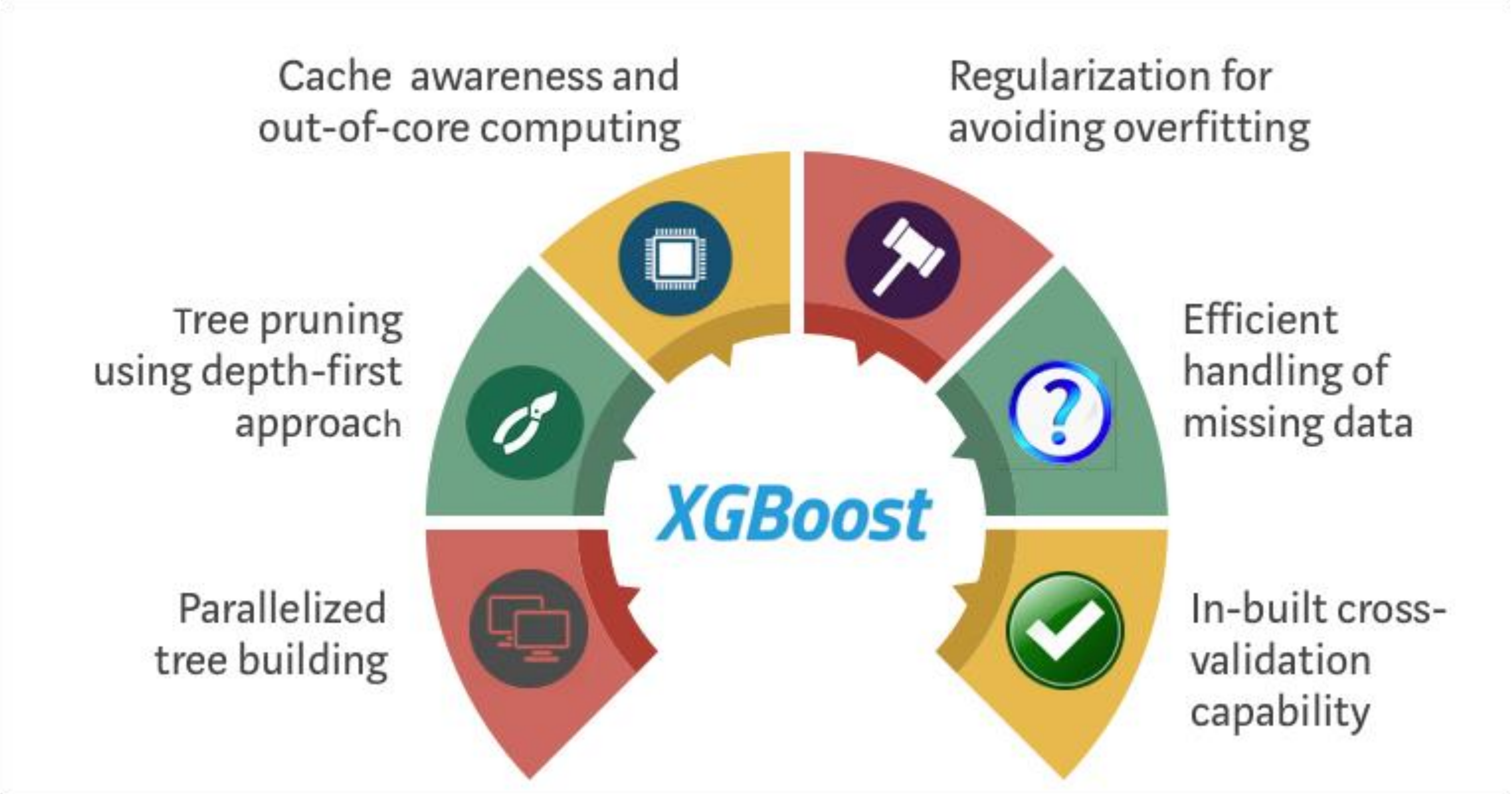
- XGBoost: A Scalable Tree Boosting System



https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

XGBoost

- XGBoost: An optimized version of GBM enabling



https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

XGBoost

- Split Finding Algorithm
  - ✓ Basic exact greedy algorithm
    - Pros: Always find the optimal split point because it enumerates over all possible splitting points greedily
    - Cons
      - Impossible to efficiently do so when the data does not fit entirely into memory
      - Cannot be done under a distributed setting

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

XGBoost

- Split Finding Algorithm
  - ✓ 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.

고려대학교 KOREA UNIVERSITY

DSBA Data Science & Business Analytics

XGBoost

- Split Finding 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

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

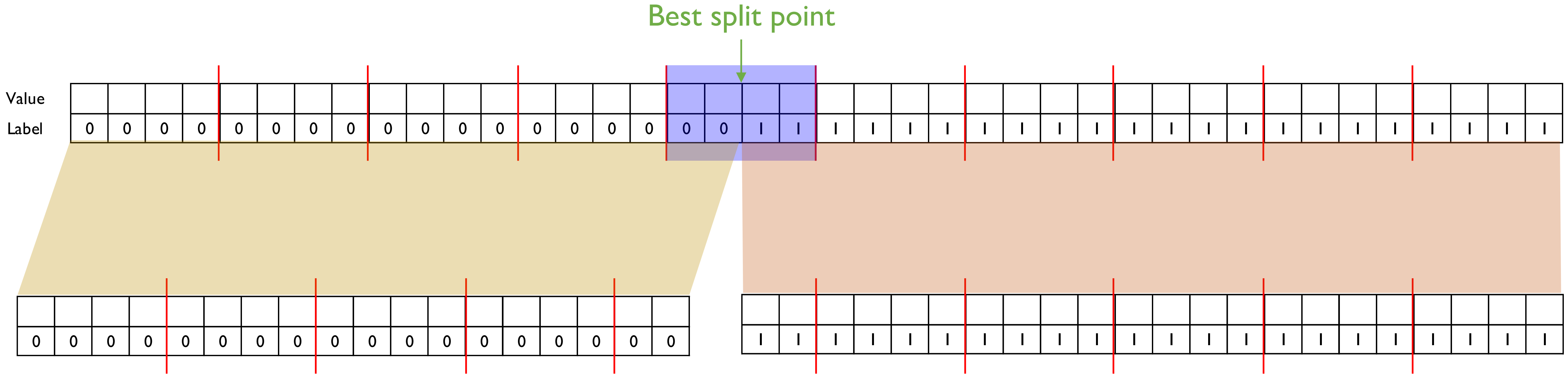
    - Compute the gradient for each bucket and find the best split

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

Best split point

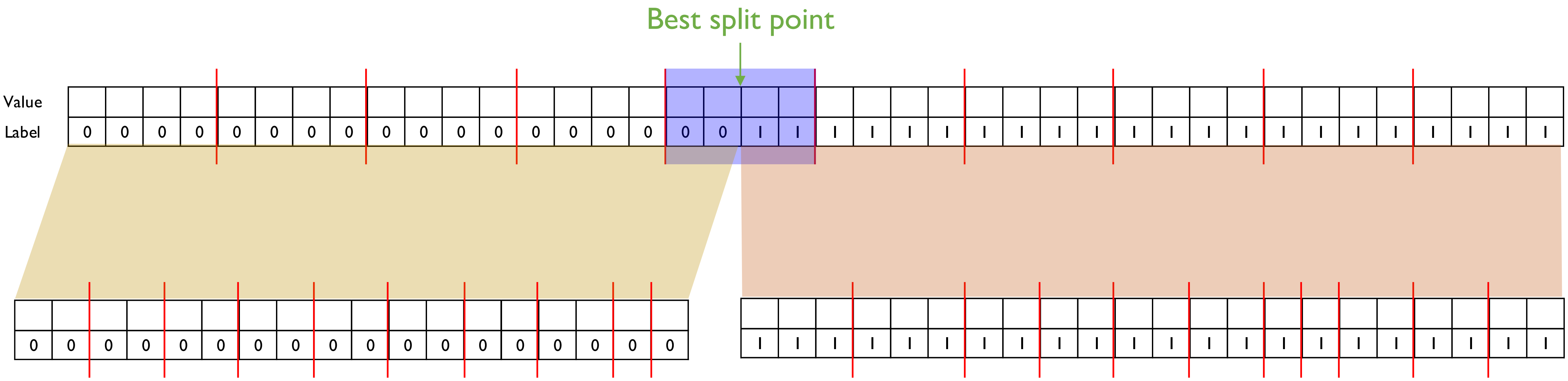
XGBoost

- Split Finding Algorithm
  - ✓ Approximate algorithm: an illustrative example
    - ▪ Global variant vs. Local variant



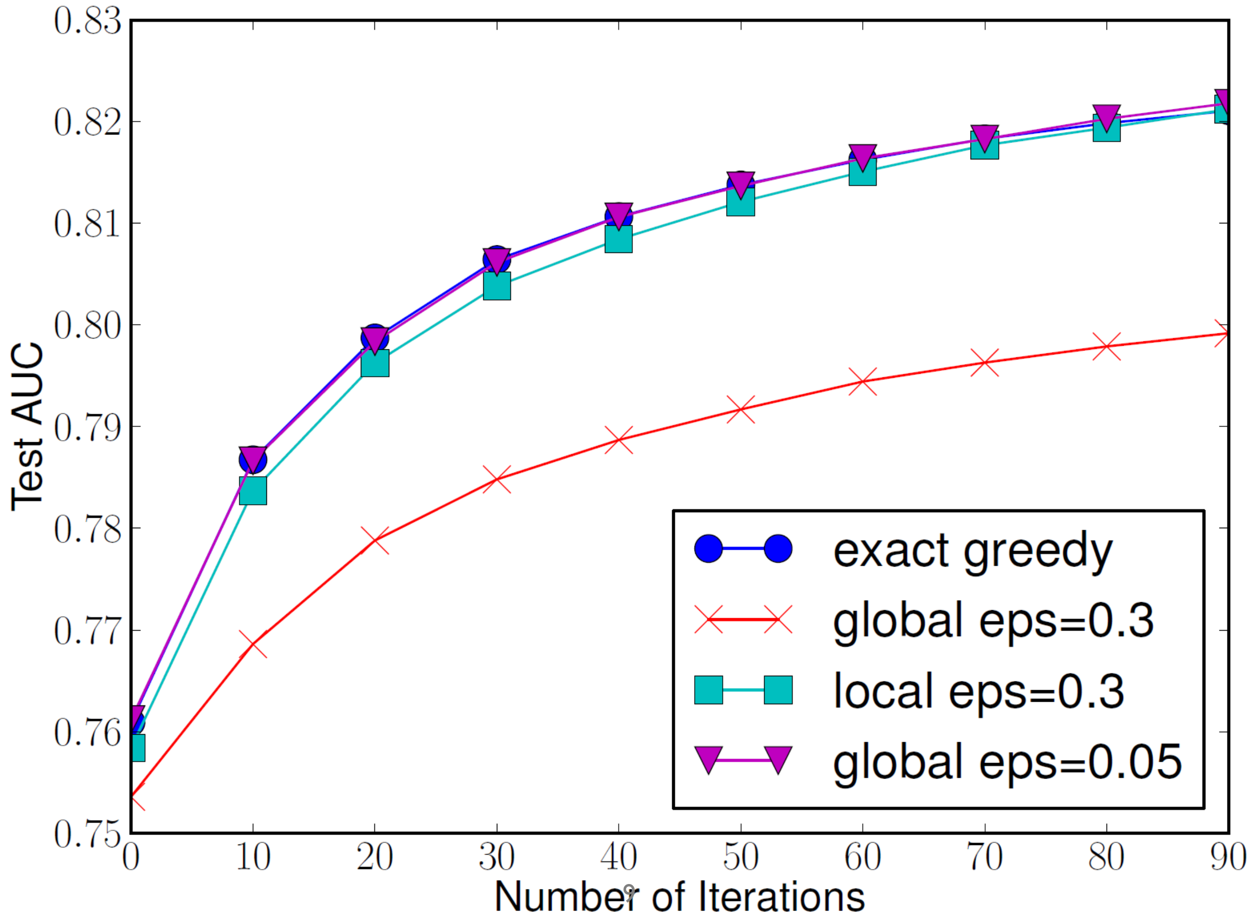
XGBoost

- Split Finding Algorithm
  - ✓ Approximate algorithm: an illustrative example
    - ▪ Global variant vs. Local variant

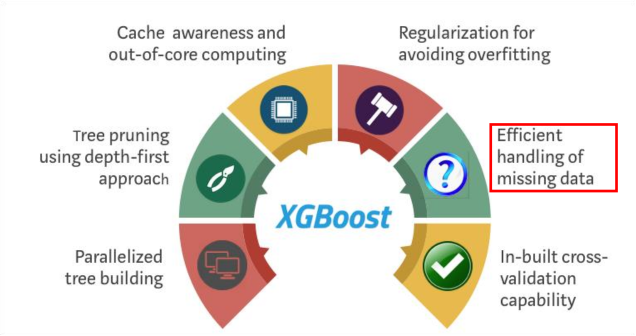


XGBoost

- Split Finding Algorithm
  - ✓ Approximate algorithm
    - Global proposal vs. Local proposal

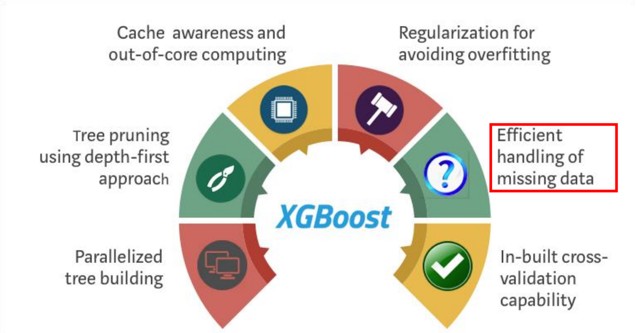


XGBoost



- Sparsity-Aware Split Finding
  - ✓ In many real-world problems, it is quite common for the input x to be sparse
    - presence of missing values in the data
    - frequent zero entries in the statistics
    - artifacts of feature engineering such as one-hot encoding
  - ✓ Solution: Set the default direction that is learned from the data

XGBoost



- 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

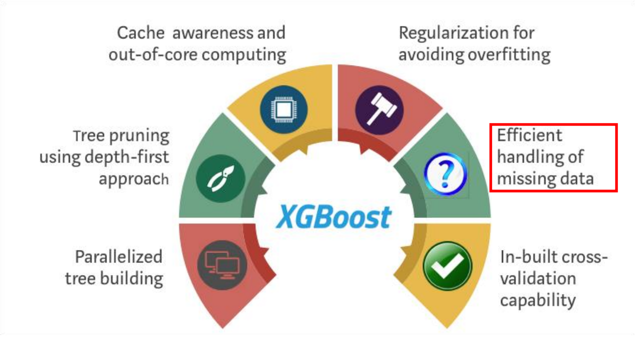
Value	1.3		1.1	0.2		1.9	0.5		1.5	1.8
Class	1	0	1	0	0	1	0	0	1	1

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

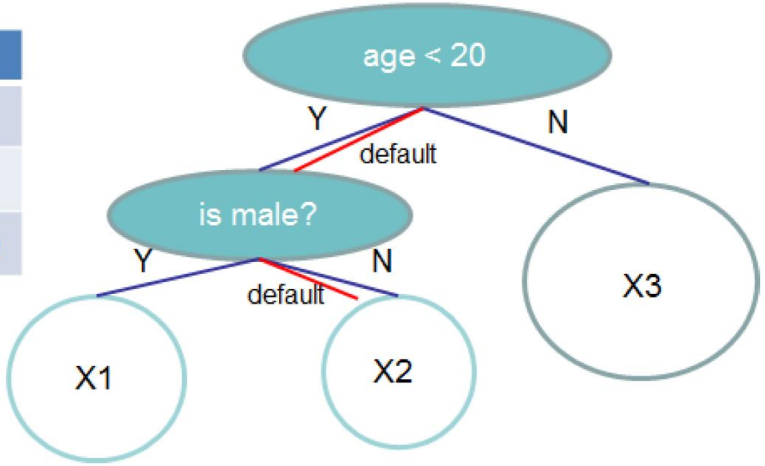
XGBoost

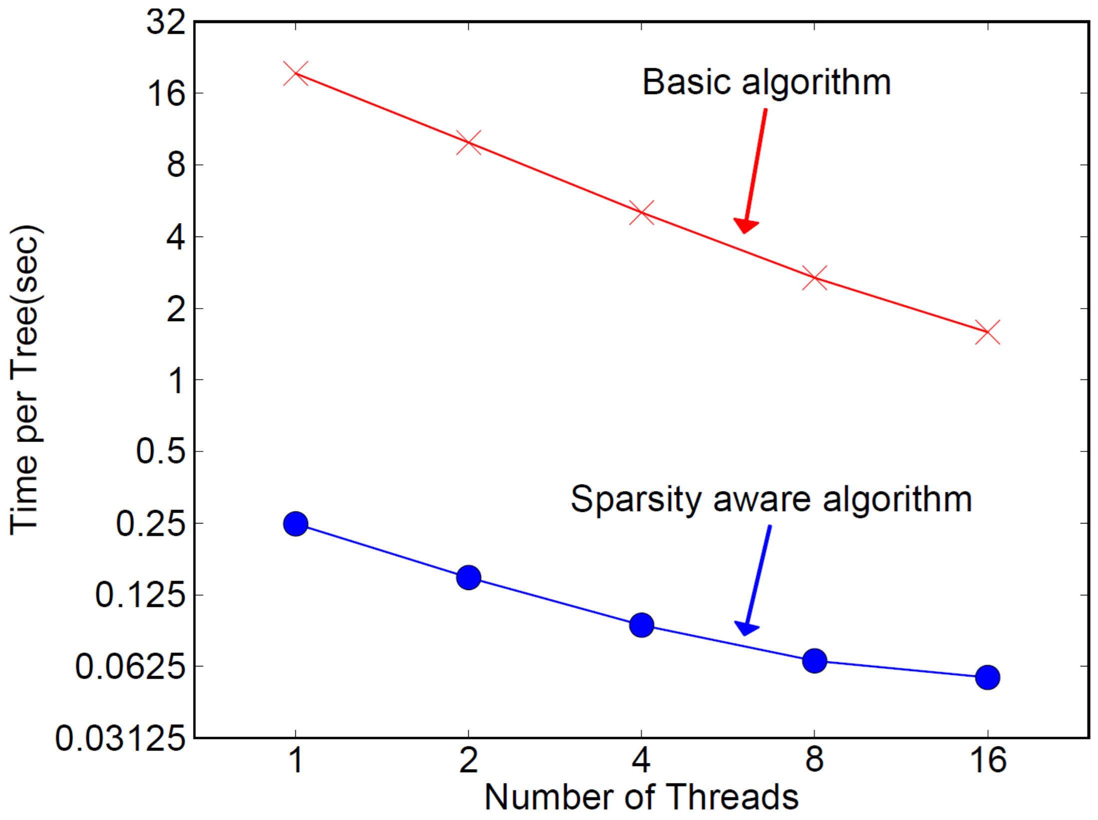


- Sparsity-Aware Split Finding

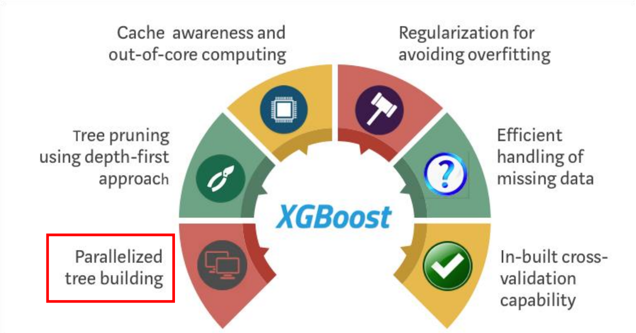
Data

Example	Age	Gender
X1	?	male
X2	15	?
X3	25	female

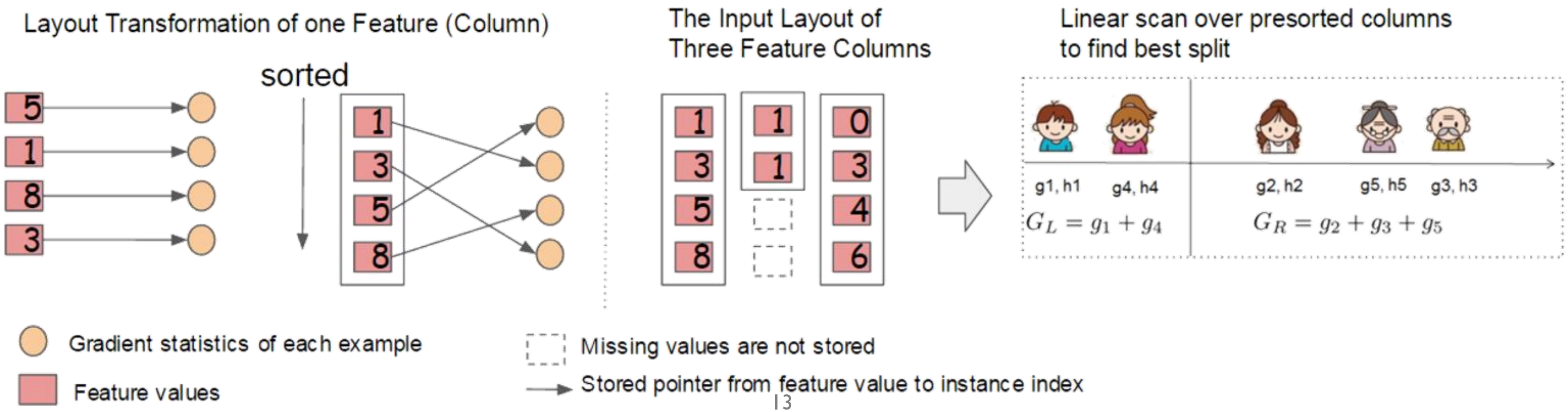




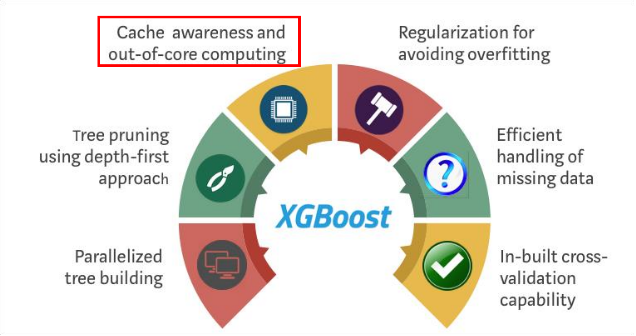
XGBoost



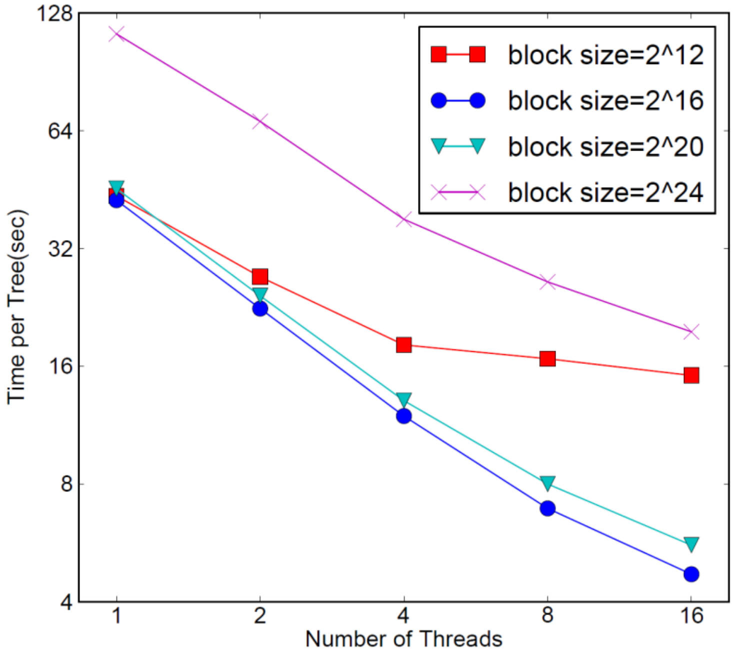
- System Design for Efficient Computing
  - ✓ The most time-consuming part of tree learning
    - to get the data into sorted order
  - ✓ XGBoost propose to store the data in in-memory units called block
    - Data in each block is stored in the compressed column (CSC) format, with each column sorted by the corresponding feature value
    - This input data layout only needs to be computed once before training and can be reused in later iterations.



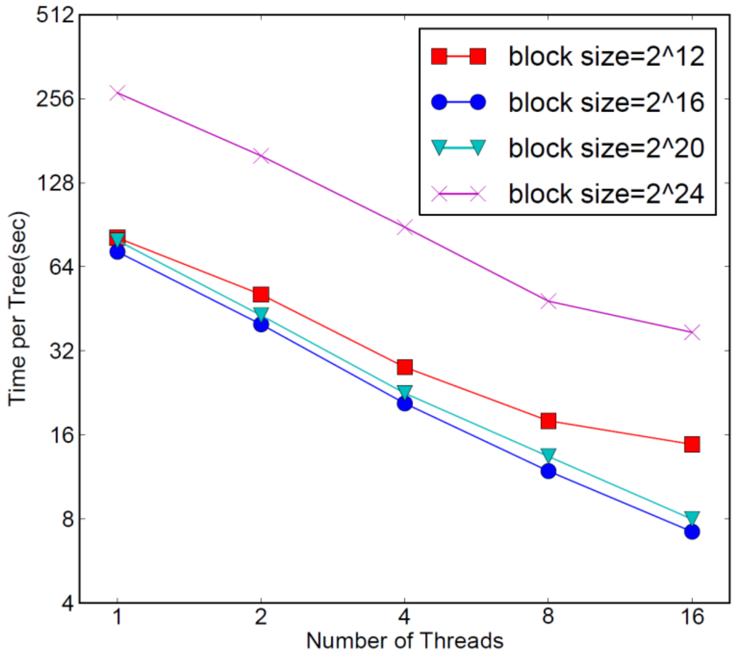
XGBoost



- System Design for Efficient Computing
  - ✓ Cache-aware access
    - For the exact greedy algorithm, we can alleviate the problem by a cache-aware prefetching algorithm
    - For approximate algorithms, we solve the problem by choosing a correct block size



(a) Allstate 10M

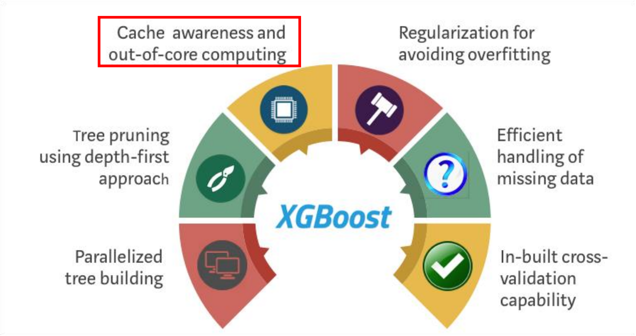


(b) Higgs 10M

고려대학교 KOREA UNIVERSITY

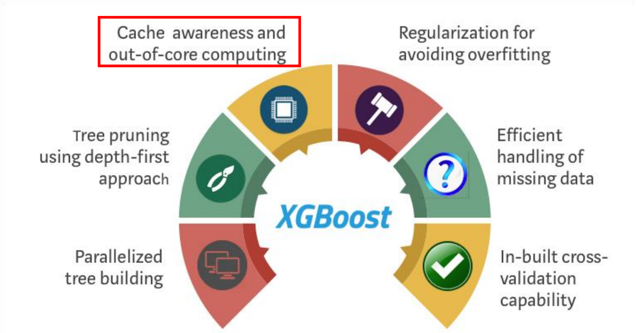
DSBA Data Science & Business Analytics

XGBoost



- System Design for Efficient Computing
  - ✓ Out-of-core computing
    - Besides processors and memory, it is important to utilize disk space to handle data that does not fit into main memory
    - To enable out-of-core computation, the data is divided into multiple blocks and store each block on disk
    - To enable out-of-core computation, we divide the data into multiple blocks and store each block on disk
    - It is important to reduce the overhead and increase the throughput of disk IO
  - ✓ Block Compression
    - The block is compressed by columns and decompressed on the fly by an independent thread when loading into main memory

XGBoost



- System Design for Efficient Computing
  - ✓ Block Compression
    - The block is compressed by columns and decompressed on the fly by an independent thread when loading into main memory
    - This helps to trade some of the computation in decompression with the disk reading cost
  - ✓ Block Sharding
    - A pre-fetcher thread is assigned to each disk and fetches the data into an in-memory buffer
    - The training thread then alternatively reads the data from each buffer
    - This helps to increase the throughput of disk reading when multiple disks are available

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

XGBoost

- Experiments

Table 2: Dataset used in the Experiments.

Dataset	n	m	Task
Allstate	10 M	4227	Insurance claim classification
Higgs Boson	10 M	28	Event classification
Yahoo LTRC	473K	700	Learning to Rank
Criteo	1.7 B	67	Click through rate prediction

Table 3: Comparison of Exact Greedy Methods with 500 trees on Higgs-1M data.

Method	Time per Tree (sec)	Test AUC
XGBoost	0.6841	0.8304
XGBoost (colsample=0.5)	0.6401	0.8245
scikit-learn	28.51	0.8302
R.gbm	1.032	0.6224

Table 4: Comparison of Learning to Rank with 500 trees on Yahoo! LTRC Dataset

Method	Time per Tree (sec)	NDCG@10
XGBoost	0.826	0.7892
XGBoost (colsample=0.5)	0.506	0.7913
pGBRT [22]	2.576	0.7915

XGBoost

- Experiments

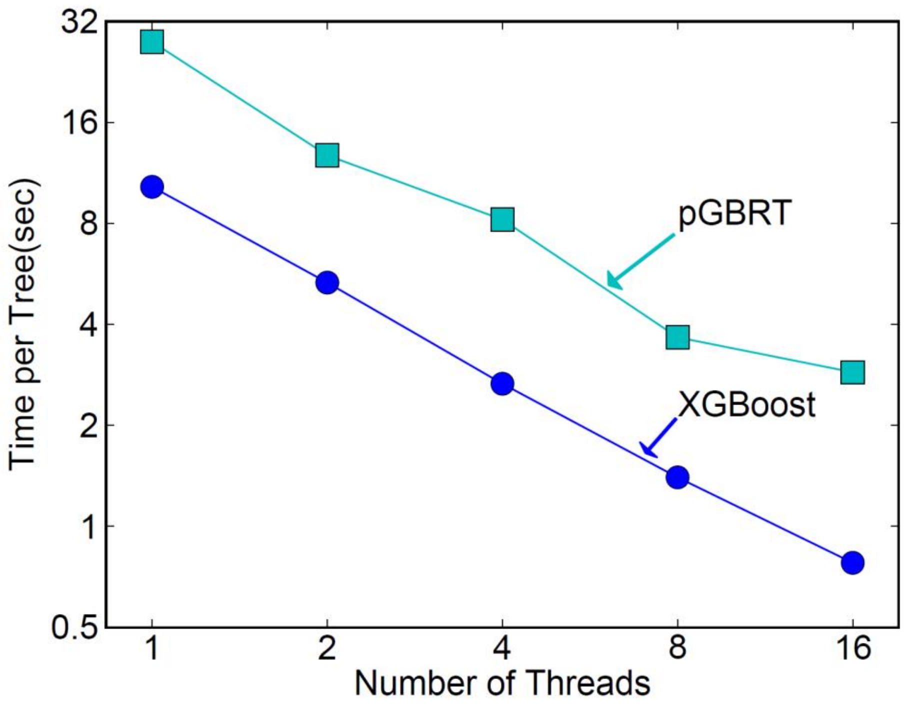


Figure 10: Comparison between XGBoost and pG-BRT on Yahoo LTRC dataset.

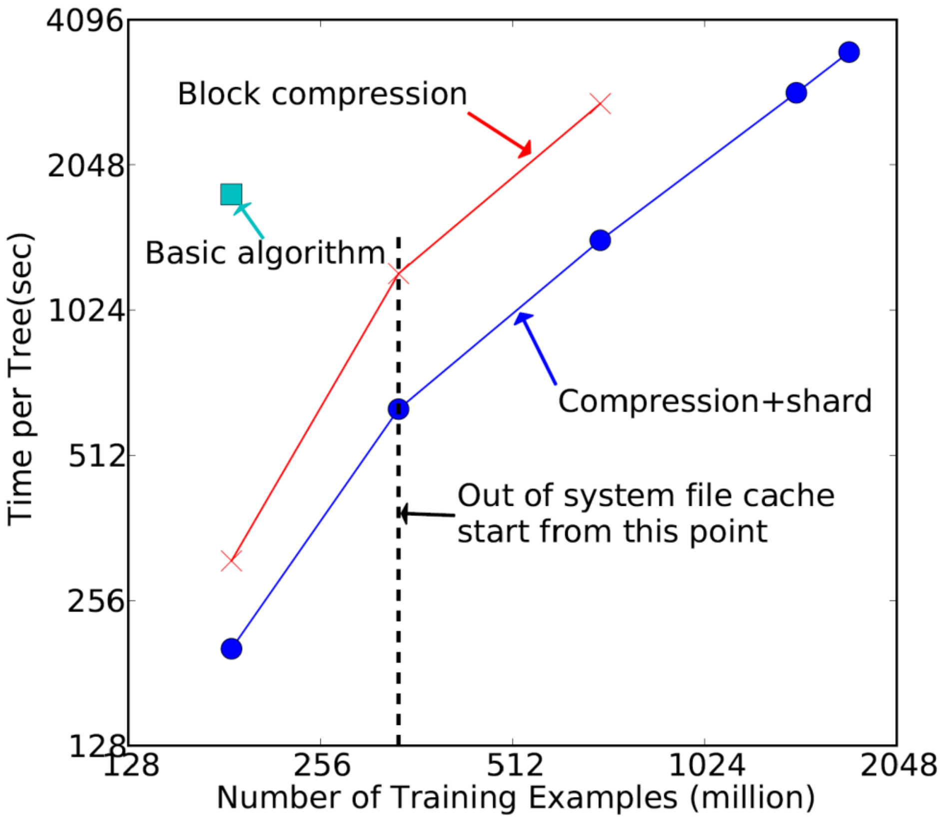
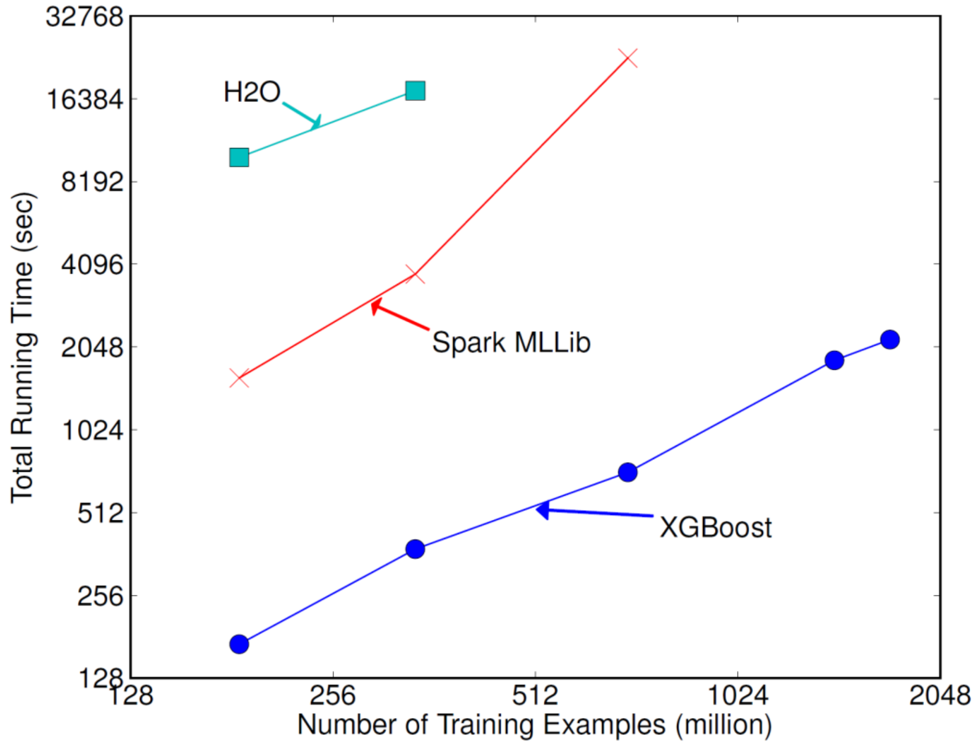


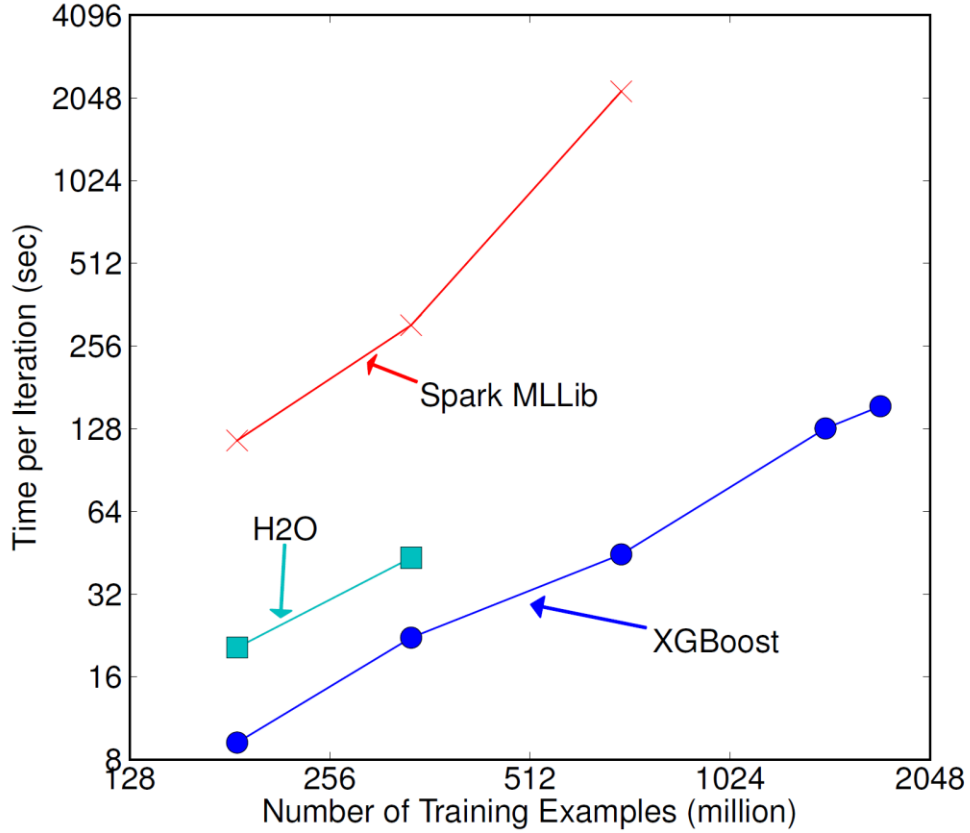
Figure 11: Comparison of out-of-core methods

XGBoost

- Experiments



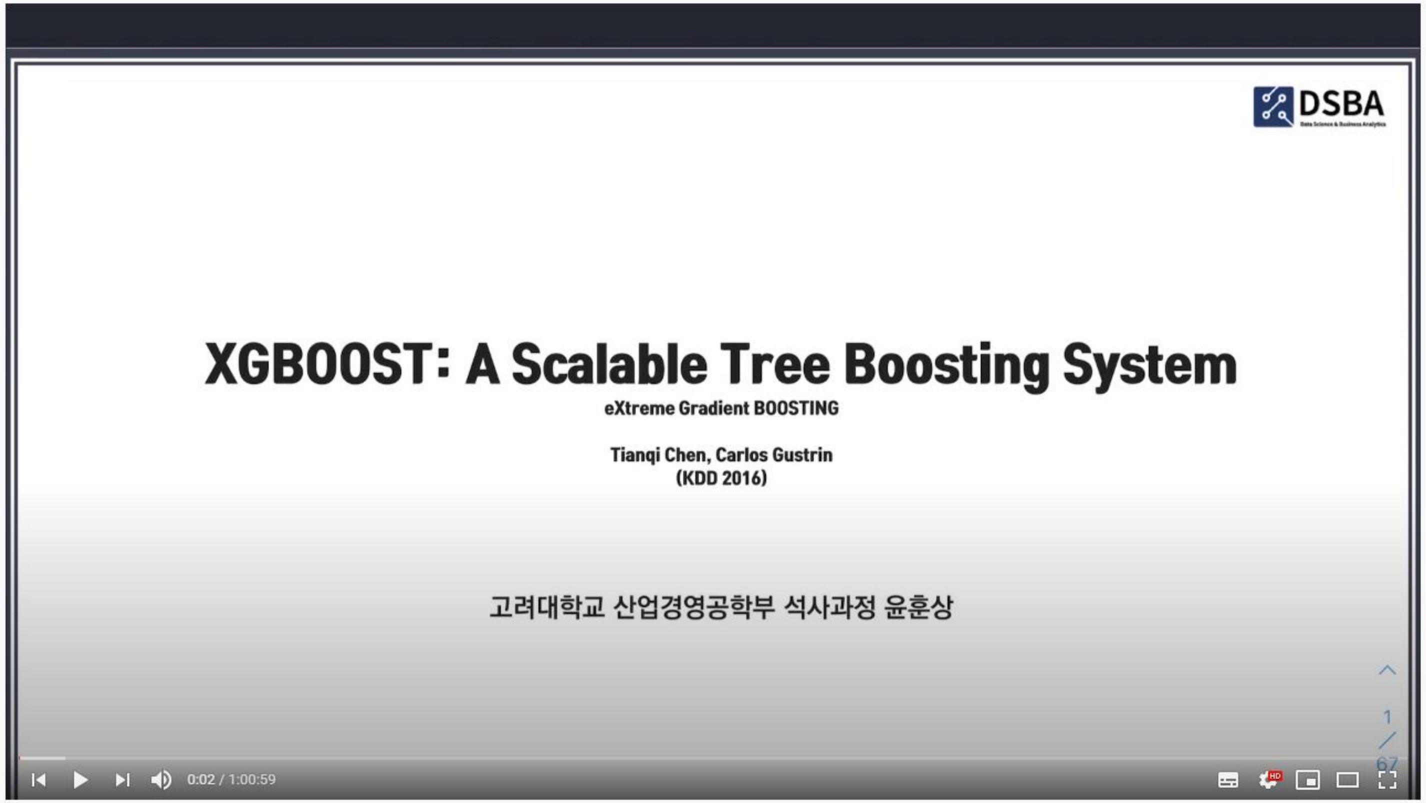
(a) End-to-end time cost include data loading



(b) Per iteration cost exclude data loading

XGBoost

- Optional resource
  - ✓ Youtube DSBA channel → [Papers You Must Read] playlist → [Paper Review] XGBoost: A Scalable Tree Boosting System (https://youtu.be/VkaZXGknN3g)




Q & A