高等机器学习

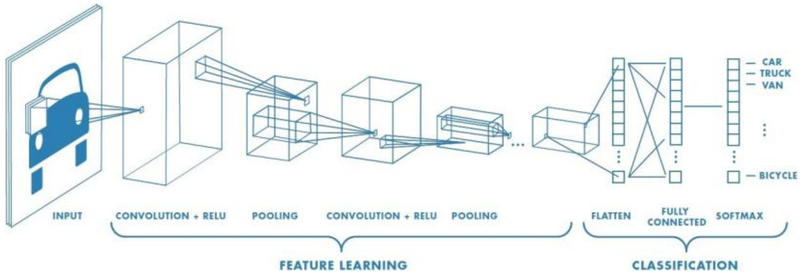
梯度提升树

施宇
微软研究院

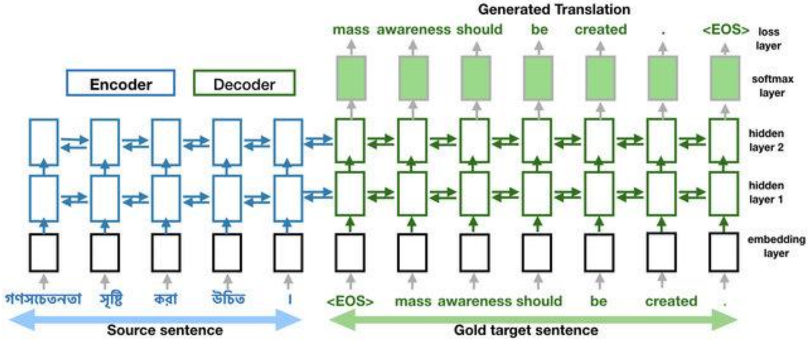
Overview

Success of Deep Learning

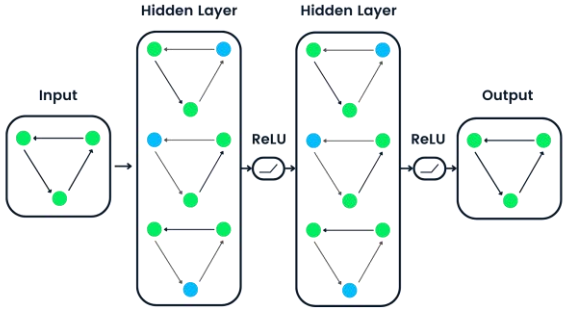
- Proper neural network structures for images/sequences



CNN for images

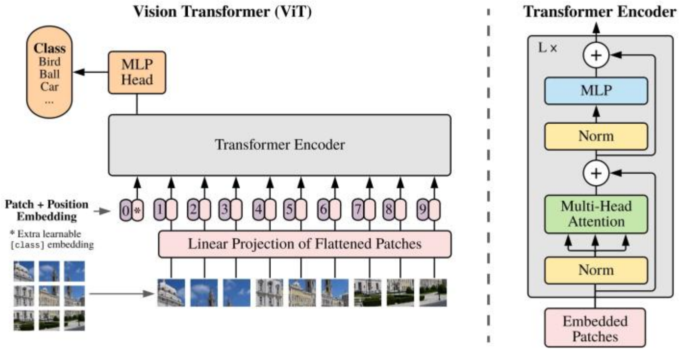


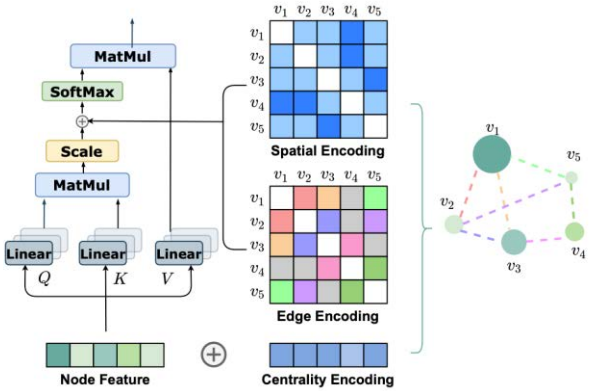
LSTM/Transformer for sequences



GNN for graphs

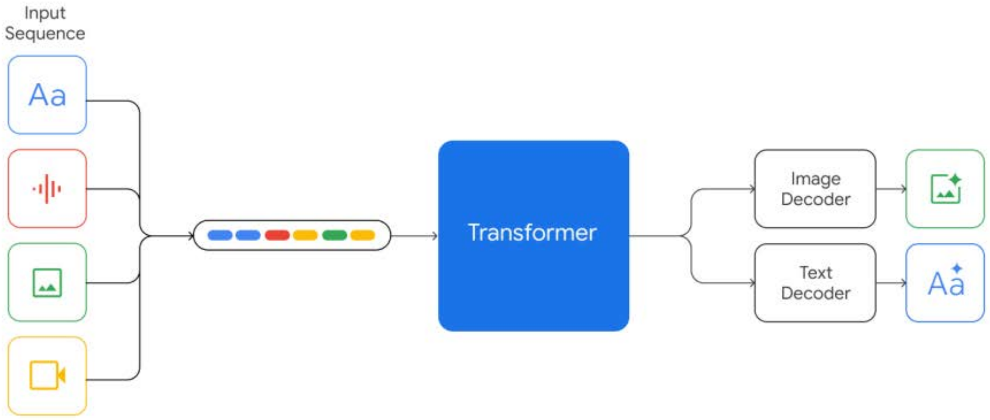
- Transformer has covered almost everything

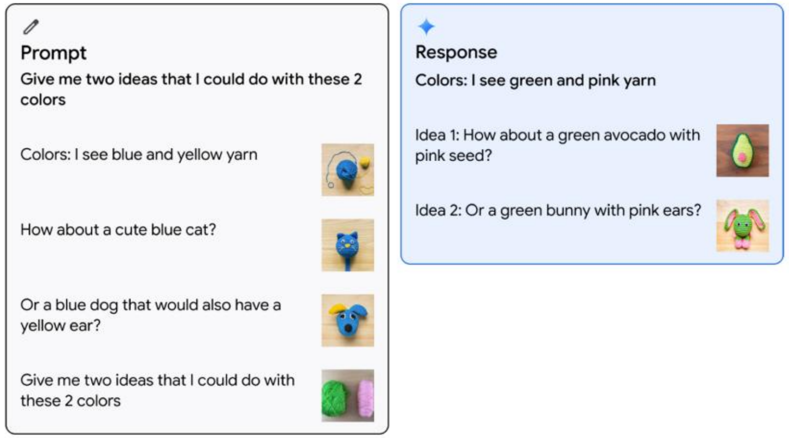


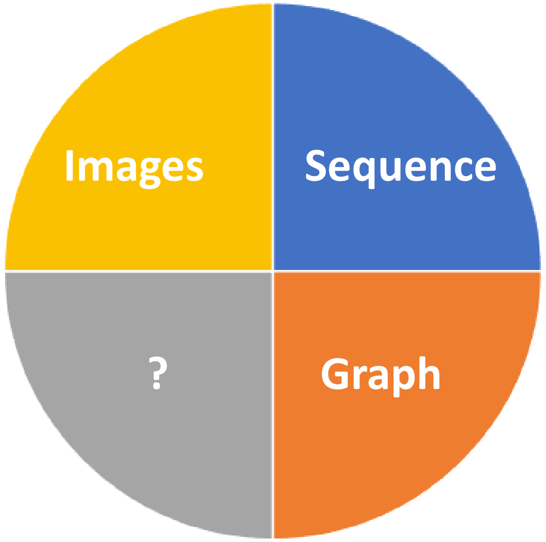


Success of Deep Learning

- Multiple modalities are unified with transformer-based models







Tabular Data – Outside Success of Deep Learning

- Tabular data: another very common data type, very diverse

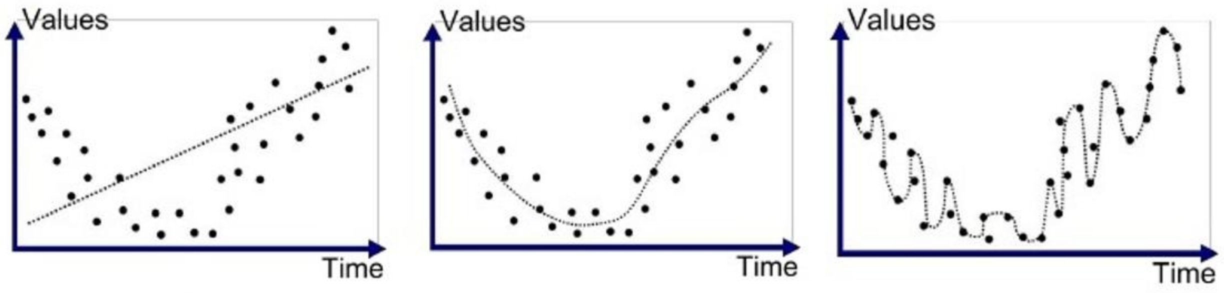
age	gender	occupation	use computer daily	like computer games
12	male	student	yes	yes
13	female	student	yes	no
36	female	nurse	yes	no
60	male	retired	no	no
57	female	retired	no	no

x y

- Diversity in types of features (attributes)
- Unknown dependency between columns
- Information is often incomplete
- Dataset size can vary from very small (hundreds) to very large (billion level)

No Free Lunch

- It is hard to pre-define a universal model/function for all kinds of tasks/data
    - Architecture of the model?
        - The types and correlations of features are various and unknown
    - Complexity of the model?
        - Simple function -> underfitting
        - Complex function -> overfitting
- Therefore, human-efforts is required in model design, for proper model architecture, and for trade-off between fitting and generalization

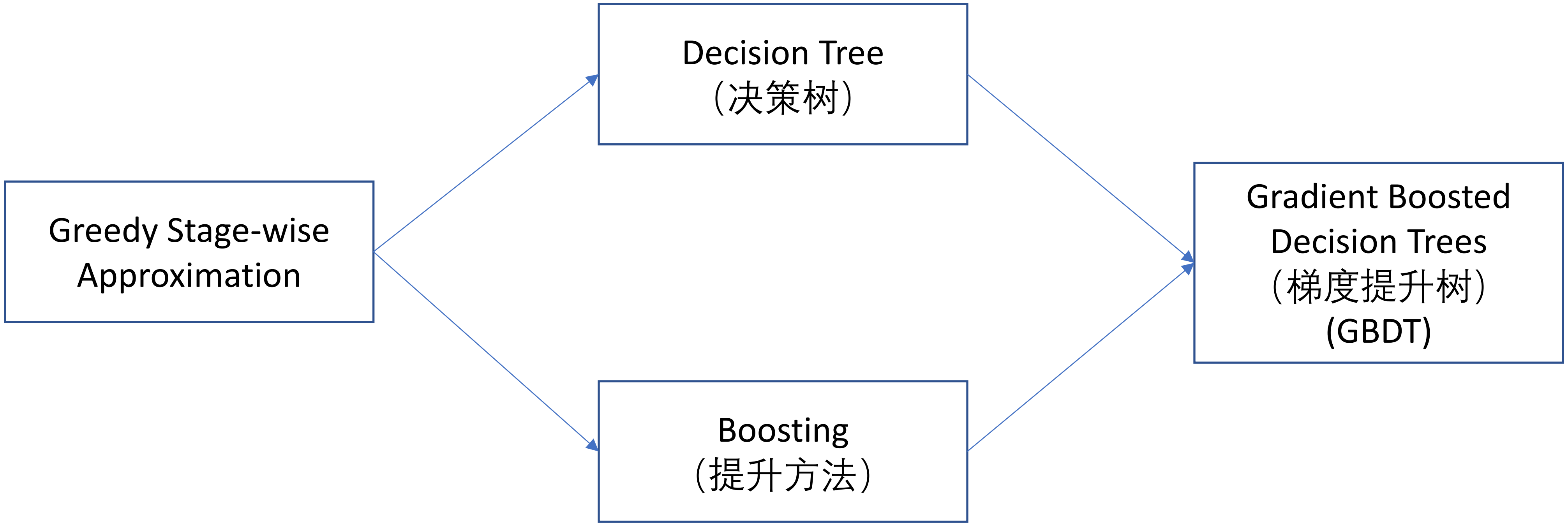


Underfitted Good Fit/Robust Overfitted

“Cheap Lunch”: Greedy Stage-wise Approximation

- Dynamic process: approximate the data and increase model complexity step by step, greedily
- Start from a simple model F_0
- Each time add a small model fraction f_m
  - $F_m(X) = F_{m-1}(X) + f_m(X)$, where $L(F_m(X), Y) < L(F_{m-1}(X), Y)$
- Can stop when “good fit”, e.g., by early-stopping on validation set
- Both Boosting and Decision Tree are in this category

Greedy Stage-wise Approximation



Best Solution for Tabular Data Learning

- GBDT tools

dmlc XGBoost   LightGBM   Yandex CatBoost

- Winning solutions of many tabular data learning tasks

LightGBM is used in many winning solutions, but this table is updated very infrequently.

Place	Competition	Solution	Date
1st	M5 Forecasting - Uncertainty	link	2020.7
3rd	M5 Forecasting - Uncertainty	link	2020.7
3rd	ALASKA2 Image Steganalysis	link	2020.7
1st	M5 Forecasting - Accuracy	link	2020.6
2nd	COVID19 Global Forecasting (Week 5)	link	2020.5
3rd	COVID19 Global Forecasting (Week 5)	link	2020.5
1st	COVID19 Global Forecasting (Week 4)	link	2020.5

XGBoost is extensively used by machine learning practitioners to create state of art data science solutions, this is a list of machine learning winning solutions with XGBoost. Please send pull requests if you find ones that are missing here.

- Bishwarup Bhattacharjee, 1st place winner of Allstate Claims Severity conducted on December 2016. Link to discussion
- Benedikt Schifferer, Gilberto Titericz, Chris Deotte, Christof Henkel, Kazuki Onodera, Jiwei Liu, Bojan Tunguz, Even Oldridge, Gabriel De Souza Pereira Moreira and Ahmet Erdem, 1st place winner of Twitter RecSys Challenge 2020 conducted from June,20-August,20. GPU Accelerated Feature Engineering and Training for Recommender Systems
- Eugene Khvedchenya,Jessica Fridrich, Jan Butora, Yassine Yousfi 1st place winner in ALASKA2 Image Steganalysis. Link to discussion
- Dan Ofer, Seffi Cohen, Noa Dagan, Nurit, 1st place in WiDS Datathon 2020. Link to discussion
- Chris Deotte, Konstantin Yakovlev 1st place in IEEE-CIS Fraud Detection. Link to discussion
- Giba, Lucasz, 1st place winner in Santander Value Prediction Challenge organized on August,2018. Solution discussion and code

Outline

- Decision Tree
- Boosting
- GBDT (Gradient Boosted Decision Trees)
- Deep Learning for Tabular Data
- GBDT Practices

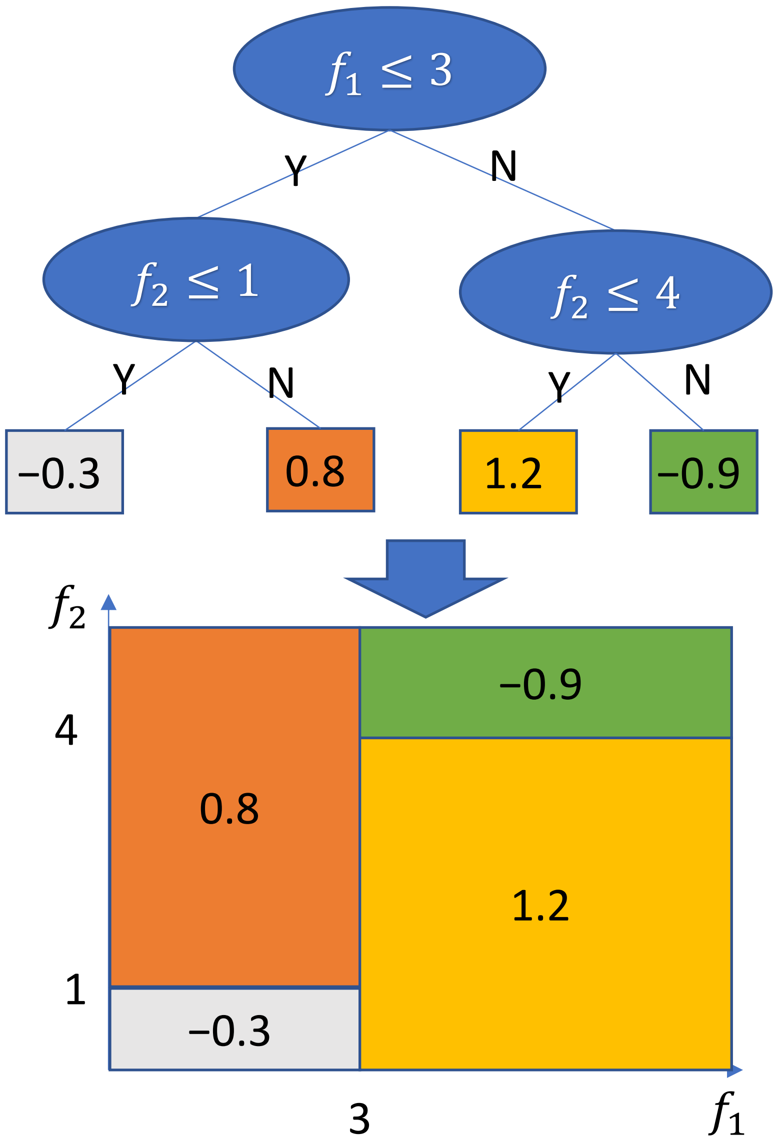
Decision Tree

Recall: Supervised Learning

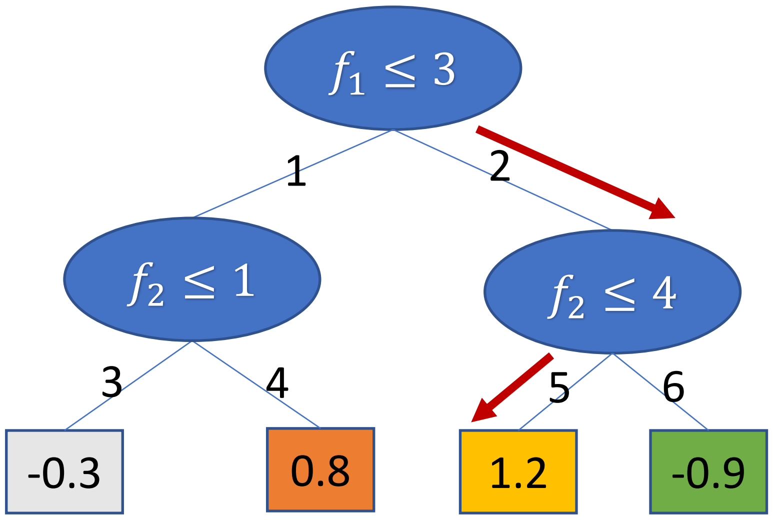
- Components of supervised learning
  - Data: $[X, Y]$
    - $X = [x_1, x_2, \ldots, x_n]^T, Y = [y_1, y_2, \ldots, y_n]^T, x_i = [x_{i1}, x_{i2}, \ldots, x_{im}]$
      - x_i is i-th training record, its label is y_i
      - x_{ij} is the j-th feature value of i-th training record.
  - Model/Function with learnable parameters $\boldsymbol{\theta}$: $F(x;\ \boldsymbol{\theta})$
    - E.g. Linear model $F(x_i\ ; \theta) = \sum_j \theta_j x_{ij}$
  - Objective Loss Function: $\sum_i l(F(x_i; \boldsymbol{\theta}), y_i)$
    - E.g. L2 loss: $l(F(x_i; \boldsymbol{\theta}), y_i) = (F(x_i; \boldsymbol{\theta}) - y_i)^2$
- Goal of supervised learning: learn the parameters $\boldsymbol{\theta}^*$ with (almost) the lowest losses, over data $[X, Y]$
  - $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}}(\sum_i l(F(x_i; \boldsymbol{\theta}), y_i))$

Decision Tree: Structure View

- A decision tree partitions data into many non-overlapping regions
- Assign a constant prediction value to each region
- Components
  - Non-leaf node, (a.k.a. internal node)
    - The highest non-leaf node is called root node
    - Contains a split rule, $\{feature, threshold\}$
    - Partitions current region into two regions
  - Leaf node
    - Each x_i belongs to one leaf
    - Each leaf has an output value



Decision Tree: Inference Example



- x = [1, 0]. decision path 1->3. predicts -0.3.
- x = [1, 2]. decision path 1->4. predicts 0.8.
- x = [4, 3]. decision path 2->5. predicts 1.2.
- x = [4, 5]. decision path 2->6. predicts -0.9.

Decision Tree Definition

- Define a tree with m leaves as $T_m = (\mathcal{S}_{m-1},\ \mathcal{R}_m)$, where
  - $\mathcal{S}_{m-1}$ contains $m-1$ internal nodes $\{S_1, \dots, S_{m-1}\}$
    - The split rule of j-th node S_j is $(f^j,\ t^j)$
  - $\mathcal{R}_m$ contains m leaf node values $\{a_1, \dots, a_m\}$

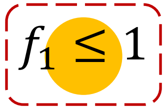
  - $T_m(x_i) = \mathcal{R}_m\big(I(\mathcal{S}_{m-1}, x_i)\big)$, which returns x_i's prediction, where
  - I is a decision function, and returns the x_i's leaf index j based on $\mathcal{S}_{m-1}$
    - The test in j'-th non-leaf node
      - Go to left node if $x_{i,f^{j'}} \le t^{j'}$, otherwise right node (numerical features)
  - $\mathcal{R}_m(j)$ returns a_j, which is the leaf output of leaf j

Decision Tree Learning: Greedy Stage-wise

- Find the optimal structure is hard. Recall: Greedy Stage-wise strategy
- 1. put all samples into root node
    - Root node is also a leaf node
- **2. search for the best split rules, in all leaves, according to split criterion**
- **3. choose the leaves to split, according to growing strategy**
- **4. split the chosen leaves in step 3, and partition the data in the new leaves accordingly**
- 5. repeat 2 to 4, until meet the stop conditions

Decision Tree Learning: Greedy Stage-wise

Index	f1	f2	y
1	0	1	1
2	1	1	1
3	2	3	0
4	2	4	0
5	1	2	1

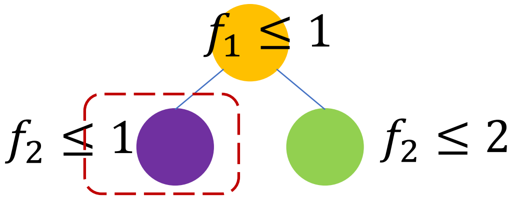


step 2. choose rule according to split criterion

step 3. choose leaves to split (growing strategy)

step 4. split

Index	f1	f2	y
1	0	1	1
2	1	1	1
3	2	3	0
4	2	4	0
5	1	2	1

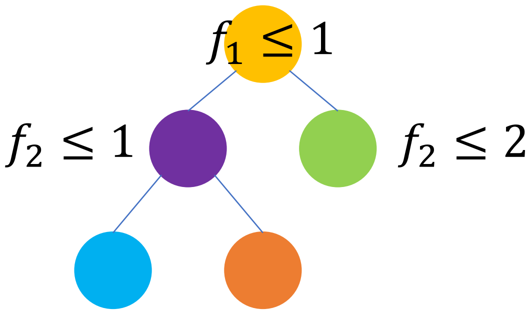


step 2. choose rule according to split criterion

step 3. choose leaves to split (growing strategy)

step 4. split

Index	f1	f2	y
1	0	1	1
2	1	1	1
3	2	3	0
4	2	4	0
5	1	2	1

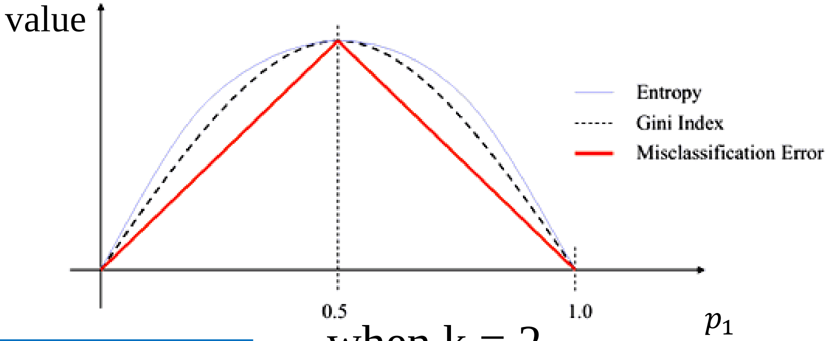


Decision Tree Learning: Split Criterion

Denote loss on leaf j as $L_j = \sum_{x_i \in \text{leaf } j} l(a_j, y_i)$

Split criterion: loss reduction after split $\Delta\text{loss} = L_p - L_{\text{left}} - L_{\text{right}}$

Well-known loss functions L



when k = 2

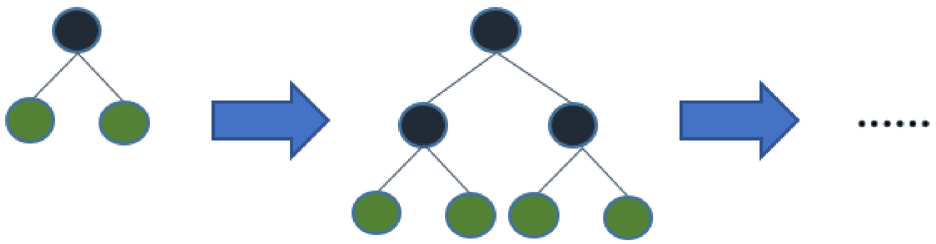
Loss Name	Task	Loss Formula	Optimal Leaf Value
Misclassification Error	K-Class Classification	$L_j = \sum_{x_i \in \text{leaf } j} I[a_j \neq y_i]$	$a_j^* =$ majority class in leaf j
Entropy	K-Class Classification	$L_j = -\sum_{k=1}^{K} p_k \log p_k$, where p_k is proportion of k in j	$a_j^* =$ majority class in leaf j
Gini Index	K-Class Classification	$L_j = 1 - \sum_{k=1}^{K} p_k^2$, where p_k is proportion of k in j	$a_j^* =$ majority class in leaf j
Squared Error	Regression	$L_j = \sum_{x_i \in \text{leaf } j} (y_i - a_j)^2$	$a_j^* = \bar{y}$, label mean in leaf j

Decision Tree Learning: Split Criterion

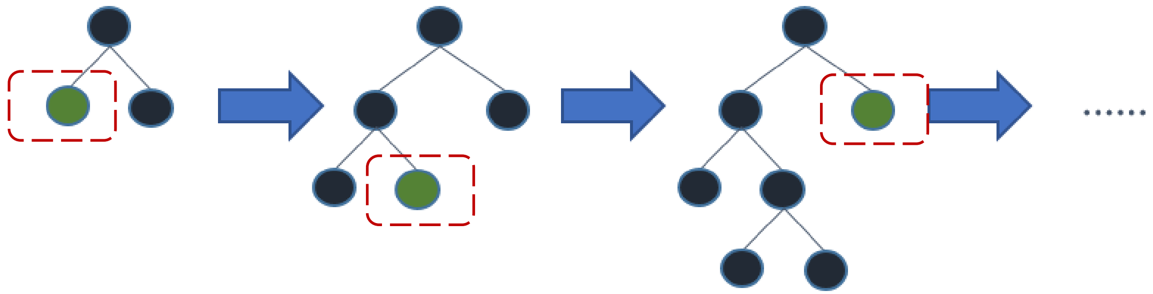
- Maximize the delta loss of the data partition
  - Denote split rule as $x_{i,f} \leq t$
  - $\arg\max_{f,t} \Delta\text{loss} = \arg\max_{f,t} \left(L_p - L_{\text{left}} - L_{\text{right}}\right)$ t is from all unique values of feature f
  - We use Squared Error loss in regression tree in the following
- Firstly, decide the best leaf output values
  - $a_j^* = \arg\min_{a_j} \sum_{x_i \in \text{leaf } j} \left(y_i - a_j\right)^2$
- To achieve the minimal loss, the leaf output is $a_j^* = \dfrac{\sum_{x_i \in \text{leaf } j} y_i}{\sum_{x_i \in \text{leaf } j} 1}$
- Then to find the split rule
  - $\operatorname{argmax}_{f,t} \left(\sum_{x_i \in \text{leaf } p} \left(y_i - a_p^*\right)^2 - \sum_{x_i \in \text{leaf left}} (y_i - a_{\text{left}}^*)^2 - \sum_{x_i \in \text{leaf right}} \left(y_i - a_{\text{right}}^*\right)^2\right)$

Decision Tree Learning: Growing Strategy

- Level-wise
  - Choose all leaves to split
- Leaf-wise
  - Choose the leaf with the maximal loss to split
- Leaf-wise usually is more effective than level-wise



Level-wise tree growth



Leaf-wise tree growth

Decision Tree Learning Algorithm

Algorithm: **DecisionTree (leaf-wise)**
Input: Training data $(\boldsymbol{X}, \boldsymbol{Y})$, number of leaves $\boldsymbol{C}$,
Loss function $\boldsymbol{l}$

⊳ put all data on root

$T_1(X) = X$

For m in (2,C):

⊳ find best split

$\boxed{(p_m, f_m, t_m) = \text{FindBestSplit}(X, Y, T_{m-1}, l)}$

⊳ perform split

$T_m(X) = T_{m-1}(X).\text{split}(p_m, f_m, t_m)$

Algorithm: **FindBestSplit**

Input: Training data $(\boldsymbol{X}, \boldsymbol{Y})$, Loss function $\boldsymbol{l}$, Current Model $\boldsymbol{T}_{\boldsymbol{m}-1}(\boldsymbol{X})$

For all Leaf p in $T_{m-1}(X)$:

$X' = \text{data\_in\_cur\_leaf}(X, p)$

For all f in $X'.\text{features}$:

For all t in $f.\text{thresholds}$:

$(\text{left}, \text{right}) = \text{partition}(p, f, t)$

$\Delta\text{loss} = L_p - L_{\text{left}} - L_{\text{right}}$

if $\Delta\text{loss} > \Delta\text{loss}(p_m, f_m, t_m)$:

$(p_m, f_m, t_m) = (p, f, t)$

Efficient Tree Learning

- The most time-consu
- The time complexity
  - Time cost for partitic
  - The partition could b
  - The $\Delta loss$ could be (

original index	0	1	2	3	4	5	6	7	8
feature values x_1	0.2	0.0	5.0	-3.0	4.0	2.2	1.2	-1.1	0.7

sort

sorted index	3	7	1	0	8	6	5	4	2
feature values x_1	-3.0	-1.1	0.0	0.2	0.7	1.2	2.2	4.0	5.0

left left $t = 0.0$ $t = 0.2$ right right

Algorithm: **FindBestSplit**

Input: Training data (X, Y) , Loss function l, Current Model $T_{m-1}(X)$

For all leaf p in $T_{m-1}(X)$:

$X' = \text{data\_in\_cur\_leaf}(X, p)$

For all f in $X'.\text{features}$:

For all t in $f.\text{thresholds}$:

$(\text{left}, \text{right}) = \text{partition}(p, f, t)$

$\boxed{\Delta\text{loss} = L_p - L_{\text{left}} - L_{\text{right}}}$

$$\sum_{x_i \in \text{leaf } p} (y_i - a_p^*)^2 - \sum_{x_i \in \text{leaf left}} (y_i - a_{\text{left}}^*)^2 - \sum_{x_i \in \text{leaf right}} (y_i - a_{\text{right}}^*)^2$$

if $\Delta\text{loss} > \Delta\text{loss}(p_m, f_m, t_m)$:

$(p_m, f_m, t_m) = (p, f, t)$

Efficient Tree Learning: Δloss Simplification

- Denote L2 loss for a leaf j as L_j
  - Denote $S_j = \sum_{x_i \in \text{leaf } j} y_i$, $SQ_j = \sum_{x_i \in \text{leaf } j} y_i^2$, and n_j the number of data in leaf j
  - Then $L_j = \sum_{x_i \in \text{leaf } j} \left(y_i - \frac{S_j}{n_j} \right)^2$
  - $L_j = \sum_{x_i \in r_j} y_i^2 - 2 \frac{S_j}{n_j} \sum_{x_i \in \text{leaf } j} y_i + n_j \left(\frac{S_j}{n_j} \right)^2 = -\frac{S_j^2}{n_j} + SQ_j$
- And we choose a split with maximal delta loss:
  - $\Delta\text{loss} = L_p - L_{\text{left}} - L_{\text{right}} = \frac{S_{\text{left}}^2}{n_{\text{left}}} + \frac{S_{\text{right}}^2}{n_{\text{right}}} - \frac{S_p^2}{n_P}$ $\quad (SQ_p = SQ_{\text{left}} + SQ_{\text{right}})$
- After simplification, Δloss could be accumulated

Efficient Tree Learning: Sorted Split Finding

Algorithm: **FindBestSplit**

Input: Training data $(\boldsymbol{X}, \boldsymbol{Y})$ **, Current Model** $\boldsymbol{T_{c-1}}(\boldsymbol{X})$

For all Leaf p in $T_{c-1}(X)$:

$X' = \text{data\_in\_cur\_leaf}(X, p)$

For all f in X'.features:

sorted_index = get_sorted_indices(f.values)

$S_{\text{left}} = n_{\text{left}} = 0, \quad S_{\text{right}} = S_p, \ n_{\text{right}} = n_p$

For i **in** (0, len(f.values) - 1):

$j = \text{sorted\_index}[i]$

$S_{\text{left}} \mathrel{+}= y_j \ ; \ n_{\text{left}} \mathrel{+}= 1$

$S_{\text{right}} \mathrel{-}= y_j; \ n_{\text{right}} \mathrel{-}= 1$

$\Delta\text{loss} = \frac{S_{\text{left}}^2}{n_{\text{left}}} + \frac{S_{\text{right}}^2}{n_{\text{right}}} - \frac{S_p^2}{n_p}$

if $\Delta\text{loss} > \Delta\text{loss}(p_m, f_m, v_m)$:

$(p_m, \ f_m, \ v_m) = (p, \ f, \ f.\text{values}[j])$

Could be cached, to avoid re-sort

From $O(\#\text{feature} \times \#\text{threshold} \times \#\text{data})$ to $O(\#\text{feature} \times \#\text{threshold})$

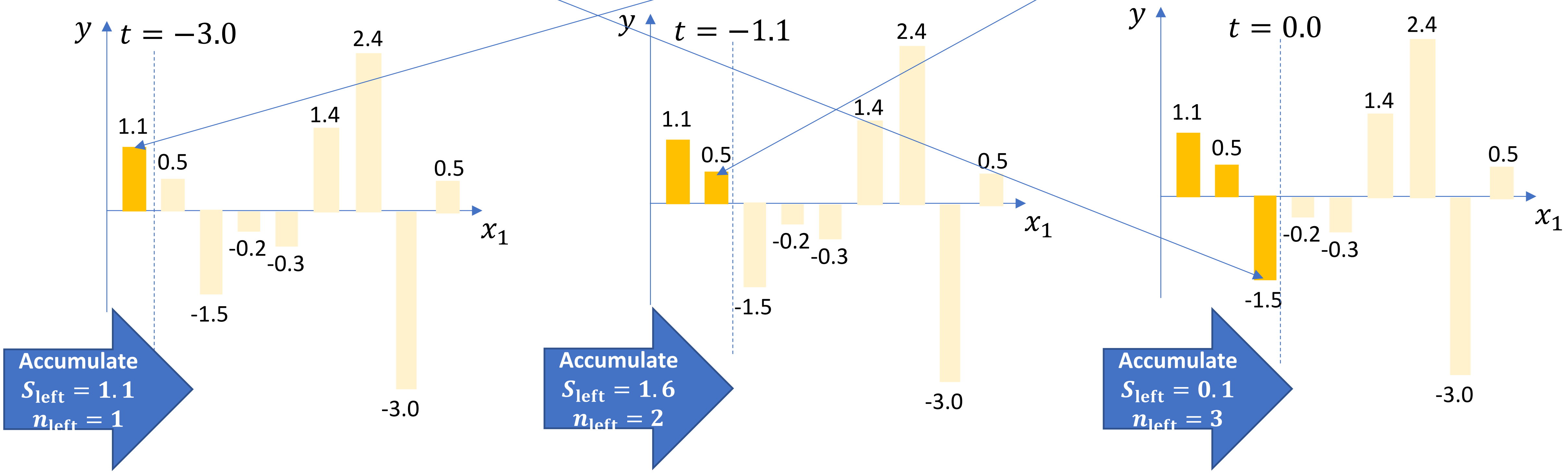
Efficient Tree Learning: Sorted Split Finding

sorted index	3	7	1	0	8	6	5	4	2

original index	0	1	2	3	4	5	6	7	8
feature values x_{ij}	0.2	0.0	5.0	-3.0	4.0	2.2	1.2	-1.1	0.7
label y	-0.2	-1.5	0.5	1.1	-3.0	2.4	1.4	0.5	-0.3

Too much thresholds

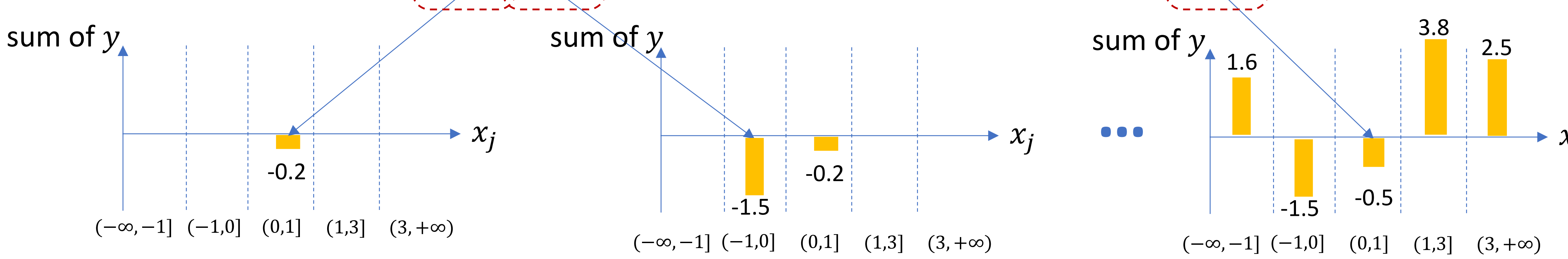
Random access to feature values and label in memory, which can be slow!

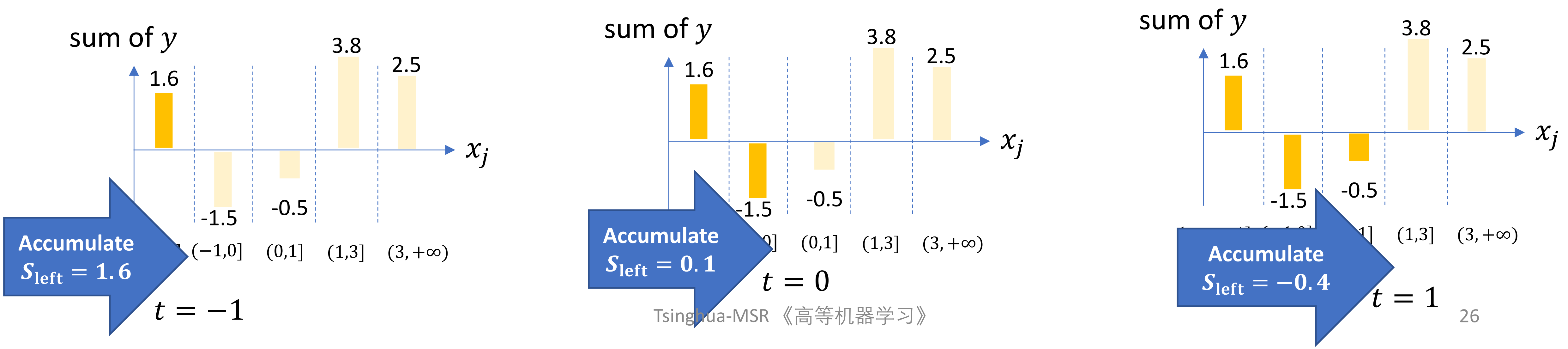


Efficient Tree Learning: Histogram Optimization

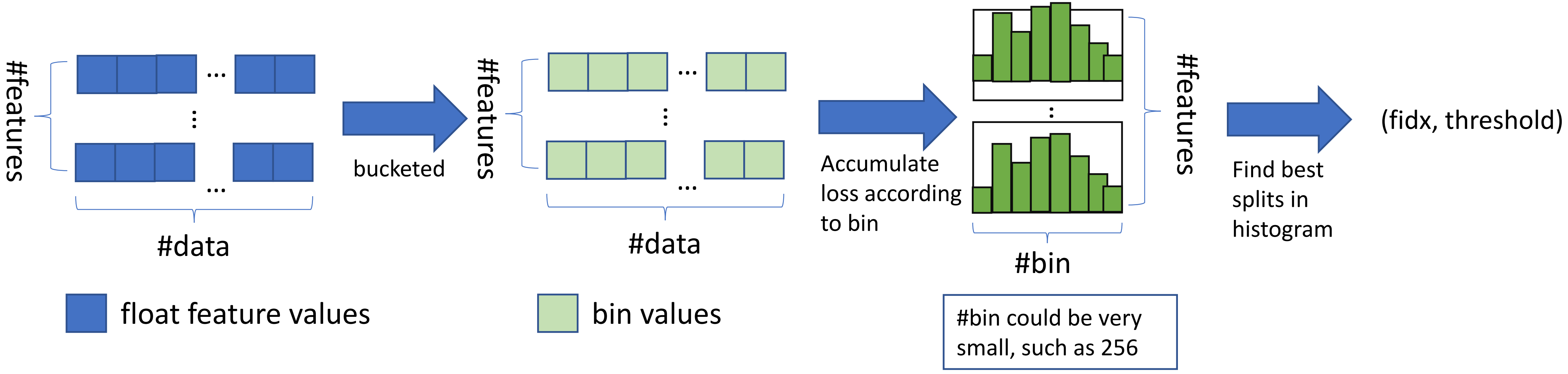
original index	0	1	2	3	4	5	6	7	8
feature values x_{ij}	0.2	0.0	5.0	-3.0	4.0	2.2	1.2	-1.1	0.7
labels y	-0.2	-1.5	0.5	1.1	-3.0	2.4	1.4	0.5	-0.3

$$\Delta\text{loss} = \frac{S_{\text{left}}^2}{n_{\text{left}}} + \frac{S_{\text{right}}^2}{n_{\text{right}}} - \frac{S_p^2}{n_P}$$





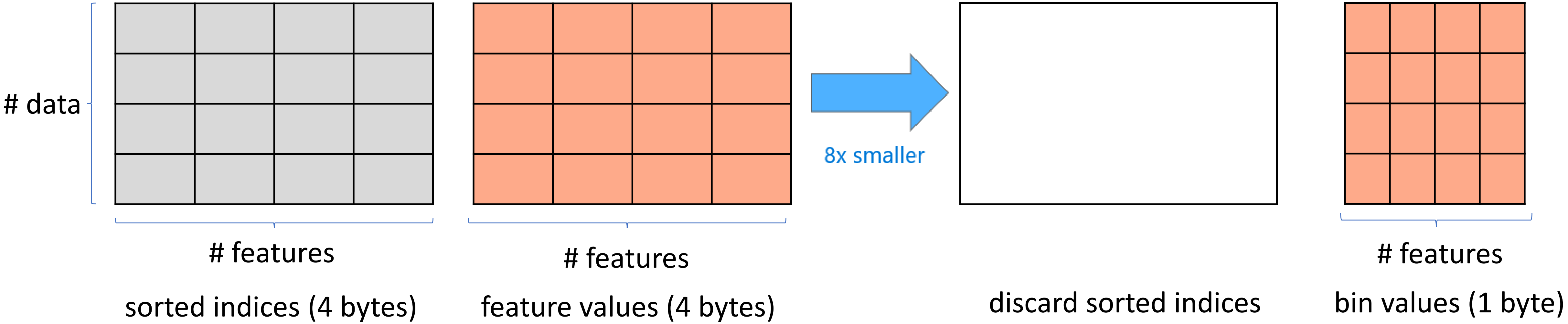
Efficient Tree Learning: Histogram Optimization



- Avoid sorting and remove the need for sorted index
  - bucket the feature values, and accumulate S_{left} and n_{left} in the same bin
- Improve generalization ability
  - Avoid overfitting from too fine-grained threshold
- Bucket continuous values to discrete values(“bin”), discards continuous values
  - E.g. [0,0.1) -> 0, [0.1,0.3)->1, …

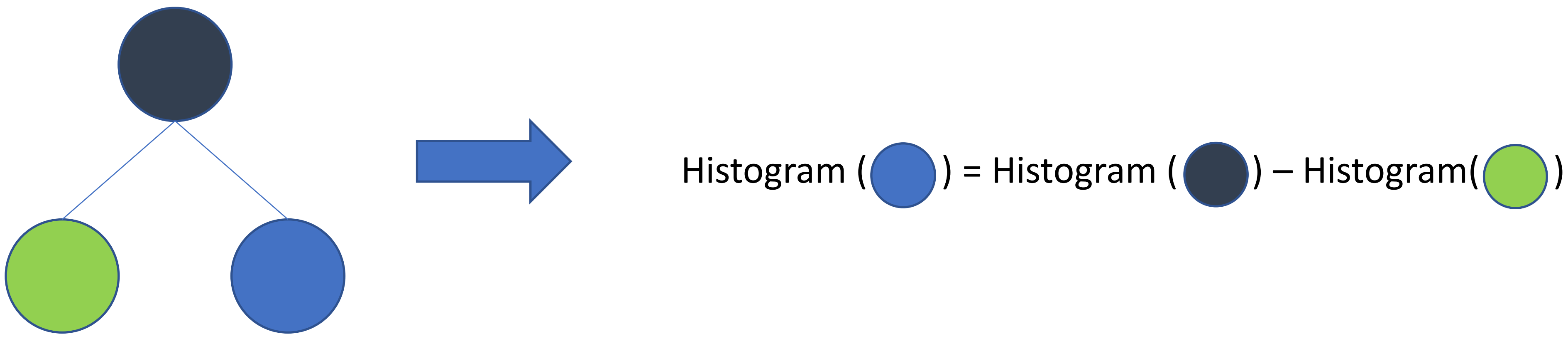
Efficient Tree Learning: Histogram Optimization

- Histogram optimization also reduces the memory cost
- Only need to save bin values.
- If #bins is small, can use small data type, e.g. uint8_t, to store training data



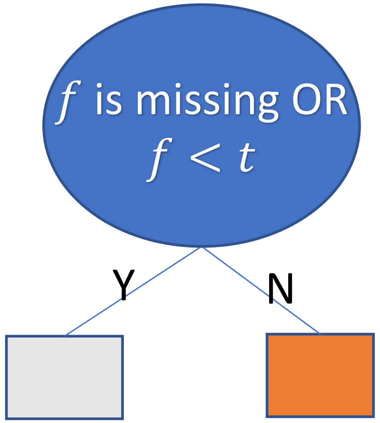
Efficient Tree Learning: Histogram Subtraction

- To get one leaf's histograms in a binary tree, we can use the histogram subtraction of its parent and its neighbor
  - Reduce the cost from #row to #bin
- More than 2x speed-up

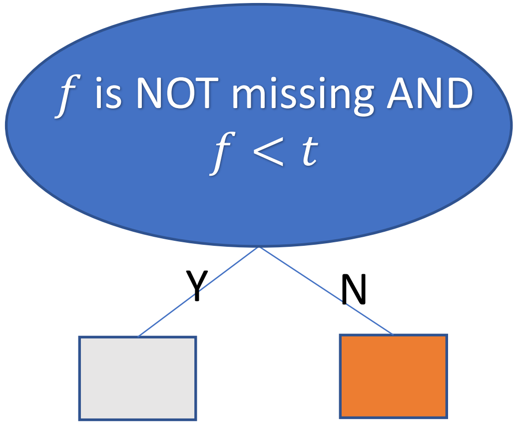


Missing Value Handle in Decision Trees

- In most models, the missing values need to be filled before training
- However, in trees, the missing values could be directly handled
- Simply test which child (left or right) is the best for the missing values
  - For each feature f and each threshold t, test which of the two is better



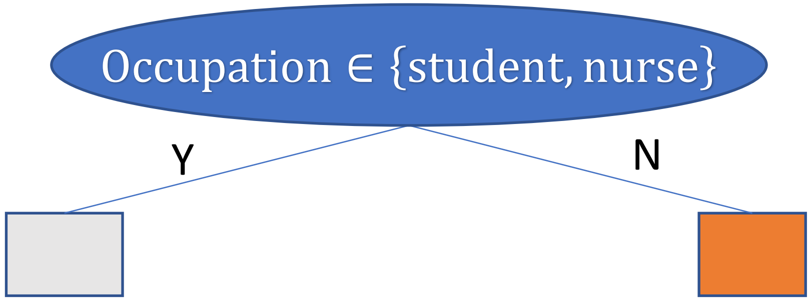
Data with missing f go to left child



Data with missing f go to right child

Categorical Feature in Tree

- Learning tree from numerical values is easier, since they can be ordered
  - Age, temperature, length, …
  - The split rule is, left child if $\mathrm{value} \leq \mathrm{threshold}$, else right child
- However, there's no ordering relation in categorical (nominal) values
  - Occupation $\{\mathrm{student}, \mathrm{nurse}, \mathrm{retired}\}$, gender $\{\mathrm{male}, \mathrm{female}\}$, …
  - The split rule is, left child if value in subset $\{c_1, c_2, \dots, c_k\}$, else right child



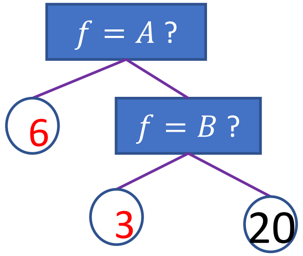
  - There are about 2^#distinct_value possible split results

Categorical Feature in Tree: Encoding

- Unsupervised Encoding: Without Label Information
  - One-hot encoding

$f \in \{A, B, C, D\}$

$A \rightarrow$	1	0	0	0
$B \rightarrow$	0	1	0	0
$C \rightarrow$	0	0	1	0
$D \rightarrow$	0	0	0	1



Note: numbers in circles represent to the $\#data$ in that node

Very unbalanced tree!

  - Count encoding: $[A, B, C, A] \rightarrow [2, 1, 1, 2]$

Categorical Feature in Tree: Encoding

- Supervised Encoding: Target Encoding
  - $A \rightarrow$ estimation of $E[y|f = A]$ (average of y's of all data with $f = A$)

feature value f	A	B	A	C	A	C	B	D
label y	0	1	0	1	1	1	0	1

$A \rightarrow \frac{0+0+1}{3} = 0.33$ $\quad B \rightarrow \frac{1+0}{2} = 0.5$ $\quad C \rightarrow \frac{1+1}{2} = 1.0$ $\quad D \rightarrow \frac{1}{1} = 1.0$

- k-fold Target Encoding: Avoid Overfitting

feature value f	A	B	A	C	A	C	B	D
label y	0	1	0	1	1	1	0	1
	fold 1		fold 2		fold 3		fold 4	

$A_{\text{fold 1}} \rightarrow \frac{0+1}{2} = 0.5$

$A_{\text{fold 2}} \rightarrow \frac{0+1}{2} = 0.5$

$A_{\text{fold 3}} \rightarrow \frac{0+0}{2} = 0$

$A_{\text{fold 4}} \rightarrow \frac{0+0+1}{3} = 0.33$

Ensemble of Decision Trees

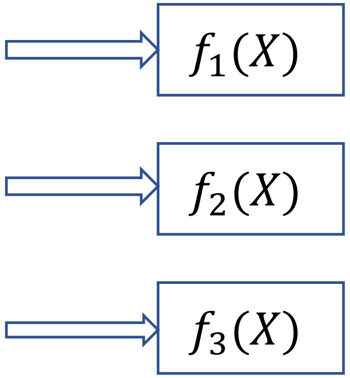
- Cannot always increase the complexity of a single tree
  - Too few data in the deep nodes, causing the unrepresentative splits
- Too deep tree -> overfitting; too shallow tree -> underfitting
- Therefore, a single tree often cannot perform well. And ensemble of shallow trees is widely-used, such as Random Forest and GBDT

Boosting

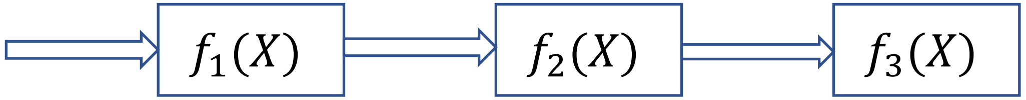
Boosting

- Boosting is an ensemble model
  - $F_m(X) = F_{m-1}(X) + f_m(X) = f_1(X) + \ \ldots f_{m-1}(X) + f_m(X)$
  - $f_i(\cdot)$ is the weak learner
- Ensemble method, like bagging, but with different learning strategies
  - Bagging: learn in parallel, independently (e.g., Random Forest)
  - Boosting: learn sequentially

Bagging:



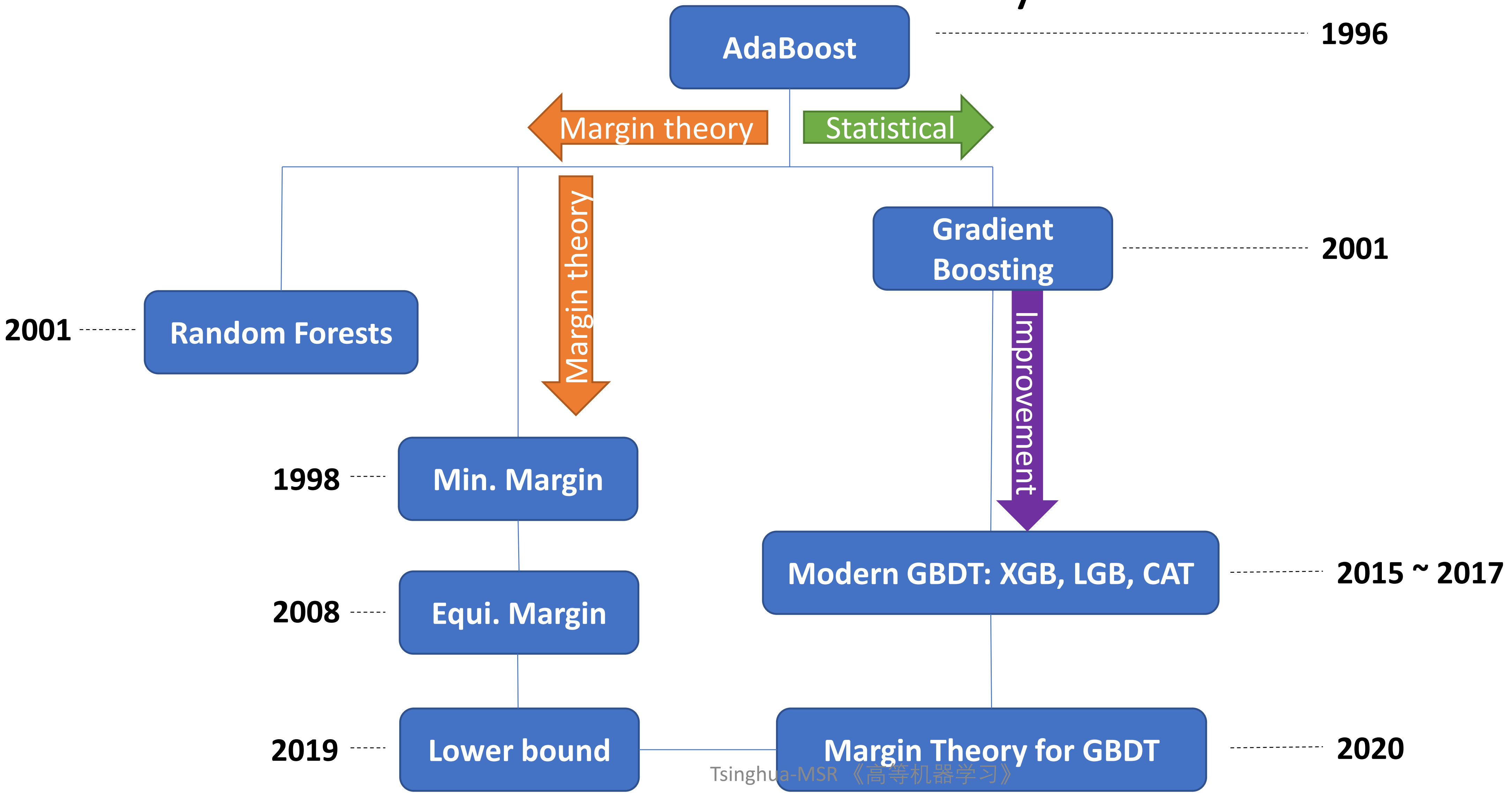
Boosting:



Boosting

- Greedy stage-wise approximation
  - Learn F_1, then $F_2, F_3, \ldots$
  - Add f_m to F_{m-1} so that $F_m(X) = F_{m-1}(X) + f_m(X)$
  - Loss can reduce: $L(F_m(X), Y) < L(F_{m-1}(X), Y)$
- AdaBoost
  - Reduce L by changing the distribution of samples when training f_m
- Gradient Boosting
  - Reduce L by changing training labels when training f_m

Ensemble Methods: A Family Tree



AdaBoost: Make a Weak Learner Strong

Consider binary classification

Strong learnable: a **function class** C over **data space** $X \subseteq R^n$, **exists an algorithm** A,

for any $\epsilon > 0, \delta > 0$, **for any target function** $c \in C$, **for all distributions** D on X,

if $m \geq poly\left(\frac{1}{\epsilon}, \frac{1}{\delta}, n, size(c)\right)$, we have $P_{S\sim D^m}[R(h_S) \leq \epsilon] \geq 1 - \delta$

h_S denotes model learned from dataset S. $R(h_S)$ is the training error.

A is almost always correct, given enough training data. Then A is a **strong learner**.

Weak learnable: a **function class** C over **data space** $X \subseteq R^n$, **exists an algorithm** A,

exists $\gamma > 0$, **for any** $\delta > 0$, **for any target function** $c \in C$, **for all distributions** D on X,

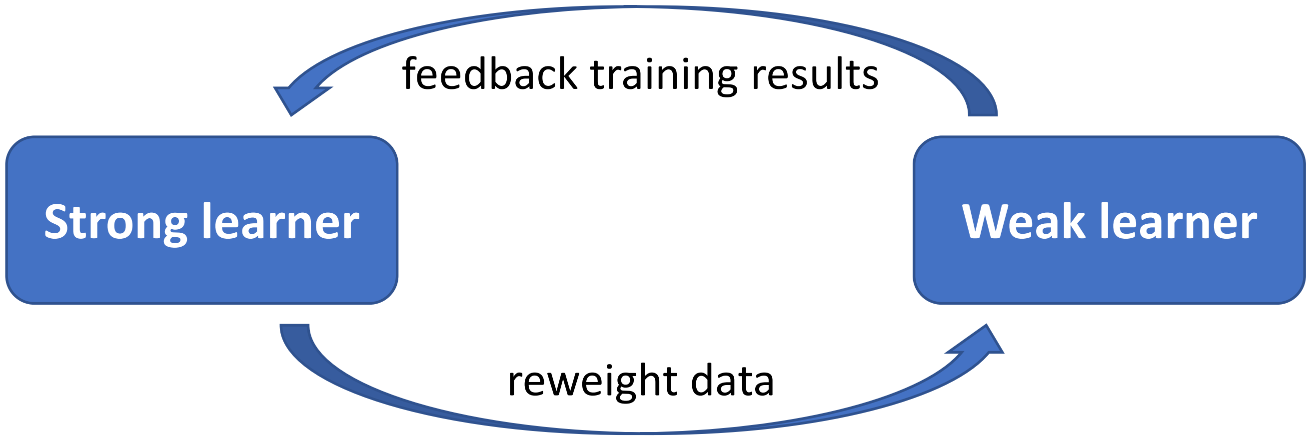
if $m \geq poly\left(\frac{1}{\epsilon}, \frac{1}{\delta}, n, size(c)\right)$, we have $P_{S\sim D^m}\left[R(h_S) \leq \frac{1}{2} - \gamma\right] \geq 1 - \delta$

A is slightly better then random guess, called a **weak learner**.

Question: Does weak learnability equal strong learnability?

AdaBoost - Algorithm

Answer: Yes. We can create a strong learner by calling weak learner as a subroutine.



$\textsc{AdaBoost}(S = ((x_1, y_1), \ldots, (x_m, y_m)))$ $y_i, f(\cdot) \in \{-1,1\}$ (Yoav Freund and Robert Schapire in 1996)

For i **from** 1 **to** m **do**

$\quad w_i^1 = \frac{1}{m}$

For t **from** 1 **to** T **do**

$\quad f_t$ is the base classifier with small weighted error $\epsilon_t = \sum_{i=1}^{m} w_i\,[f_t(x_i) \neq y_i]$ Calls weak learner

$\quad \alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$

$\quad Z_t = 2[\epsilon_t(1-\epsilon_t)]^{\frac{1}{2}}$

$\quad$ **For** i **from** 1 **to** m **do**

$\quad\quad w_i^{t+1} = w_i^t \exp(-\alpha_t y_i f_t(x_i)) / Z_t$ Reweight: increase weights of wrongly classified data in previous iterations

$F_T = \sum_{t=1}^{T} \alpha_t f_t$

Return F_T

AdaBoost – Theoretical Facts

For training, the empirical error of the classifier returned by AdaBoost satisfies:

$$\hat{R}_S(F_T) \leq \exp\left[-2\sum_{t=1}^{T}\left(\frac{1}{2}-\epsilon_t\right)^2\right]$$

Furthermore, if for all $t \in [T], \gamma \leq \left(\frac{1}{2}-\epsilon_t\right)$, then $\hat{R}_S(F_T) \leq \exp(-2\gamma^2 T)$.

For generalization, with any data distribution D and probability $> 1-\delta$

$$R_D(F) \leq \hat{R}_S(F) + O\left(\frac{1}{\sqrt{m}}\left(\log 2m + d' + \log\left(\frac{9}{\delta}\right)\right)^{\frac{1}{2}}\right)$$

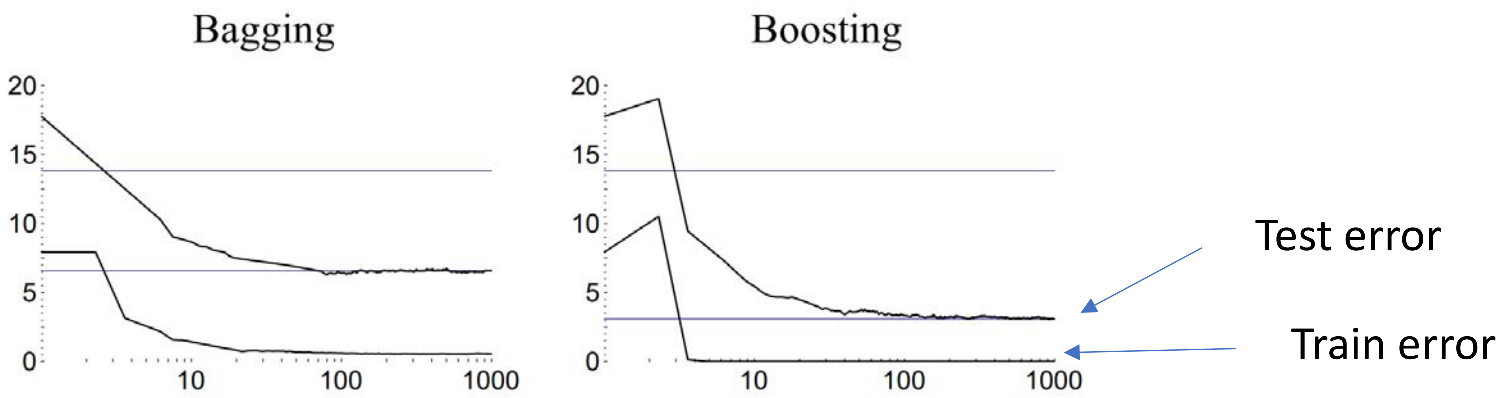
where d' is the VC-dimension of the class of ensemble of T base functions in set H, and

$$d' \in O(2(d+1)(T+1)\log(T+1))$$

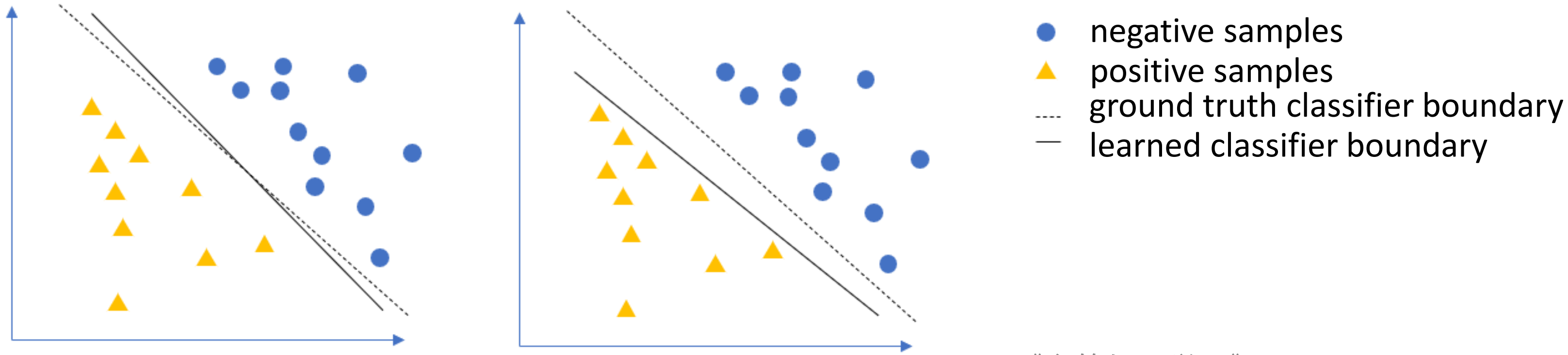
where d is the VC-dimension of H.

AdaBoost Explanation: Margin Theories

AdaBoost can continue to decrease test error, after the training error **is already 0**



Recall: SVM maximizes the minimum margin $\theta^* = \min_i y_i F(x_i)$ of the linear classifier F



AdaBoost Explanation: Margin Theories

A tighter generalization bound with margin

$\mathcal{H}$ is a set of base models, for any linear combination F of base models in $\mathcal{H}$ we have generalization bound

$$P_D[yF(x) \le 0] \le P_S[yF(x) \le \theta] + O\left(\frac{1}{\sqrt{m}}\left(\frac{\log m \log|\mathcal{H}|}{\theta^2} + \log\left(\frac{1}{\delta}\right)\right)^{\frac{1}{2}}\right)$$

Let $\theta^* = \min_i y_i F(x_i)$

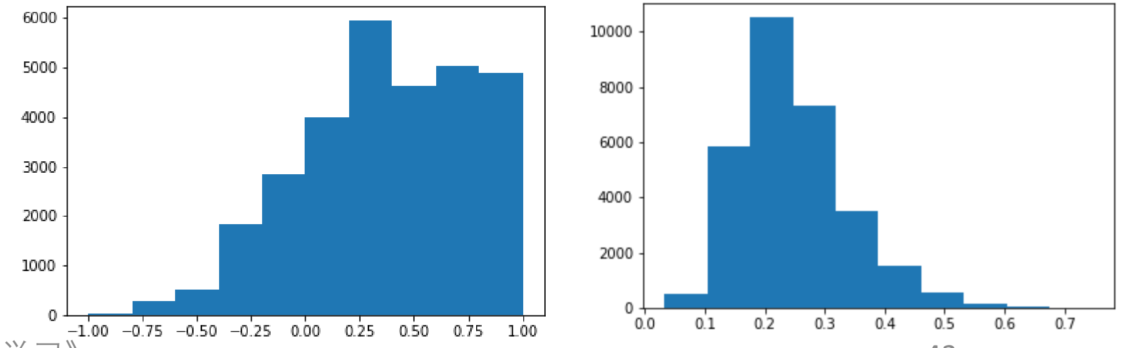
$$P_D[yF(x) \le 0] \le O\left(\frac{1}{\sqrt{m}}\left(\frac{\log m \log|\mathcal{H}|}{\theta^{*2}} + \log\left(\frac{1}{\delta}\right)\right)^{\frac{1}{2}}\right) \approx O\left(\sqrt{\frac{\log m}{m}}\right)$$

AdaBoost can increase the smaller margins:

After T iterations, we have

$$P_{(x,y)\sim S}[yF_T(x) \le \theta] \le \left(\sqrt{(1-2\gamma)^{1-\theta}(1+2\gamma)^{1+\theta}}\right)^T$$

When $\theta < \gamma$, $\quad (1-2\gamma)^{1-\theta}(1+2\gamma)^{1+\theta} < 1$



after 5 iterations　　after 50 iterations

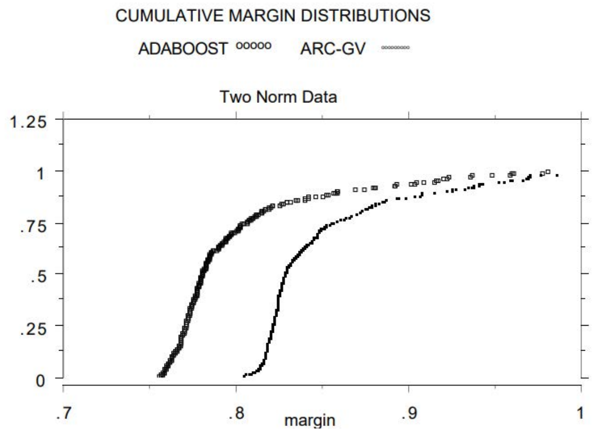
AdaBoost Explanation: Margin Theories

Question: Can we design a new boosting algorithm to **explicitly** optimize the margins?

arc-gv proposed by Breiman in 1998:

- a new ensemble algorithm maximize minimum margin $\theta^* = \min_{\mathrm{i}} y_i F(x_i)$
- a sharper bound with minimum margin: $O\left(\frac{\log m}{m}\right)$ (shaper than $O\left(\sqrt{\frac{\log m}{m}}\right)$)

However, the experiments failed



	Test Set Error	
data set	arc-gv	Adaboost
twonorm		
k=8	5.3	4.9
k=16	6.0	4.9
threenorm		
k=8	18.6	17.9
k=16	18.5	17.8
ringnorm		
k=8	6.1	5.4
k=16	8.3	6.3
breast cancer		
k=16	3.3	2.9
k=32	3.4	2.7
ionosphere		
k=8	3.7	5.1
k=16	3.1	3.1
sonar		
k=8	11.9	8.1
k=16	16.7	14.3

The previous margin-based generalization bound fails?

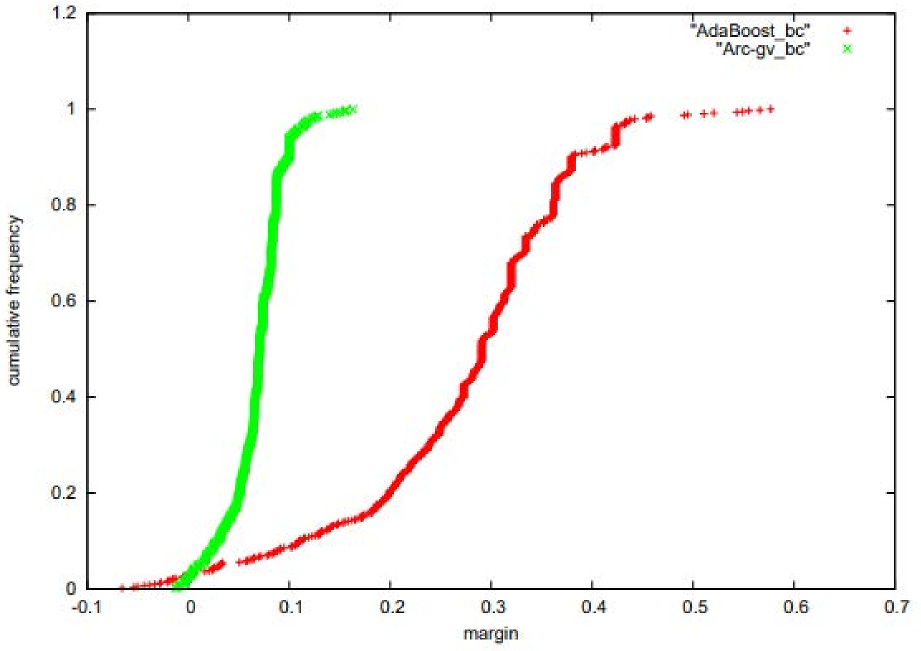
Minimum Margin vs. Equilibrium Margin

Schapire etal. 1998

Brieman's experiment does not strictly control tree size

Exactly same tree size for both algorithms

Arc-gv does produce smaller min. margin



Wang etal. 2008

The distribution of margins matters!

Not only the minimum margin.

A tighter bound with equilibrium margin in 2008.

Theorem 3 *If $|H| < \infty$, then for any $\delta > 0$, with probability at least $1-\delta$ over the random choice of the training set S of n examples, every voting classifier f satisfies the following bound:*

$$P_D\Big(yf(x) \le 0\Big) \le \frac{\log|H|}{n} + \inf_{q\in\{0,\frac{1}{n},\frac{2}{n},\ldots,1\}} D^{-1}\Big(q, u\boxed{\left[\hat{\theta}(q)\right]}\Big), \quad (3)$$

Quantiles of margins

Random Forests

Train an **ensemble** of **diverse** decision trees, **independently**

Diversity: 1. Randomly sample a fraction of training data for each tree (bootstrap / bagging)
2. Randomly select a fraction of features when splitting each node

Explanation: Ensemble of diverse base models can decrease the variance of the ensemble

Given B $i.i.d.$ random variables $X_1, \ldots, X_B$, with **correlation** $E\left[(X_i - \mu)\left(X_j - \mu\right)\right] = \rho$, **variance** $E[(X_i - \mu)^2] = \sigma^2$, then the **variance of their average is**

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

ρ is reduced by **diversity**
B is increased by **ensemble**

Recall: BV Decomposition
$Y = f(x) + \epsilon$
$Var(\epsilon) = \sigma_\epsilon^2$

$$\begin{aligned}
\mathrm{Err}(x_0) &= E[(Y - \hat{f}(x_0))^2 | X = x_0] \\
&= \sigma_\varepsilon^2 + [\mathrm{E}\hat{f}(x_0) - f(x_0)]^2 + E[\hat{f}(x_0) - \mathrm{E}\hat{f}(x_0)]^2 \\
&= \sigma_\varepsilon^2 + \mathrm{Bias}^2(\hat{f}(x_0)) + \mathrm{Var}(\hat{f}(x_0)) \\
&= \text{Irreducible Error} + \mathrm{Bias}^2 + \text{Variance}.
\end{aligned}$$

Gradient Boosting

- We want to get f_m that satisfies
  - $L(F_{m-1}(X) + f_m(X), Y) < L(F_{m-1}(X), Y)$
- Calculate the negative gradients
  - $\hat{y}_i = -\partial_{F_{m-1}(x_i)} l(F_{m-1}(x_i), y_i)$

When l is squared loss, $\hat{y}_i = y_i - F_{m-1}(x_i)$

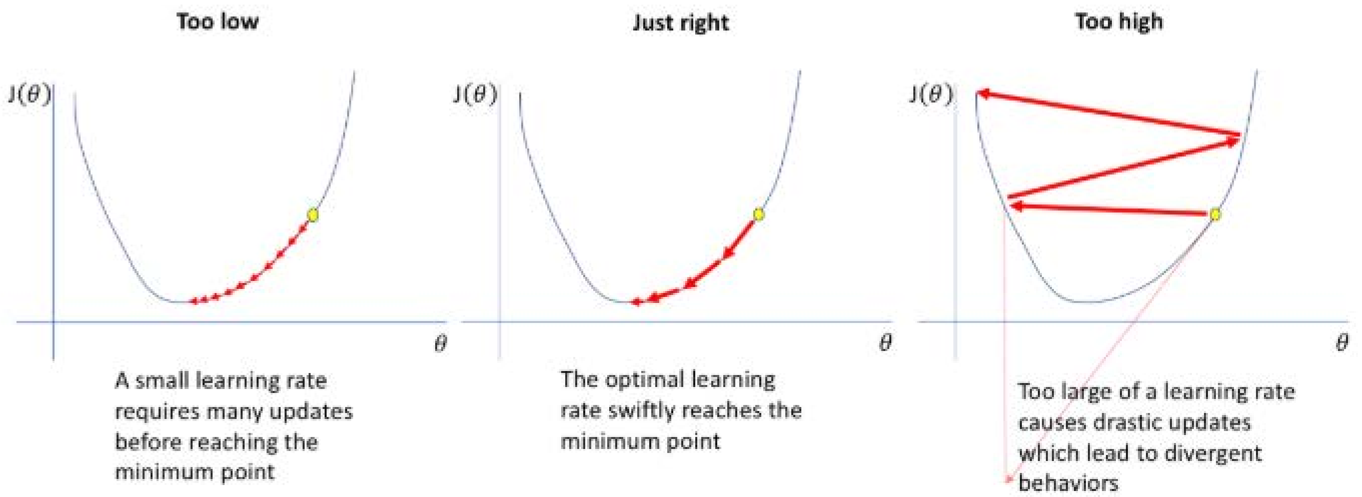
- Learn f_m to fit $\hat{Y}$ by minimizing squared loss
  - $f_m = \arg\min_f \sum_{i=1}^n (f(x_i) - \hat{y}_i)^2$

When l is squared loss, $f_m = \arg\min_f \sum_{i=1}^n \left(f(x_i) - (y_i - F_{m-1}(x_i))\right)^2$

- First-order Taylor Expansion
  - $l(y_i, F_{m-1}(x_i) + f_m(x_i)) = l(y_i, F_{m-1}(x_i)) + \partial_{F_{m-1}(x_i)} l(F_{m-1}(x_i), y_i) f_m(x_i)$
  - And $f_m(x_i) \approx \hat{y}_i = -\partial_{F_{m-1}(x_i)} l(F_{m-1}(x_i), y_i)$
  - Then $l(y_i, F_{m-1}(x_i) + f_m(x_i)) \approx l(y_i, F_{m-1}(x_i)) - \hat{y}_i^2 < l(y_i, F_{m-1}(x_i))$

Shrinkage

- Shrinkage of $f_m(X)$ on each iteration
  - $F_m(X) = F_{m-1}(X) + \gamma f_m(X),$ where γ is shrinkage rate
- Avoid too large optimization steps
  - Like the learning rate in gradient descent



Stochastic Boosting

- Use a random subset in each iteration
  - Sub-rows: could be used when #data is relatively large
  - Sub-features: could be used most of time
- Leverage the Bagging into Boosting framework
- Speed up the learning, as only use subset in training
- Better generalization ability, benefit from bagging

Equivalence of AdaBoost and Gradient Boosting

- With loss function $L(F,Y) = \sum_{i=1}^{n} e^{-y_i F(x_i)}$, constraining $f(\cdot) \in \{-1,1\}, y_i \in \{-1,1\}$
- AdaBoost can be derived from gradient boosting.

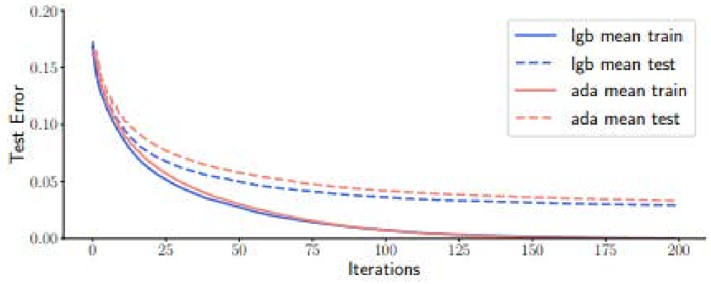
$$
\begin{aligned}
L(F_{t+1}, Y) &= \sum_{i=1}^{n} e^{-y_i \sum_{j=1}^{t+1} \alpha_j f_j(x_i)} = \sum_{i=1}^{n} w_i e^{-y_i \alpha_{t+1} f_{t+1}(x_i)} \\
&= e^{-\alpha_{t+1}} \sum_{i: y_i = f_{t+1}(x_i)} w_i + e^{\alpha_{t+1}} \sum_{i: y_i \neq f_{t+1}(x_i)} w_i \\
&= e^{-\alpha_{t+1}} \sum_{i=1}^{n} w_i + \left(e^{\alpha_{t+1}} - e^{-\alpha_{t+1}}\right) \sum_{i: y_i \neq f_{t+1}(x_i)} w_i \\
&= e^{-\alpha_{t+1}} \sum_{i=1}^{n} w_i + \left(e^{\alpha_{t+1}} - e^{-\alpha_{t+1}}\right) \sum_{i=1}^{n} w_i I\left[f_{t+1}(x_i) \neq y_i\right].
\end{aligned}
$$

- Optimal value of α_{t+1}

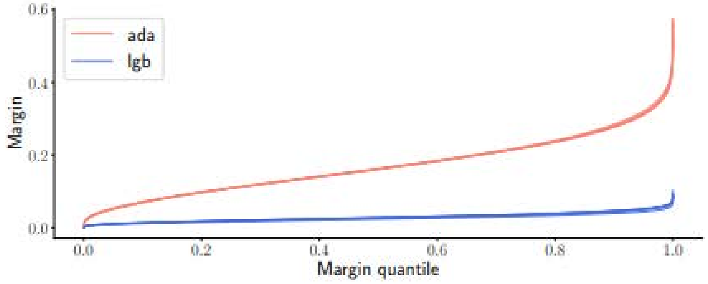
$$
\alpha_{t+1}^{*} = \frac{1}{2} \ln \frac{\sum_{i:\, y_i = f_{t+1}(x_i)} w_i}{\sum_{i:\, y_i \neq f_{t+1}(x_i)} w_i}
$$

Does Margin Theory Works for Gradient Boosting

- Again, gap between margin and generalization error

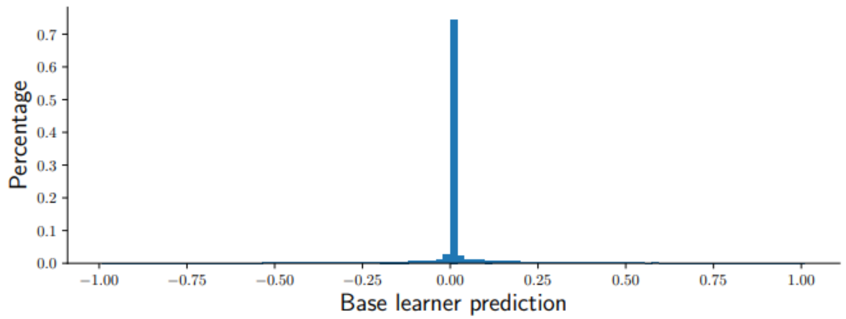


(a) Mean training and test error over five runs. The standard deviation of the final test error is 0.00037 for AdaBoost and smaller for LightGBM.



(b) Sorted margin values.

- Observation: GBDT tends to produce trees with small leaf values
  - But margin theory assumes base learners output {-1, 1}



Does Margin Theory Works for Gradient Boosting

Introduce moment N to describe the variance in base learner contributions

$$\mathcal{L}_{\mathcal{D}}(f) \leq \mathcal{L}_S^{\theta}(f) + O\left(\frac{\boxed{N} \lg|\mathcal{H}| \lg m}{m} + \sqrt{\mathcal{L}_S^{\theta}(f) \cdot \frac{\boxed{N} \lg|\mathcal{H}| \lg m}{m}}\right),$$

$$\text{where } N = \max\{\theta^{-2} \cdot \left(\mathbb{E}_{(x,y)\sim S}\left[\mathbb{E}_{h\sim\mathcal{Q}(f)}\left[\Delta(x,h)^2\right]^{(\lg(16m))/2}\right]\right)^{2/(\lg(16m)}, \theta^{-1}\}. \quad \Delta(x,h) := |f(x) - h(x)|$$

Data Set	Alg.	Train Err	Test Err	Mean Margin	Max Depth	Mean Depth	Moment
Forest	ada	0.0001	0.0331	0.1696	22.0	12.4	0.969
	lgb	0.0002	0.0291	0.0280	23.7	13.9	0.025
Boone	ada	0.00009	0.0589	0.311	17.5	10.2	0.917
	lgb	0.00009	0.0552	0.0818	17.6	10.4	0.0564
Higgs	ada	0.178	0.277	0.0747	24.9	13.5	0.99
	lgb	0.185	0.251	0.018	26	14.7	0.0289
Diabetes	ada	0	0.268	0.148	3.5	2.63	0.973
	lgb	0.0264	0.26	0.142	3.5	2.63	0.214

Compared with Decision Tree

- Both are in Greedy Stage-wise Approximation framework
- Boosting adds new models, tree partitions data and adds leaves
- Boosting can use the full dataset on all stages, while tree can only use the data in that node
- Boosting can train many iterations without overfitting, while tree cannot
- Boosting needs the weak learner, while tree doesn’t need

GBDT

GBDT

- GBDT = Gradient Boosting + Decision Tree
  - Decision tree as weak learner of gradient boosting
  - The combination of two greedy stage-wise approximation models
  - Solve the problems in both boosting and tree:
    - Boosting needs a weak learner
    - Tree cannot always increase its complexity
- For different task/application, the main difference is the loss function

Algorithm: **GBDT**
Input: Training data $(\boldsymbol{X}, \boldsymbol{Y})$, Iteration $\boldsymbol{M}$,
Loss function $\boldsymbol{l}$, number of leaf C
$F_0(X) = 0$
For m in (1,M):
⊳ get the training targets
For all $(x_i, y_i) \in (X, Y)$:
$y_i^m = -\partial_{F_{m-1}(x_i)} l\big(y_i, F_{m-1}(x_i)\big)$
⊳ use decision tree to fit targets
$f_m(X) = DecisionTree(X, Y^m, C, L2Loss)$
$F_m(X) = F_{m-1}(X) + \gamma f_m(X)$

Regression: $l(y_i, F_{m-1}(x_i)) = (y - F_{m-1}(x_i))^2$

Binary Classification: $\bar{y}_i = \frac{1}{1+e^{-F_{m-1}(x_i)}}$

$$l(y_i, \bar{y}_i) = y_i \log \bar{y}_i + (1 - y_i)\log(1 - \bar{y}_i)$$

Lambdarank: Using $y_i^m = \lambda_i = \sum_{i<j} \lambda_{ij} - \sum_{j<i} \lambda_{ji}$

$$\lambda_{ij} = \frac{-\sigma\left|\Delta NDCG_{ij}\right|}{1 + e^{\sigma(F_{m-1}(x_i) - F_{m-1}(x_j))}},$$

GBDT – Second-order Gradients

- Approximate the boosting loss with 2nd order Taylor expansion

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

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

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

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

- Optimal leaf value and loss

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda},$$

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T.$$

- Both gradients and hessians (second-order gradients) are required

Chen T, Guestrin C. Xgboost: A scalable tree boosting system[C]//Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016: 785-794.

GBDT Tools

dmlc XGBoost

- https://github.com/dmlc/xgboost
- Pre-sorted with level wise algorithm
- The first high performance GBDT tool and remaining its popularity

LightGBM

- https://github.com/Microsoft/LightGBM
- Histogram with leaf wise algorithm
- The fastest GBDT tool and becoming more and more popular

Yandex CatBoost

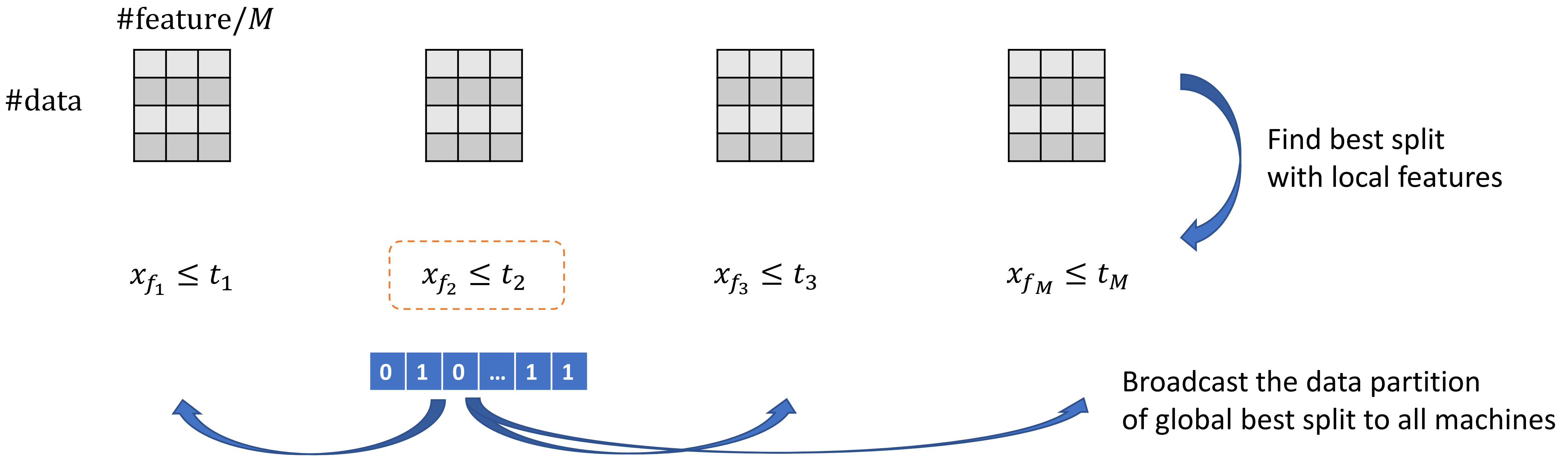
- https://github.com/catboost/catboost
- Categorical feature handling
- Improved boosting framework

LightGBM Highlights

- Highly efficient implementation for GBDT
- Memory saving
- Distributed and GPU training support
- Novel algorithms to further speed up the training
  - Gradient-based One Side Sampling (GOSS) -> reduce the #row in training
  - Exclusive Feature Bundling (EFB) -> reduce the #feature in training
  - Quantized Training
  - Dynamic Categorical Feature Encoding
  - Piece-wise Linear Trees

Distributed Training: Feature Parallel

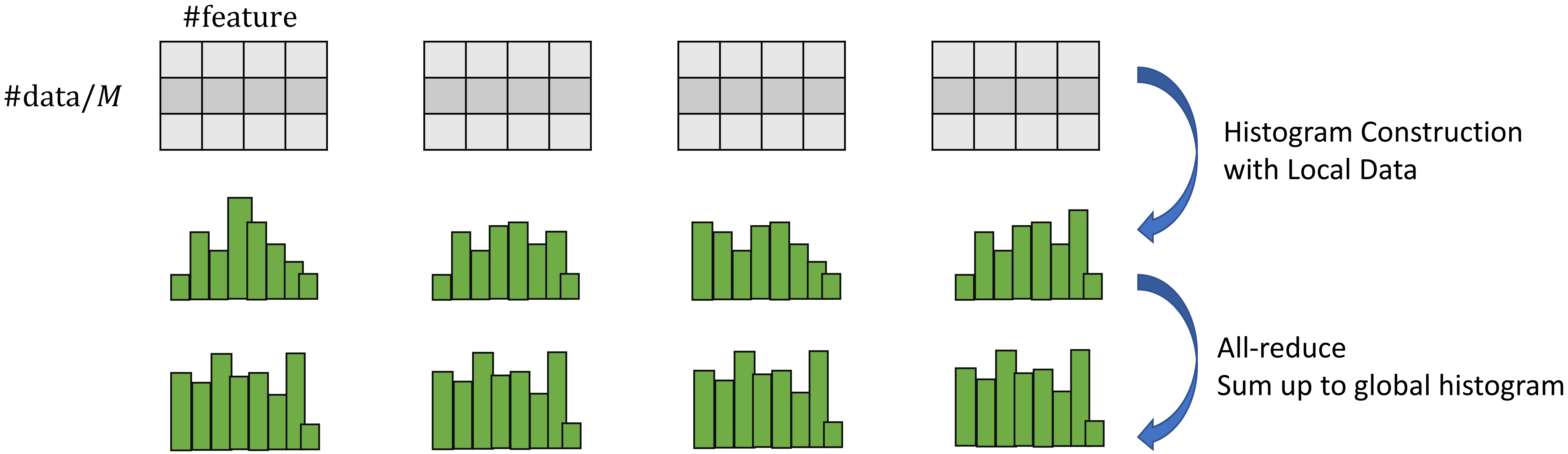
- Partition by columns (features), distribute to M machines



- Communication cost: $O(\#\mathrm{data})$

Distributed Training: Data Parallel

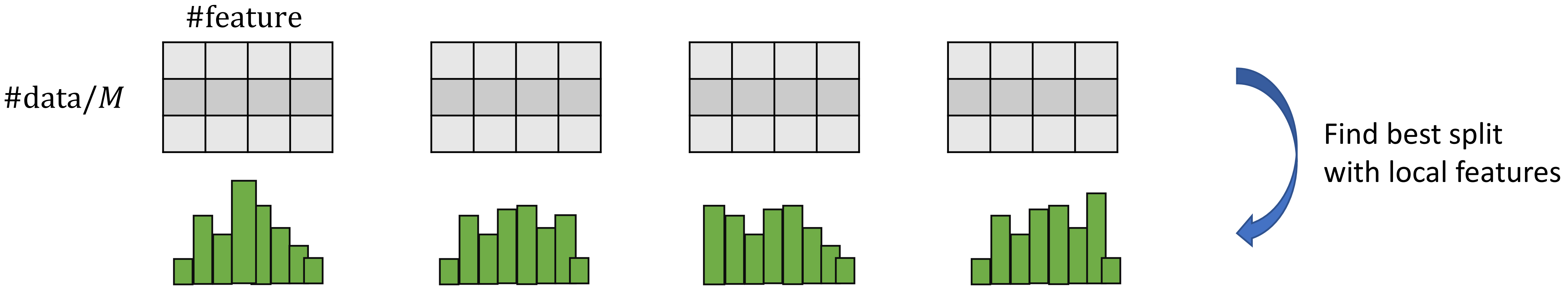
- Partition by rows (data), distribute to M machines



- Communication cost: $O(\#\text{feature} \times \text{histogram size})$

Distributed Training: Voting Parallel

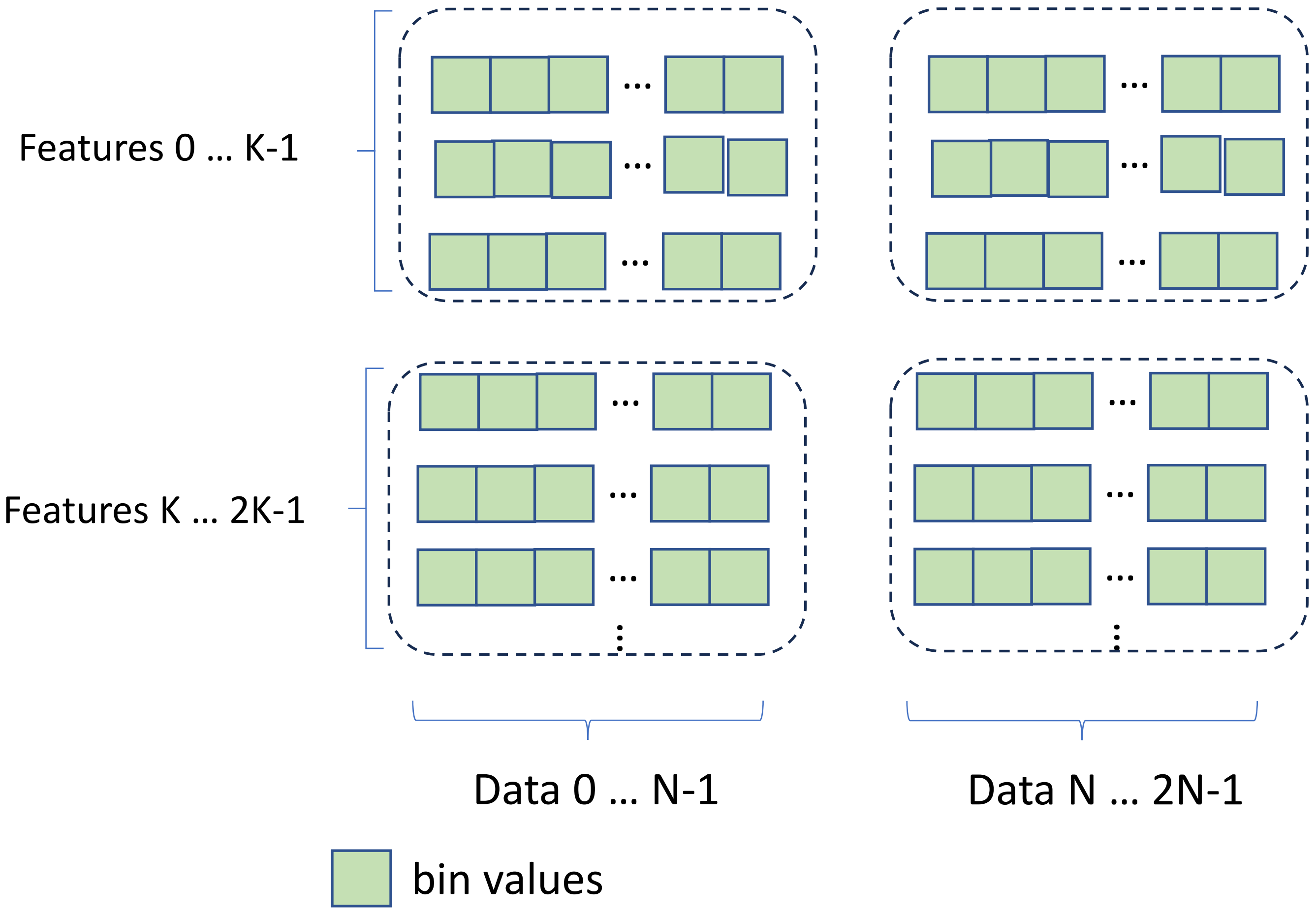
- Similar with data parallel



- Calculate top-K best split features using local histograms (vote)
- Select top-$2K$ features that get the highest number of votes
- All-reduce the histograms only for these $2K$ features
- Communication cost: $O(K \times \text{histogram size})$

GPU Acceleration

- Data + Feature partitioning across GPU streaming multiprocessors



Gradient-based One-Side Sampling (GOSS)

- Speed up the training by using a sample set, without hurting the accuracy
- The sample make the estimation of gradient sum S unbiased
  - Keep the instances with large gradient values
  - Sample the instances with small error and give them a larger weight

Recall: $\frac{S_{\text{left}}^2}{n_{\text{left}}} + \frac{S_{\text{right}}^2}{n_{\text{right}}} - \frac{S_P^2}{n_P}$

Row id	gradients
4	-5
3	3
2	0.5
6	0.2
5	0.1
1	0

select top 2

and randomly sample 2 from the rest

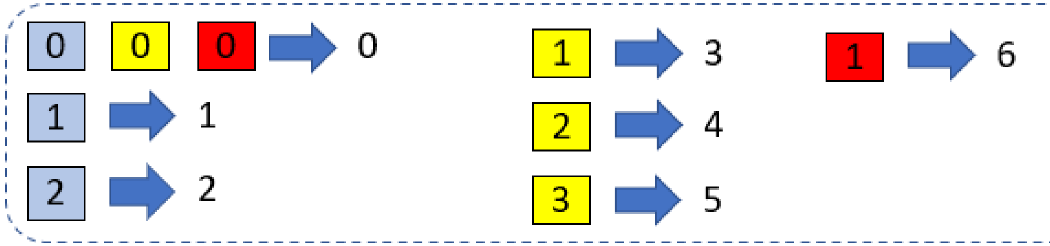
Sampling data

Row id	gradients	weights
4	-5	1
3	3	1
6	0.2	2
5	0.1	2

Exclusive Feature Bundling (EFB)

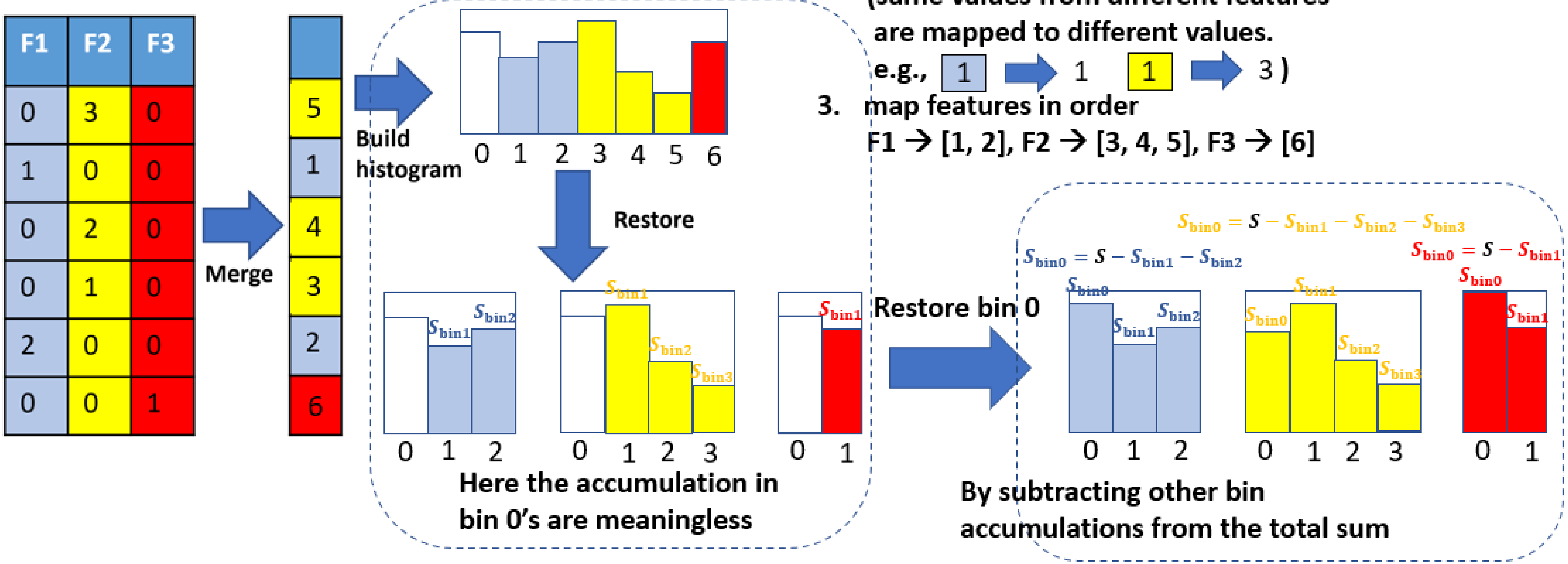
- Speed up the training by reducing #features used in histogram construction
- High-dimensional data are usually very sparse. In such a sparse space, many features are exclusive to each other, i.e., they will not take non-zero values simultaneously
- Thus, the #features can be reduced by bundling these exclusive features

Exclusive Feature Bundling (EFB)



Details in Merge:

1. **map missing values of all features (represented by 0's here) to 0**
2. **remap non-missing values (same values from different features are mapped to different values. e.g., 1 → 1 1 → 3)**
3. **map features in order F1 → [1, 2], F2 → [3, 4, 5], F3 → [6]**



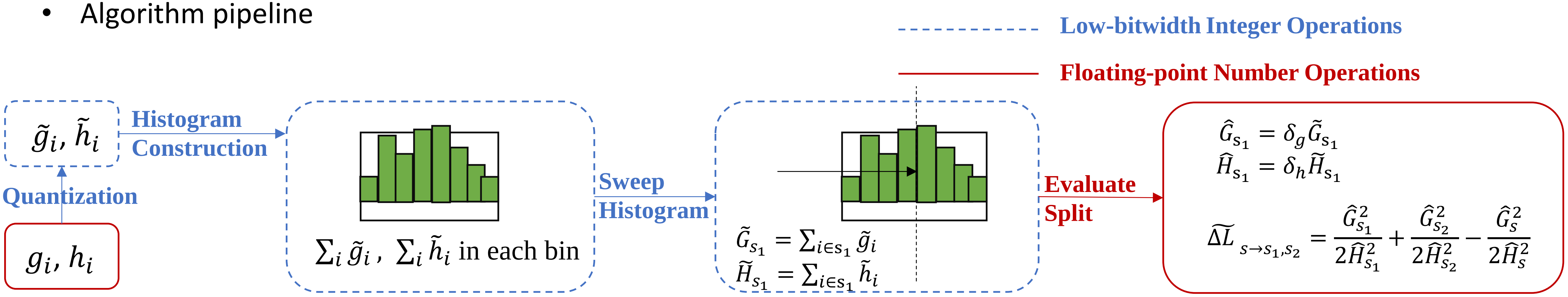
Quantized Training

- Gradient Quantization: Equal-distance division of the gradient value range

$$\alpha = \frac{2 \cdot \max_j |g_j|}{B} \qquad \hat{g}_i \in \left\{-\frac{B}{2}, -\left(\frac{B}{2}-1\right), \dots, -1, 0, 1, \dots, \left(\frac{B}{2}-1\right), \frac{B}{2}\right\}$$

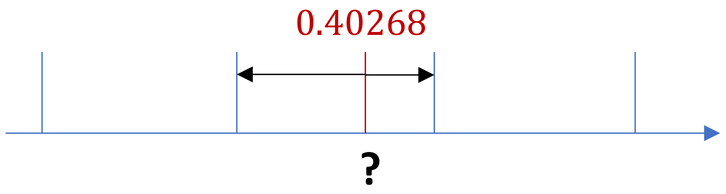
$$\beta = \frac{\max_j h_j}{B} \qquad \hat{h}_i \in \{0, 1, \dots, (B-1), B\}$$

- Algorithm pipeline



Quantized Training

- Quantization: Cast more values into fewer values



32-bit FP number
2-bit Integer

- Round-to-nearest

$$\mathrm{RN}(x) = \begin{cases} \lfloor x \rfloor, & x < \lfloor x \rfloor + \frac{1}{2} \\ \lceil x \rceil, & x \geq \lfloor x \rfloor + \frac{1}{2} \end{cases}$$

- Stochastic rounding

$$\mathrm{SR}(x) = \begin{cases} \lfloor x \rfloor, & \text{w.p.} \quad \lceil x \rceil - x \\ \lceil x \rceil, & \text{w.p.} \quad x - \lfloor x \rfloor \end{cases}$$

Recall: $\frac{S_{\text{left}}^2}{n_{\text{left}}} + \frac{S_{\text{right}}^2}{n_{\text{right}}} - \frac{S_P^2}{n_P}, \mathrm{E}[\hat{S}] = \mathrm{E}[S]$

Quantized Training – Theorem and Implementation

- Quantization does not affect the selection of split much

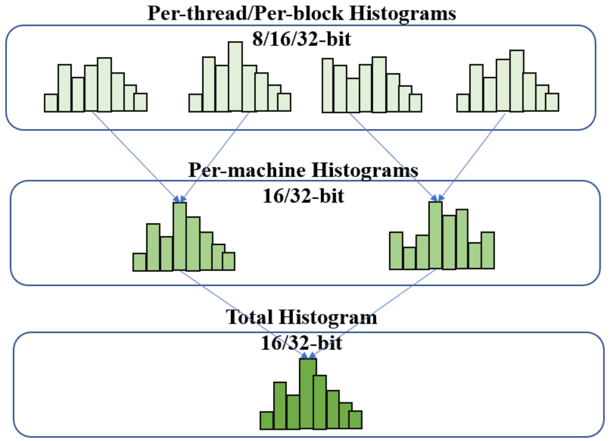
Theorem 5.3 For loss functions with constant hessian value $h > 0$, if Assumption 5.2 holds for the subset $\mathcal{D}_s$ in leaf s for some $\gamma_s > 0$, then with stochastic rounding and leaf-value refitting, for any $\epsilon > 0$, and $\delta > 0$, at least one of the following conclusions holds:

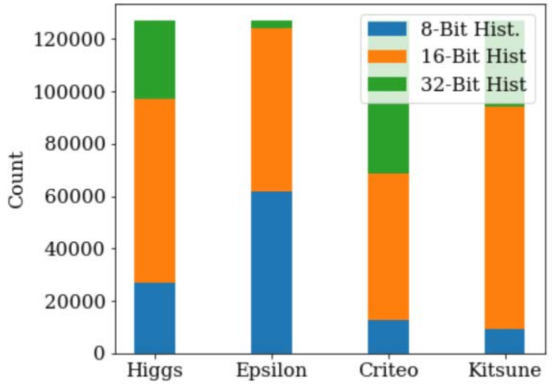
1. With any split of leaf s and its descendants, the resultant average of absolute values of prediction values by the tree in the current boosting iteration for data in $\mathcal{D}_s$ is no greater than ϵ/h.

2. For any split $s \to s_1, s_2$ of leaf s, with a probability of at least $1-\delta$,

$$\frac{\left|\widetilde{\mathcal{G}}_{s\to s_1,s_2} - \mathcal{G}_{s\to s_1,s_2}\right|}{\mathcal{G}_s^*} \le \frac{\max_{i\in[N]}|g_i|\sqrt{2\ln\frac{4}{\delta}}}{\gamma_s^2\epsilon\cdot 2^{B-1}}\left(\sqrt{\frac{1}{n_{s_1}}}+\sqrt{\frac{1}{n_{s_2}}}\right)+\frac{\left(\max_{i\in[N]}|g_i|\right)^2\ln\frac{4}{\delta}}{\gamma_s^2\epsilon^2 n_s\cdot 4^{B-2}}. \tag{9}$$

- Hierarchical Histogram Buffers





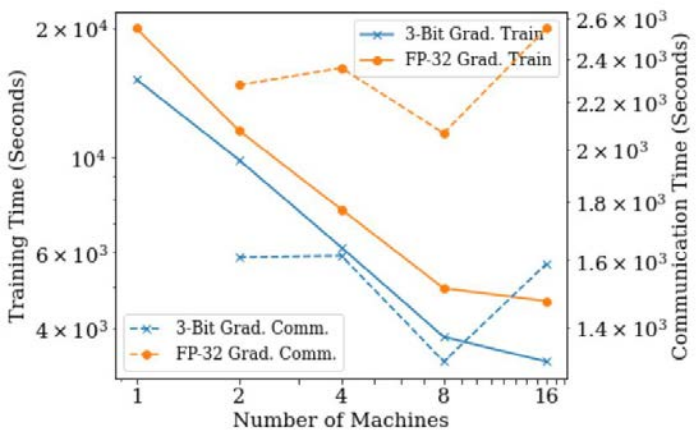
Quantized Training

Table 2: Comparison of accuracy, w.r.t. different quantized bits.

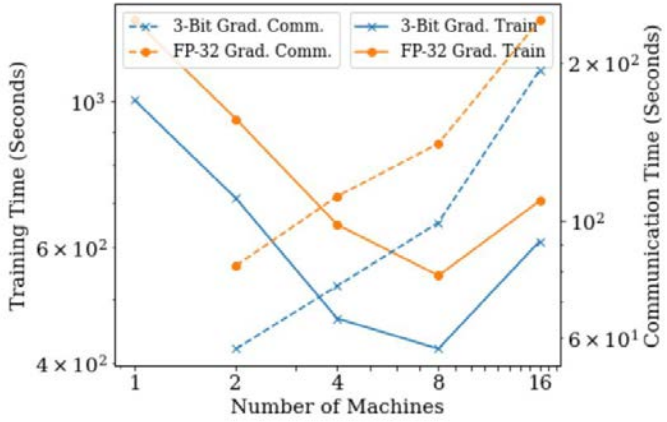
Algorithm	Binary Classification					Regression	Ranking	
	Higgs↑	Epsilon↑	Kitsune↑	Criteo↑	Bosch↑	Year↓	Yahoo LTR↑	LETOR↑
XGBoost	0.845778	0.950210	0.948329	0.802030	**0.706423**	8.954460	**0.794919**	0.505058
CatBoost	0.845425	0.943211	0.944557	0.803150	0.687795	8.951745	0.794215	0.519952
LightGBM	0.845694	0.950203	0.950561	0.803791	0.703471	8.956278	0.793792	0.524191
2-bit SR_{refit}	0.845587	0.949472	0.952703	0.803293	0.700040	8.953388	0.788579	0.519067
3-bit SR_{refit}	0.845725	0.949884	0.951309	0.803768	0.702025	**8.937374**	0.791077	0.522220
4-bit SR_{refit}	0.845507	0.950049	0.950911	0.803783	0.702959	8.942898	0.792664	0.523702
5-bit SR_{refit}	0.845706	0.950298	0.949229	0.803766	0.703242	8.948542	0.793166	**0.524616**

Table 3: Detailed time costs for different algorithms in different datasets (seconds).

	Algorithm	Higgs	Epsilon	Kitsune	Criteo	Bosch	Year	Yahoo LTR	LETOR
GPU total time	XGBoost	33.97	311.12	181.24	326.82	68.44	20.47	28.64	51.29
	CatBoost	61.10	105.00	80.20	187.80	22.12	33.96	59.22	N/A
	LightGBM+	29.05	87.12	77.43	102.33	21.41	24.33	30.79	41.79
	LightGBM+ 2-bit	24.78	**39.04**	**38.26**	61.04	12.57	**18.19**	**23.09**	**33.60**
	LightGBM+ 3-bit	**24.45**	39.25	38.63	59.93	12.60	18.24	24.93	33.87
	LightGBM+ 4-bit	24.53	39.82	40.00	**59.49**	12.55	18.34	25.65	34.11
	LightGBM+ 5-bit	24.55	41.30	40.83	60.24	**12.08**	18.41	25.50	34.36
CPU total time	XGBoost	109.16	1282.97	281.72	565.52	130.92	28.85	103.87	72.37
	CatBoost	1009.8	1283.4	1495.0	7702.2	998.4	95.8	588.2	865.4
	LightGBM	83.27	519.89	332.12	524.61	59.94	12.67	75.44	103.09
	LightGBM 2-bit	73.36	**426.50**	215.91	444.28	46.63	12.94	61.50	**72.08**
	LightGBM 3-bit	69.64	459.39	**207.96**	440.68	47.35	12.79	**61.07**	74.35
	LightGBM 4-bit	**69.30**	458.62	208.99	**416.60**	**46.45**	11.90	61.15	77.66
	LightGBM 5-bit	69.86	457.68	211.53	423.80	47.52	**11.79**	61.76	77.92
GPU Hist. time	LightGBM+	11.26	46.96	54.77	70.97	16.57	9.61	11.59	17.75
	LightGBM+ 2-bit	4.84	12.11	16.41	21.74	8.52	4.08	8.23	10.20
CPU Hist. time	LightGBM	50.74	458.46	253.07	385.98	53.08	6.68	58.53	66.39
	LightGBM 2-bit	32.82	375.70	147.10	269.00	39.80	5.99	43.59	38.23



(a) Epsilon-8M



(b) Criteo

Dynamic Categorical Feature Encoding

$x_3 \in \{1,3\}$?

Number of binary splits for a categorical feature with K values is $2^{K-1} - 1$

K can be extremely large (e.g. 10,000+)

Luckily, the optimal split can be found by:

1. Encode the categorical value c's of feature j by $\frac{\sum_{i:\ x_{ij}=c} g_i}{\sum_{i:\ x_{ij}=c} h_i}$
2. Find split in the same way as a numerical feature

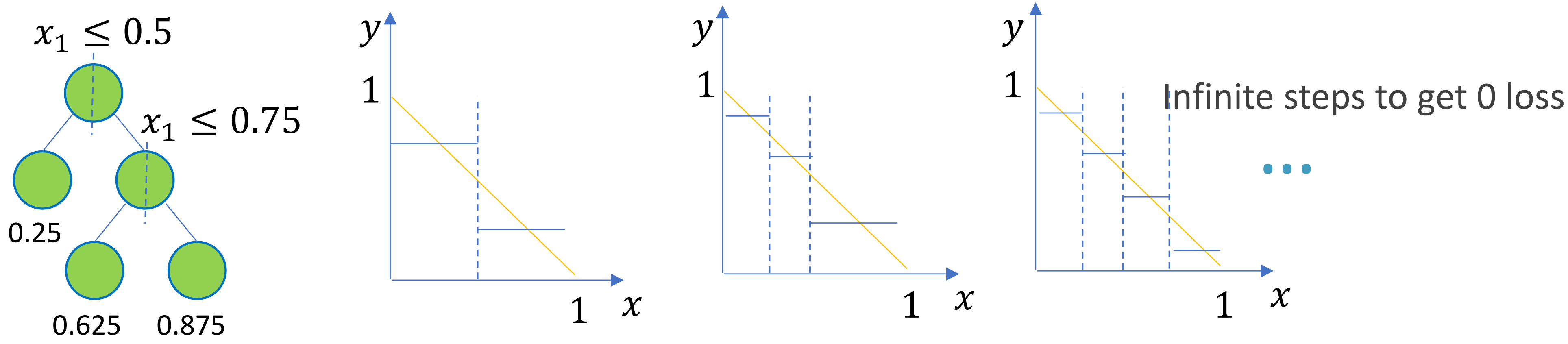
Need regularizations:

1. Restrict the size of set in the split condition
2. Smoothing: $\frac{\sum_{i:\ x_{ij}=c} g_i}{\sum_{i:\ x_{ij}=c} h_i \quad + \boldsymbol{a}}$
3. Extra L2 penalty for categorical splits (bigger γ in split gain calculation)

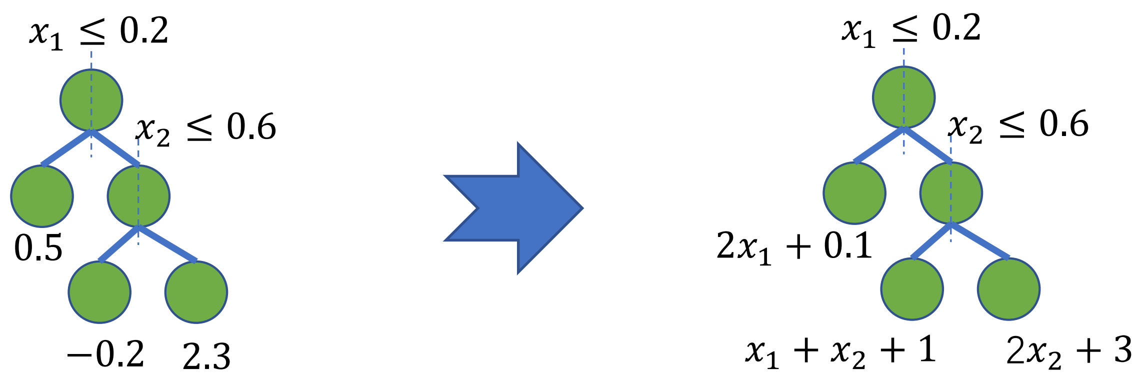
$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

Linear Trees for GBDT

- Piecewise constant trees limits the flexibility (fit function $y = 1 - x$)



- More flexible base learners: Piecewise Linear Regression Trees (PL Trees)



Piecewise Constant **Piecewise Linear**

CatBoost Highlights

- Ordered Boosting for Unbiased Gradients
- Unbiased Categorical Feature Encoding
- Oblivious Tree Structure
- Fast GPU Acceleration

Ordered Boosting

Gradient boosting steps has bias

What we want: $f_t^* = \mathrm{argmin}_{f_t}\ E_{(x,y)\sim p(x,y)}[(f_t(x) - (-\hat{g}_t))^2]$, p is the ground truth data distribution

What we have in fact: $f_t{}' = \mathrm{argmin}_{f_t} E_{(x,y)\sim D'}[(f_t(x) - (-\hat{g}_t))^2]$, D' is the training data sampled from p

Only if D' is independent with $\hat{g}_t$, f_t' is an unbiased estimation of f_t^*

Algorithm 1: Ordered boosting

input : $\{(\mathbf{x}_k, y_k)\}_{k=1}^n, I$;

$\sigma \leftarrow$ random permutation of $[1, n]$;
$M_i \leftarrow 0$ for $i = 1..n$;
for $t \leftarrow 1$ **to** I **do**
  for $i \leftarrow 1$ **to** n **do**
    $r_i \leftarrow y_i - M_{\sigma(i)-1}(\mathbf{x}_i)$;
  for $i \leftarrow 1$ **to** n **do**
    $\Delta M \leftarrow$ $LearnModel((\mathbf{x}_j, r_j) : \sigma(j) \leq i)$;
    $M_i \leftarrow M_i + \Delta M$;
return M_n

Maintain n boosting models!
Reduce cost by maintaining $\log n$ boosting models.

Unbiased Categorical Feature Encoding

Encoding with target values is powerful:

$$\hat{x}_{ij} = \frac{\sum_{k \in D} 1_{x_{kj} = x_{ij}} \cdot y_k}{\sum_{k \in D} 1_{x_{kj} = x_{ij}}}$$ which approximates $E[y|x_j = x_{ij}]$

Again, $\hat{x}_{ij}$ uses y_i, thus has bias, and can easily cause overfitting

Ordered target encoding:

$$\hat{x}_{ij} = \frac{\sum_{k < \mathrm{i}} 1_{x_{kj} = x_{ij}} \cdot y_k}{\sum_{k < \mathrm{i}} 1_{x_{kj} = x_{ij}}}$$

Neural Networks for Tabular Data

GBDT vs. Deep Learning

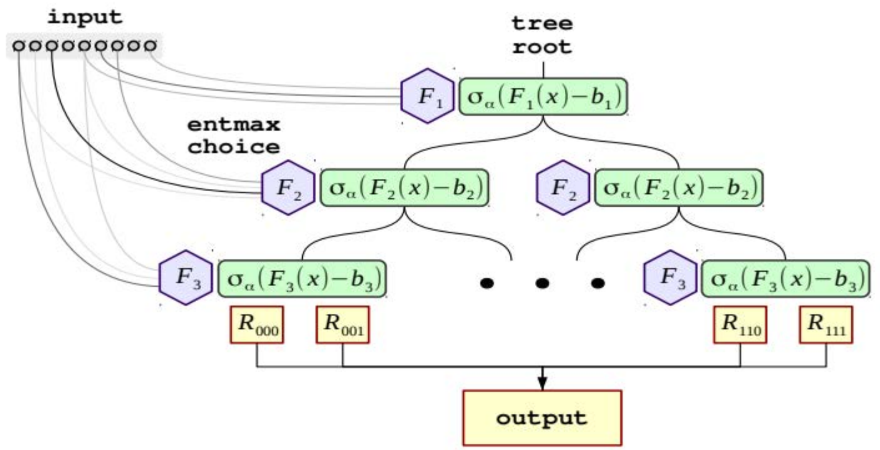
- Deep Learning (NN) is the model with human prior knowledge
  - Special structures designed by human to automatically extract useful information from data
    - CNN: “local receptive fields” from human vision
    - RNN: context in text/speech
    - Transformer variants: adaptation for different tasks
  - Therefore, DL works very well for image, text and speech
  - However, need to design a new structure when applied in new tasks/data
- GBDT is a powerful function approximator, with excellent trade-off between bias and variance
  - No special design in models, can approximate all kinds of distribution
  - Therefore, GBDT works well for tabular data in many tasks, such as click prediction, recommendation, etc.
  - However, GBDT doesn’t have prior knowledge to extract useful information, therefore, feature engineering is often needed for better performance

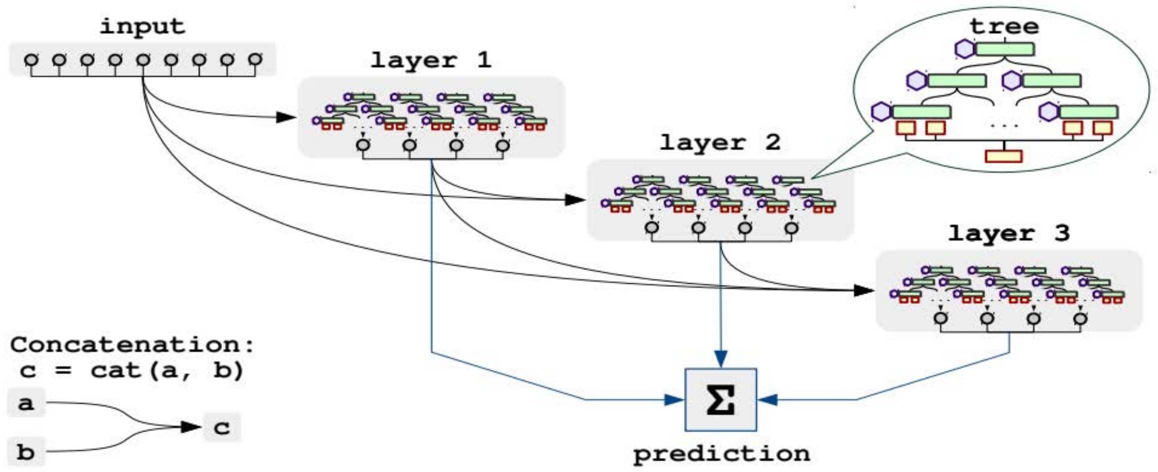
GBDT vs. Deep Learning

	GBDT	NN
Tasks	For all kinds of tabular data	Image, Text, Speech
Human efforts	Feature engineering	Architecture design, Hyper-parameter tuning
Resource consumption	CPU	GPU
Auto feature selection	Yes	No
Mini-batch training	No	Yes
Fine tuning	Difficult	Yes
Categorical feature	Encoding	Embedding

NN in Tree Style

- Neural Oblivious Decision Ensembles





Basic Components: Tree-like NN

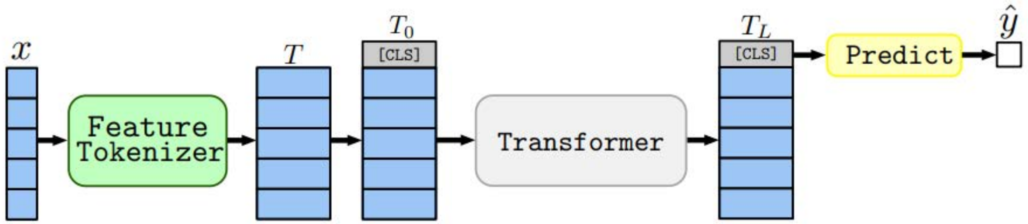
Ensemble and Stacking

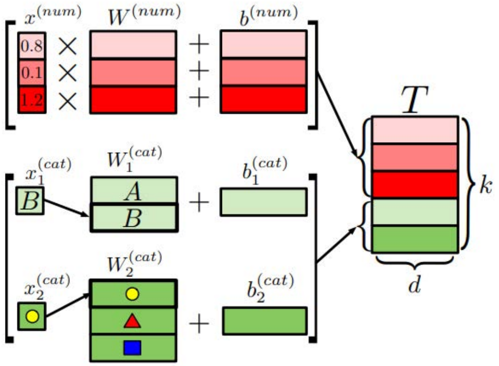
Feature selection: $F_i(x) = \mathrm{entmax}(\mathbf{F}_i \cdot [x_1, \ldots, x_d]^T) \cdot \boldsymbol{x}$

Sigmoid function σ instead of hard split: $\frac{1}{1+\exp(F_i(\boldsymbol{x})-b_i)}$ instead of $I[F_i(\boldsymbol{x}) \leq b_i]$

Transformer-based

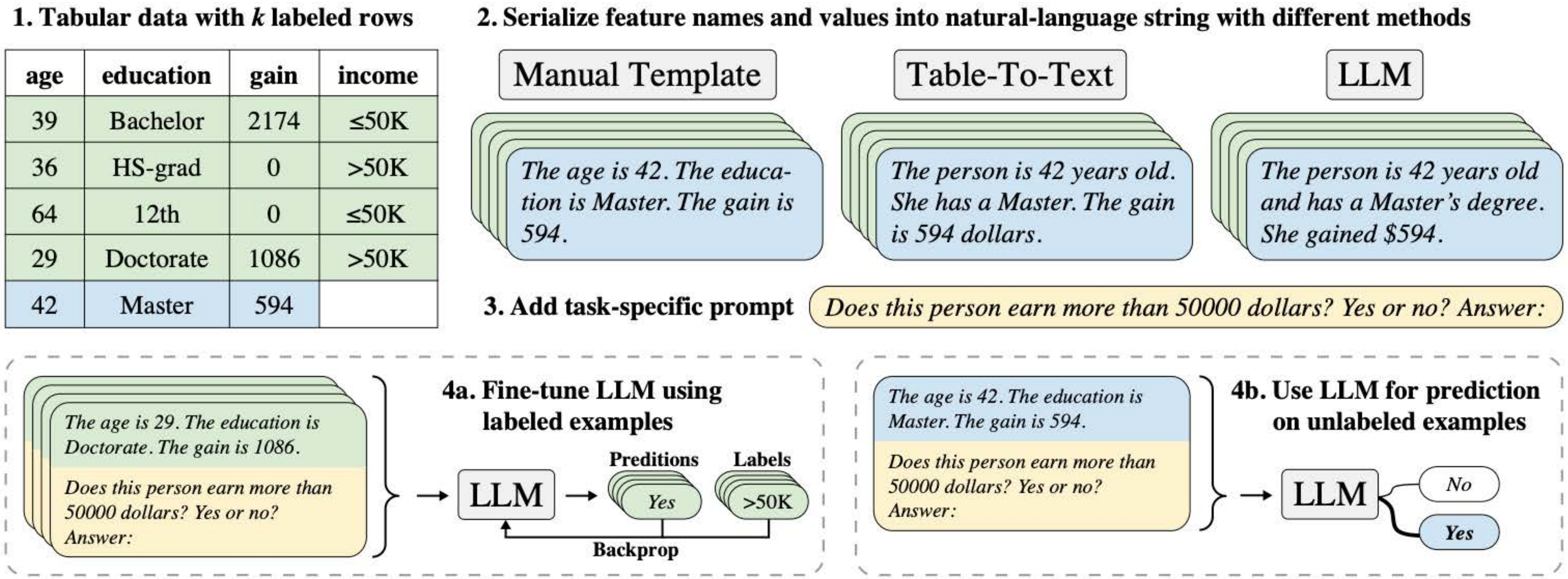
- Simple variant of Transformers



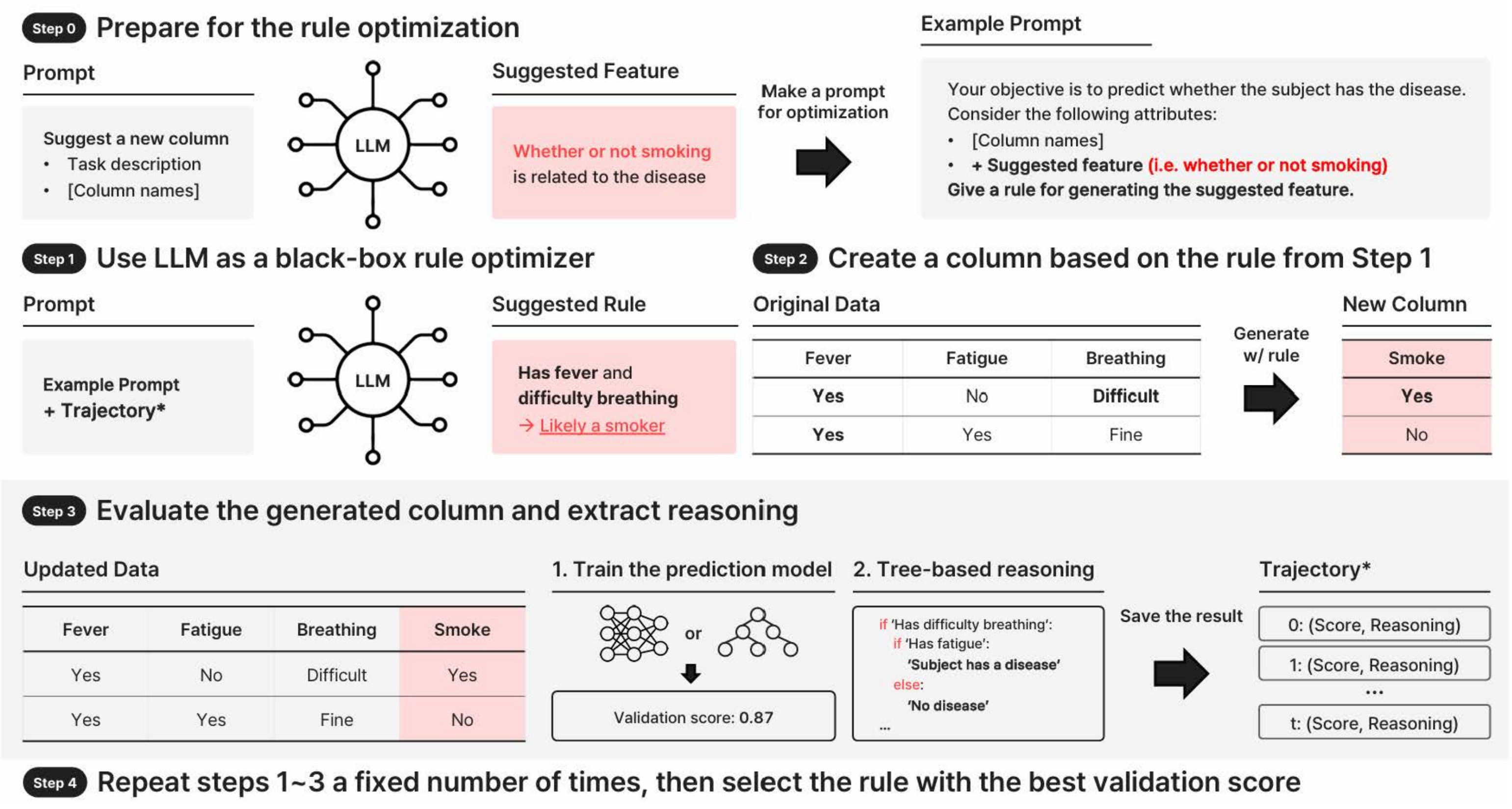


Feature Tokenizer

LLM for Tabular Data Task



LLM as Feature Engineer



GBDT Practices

GBDT Hyper-parameter Tuning

- Most important hyper-parameters
  - Number of iterations M, shrinkage rate γ, number of leaves C
- A common process
  - Fix M to a small value, e.g. 100, and γ to a large value, e.g. 0.1
  - Tune leaves C
  - Fine-tune M and γ
- Some other important hyper-parameters
  - Minimal data per leaf
  - Feature sampling per tree/node

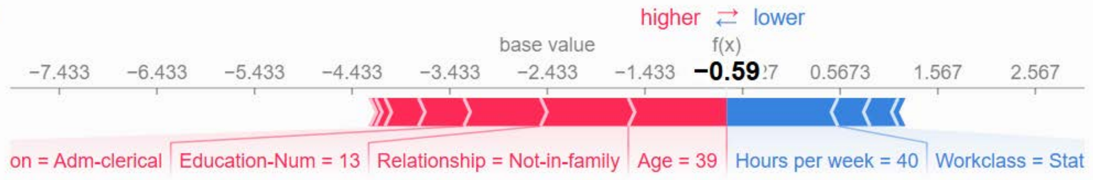
SHAP Values

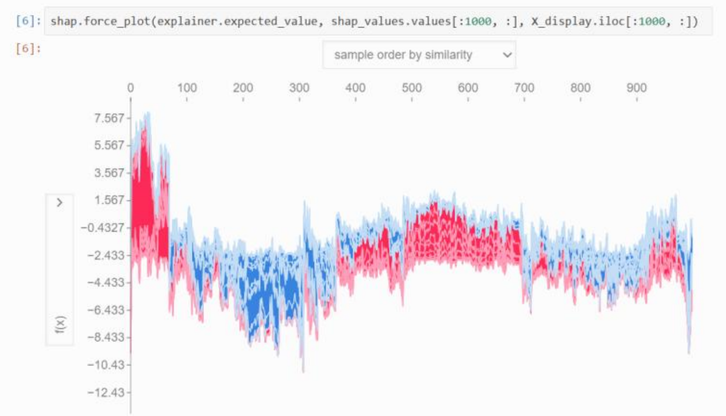
- SHAP value calculation with LightGBM
  - `predict_contrib`, default = `false`, type = bool, aliases: `is_predict_contrib`, `contrib`
    - used only in `prediction` task
    - set this to `true` to estimate SHAP values, which represent how each feature contributes to each prediction
    - produces `#features + 1` values where the last value is the expected value of the model output over the training data
    - **Note**: if you want to get more explanation for your model's predictions using SHAP values like SHAP interaction values, you can install shap package
    - **Note**: unlike the shap package, with `predict_contrib` we return a matrix with an extra column, where the last column is the expected value
    - **Note**: this feature is not implemented for linear trees
- SHAP Package: calculation and visualization of SHAP values
  - Out-of-the-shelf support for LightGBM and XGBoost models

```
[4]: explainer = shap.TreeExplainer(model)
     shap_values = explainer(X)
```

```
[5]: shap.force_plot(explainer.expected_value, shap_values.values[0, :], X_display.iloc[0, :])
```

[5]:





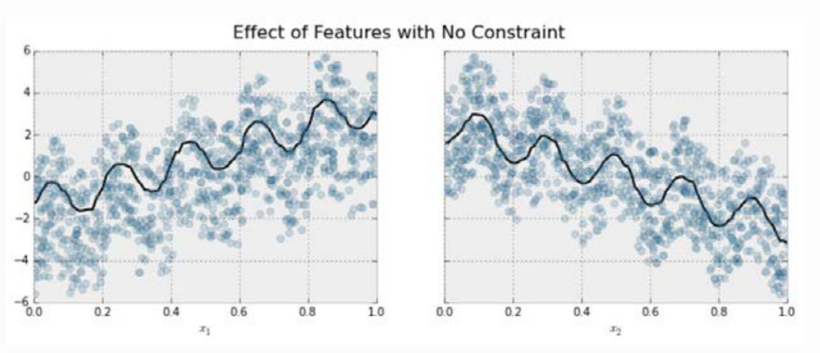
Census income classification with LightGBM — SHAP latest documentation

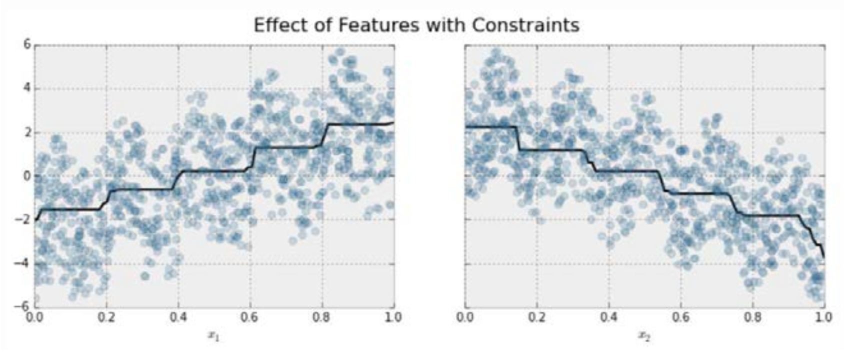
https://shap.readthedocs.io/en/latest/example_notebooks/tabular_examples/tree_based_models/Census%20income%20classification%20with%20LightGBM.html#Explain-predictions

Monotonic Constraints

- Enforce prior knowledge of a feature contribution to the output

$$y = 5x_1 + \sin(10\pi x_1) - 5x_2 - \cos(10\pi x_2) + N(0, 0.01) x_1, x_2 \in [0, 1]$$

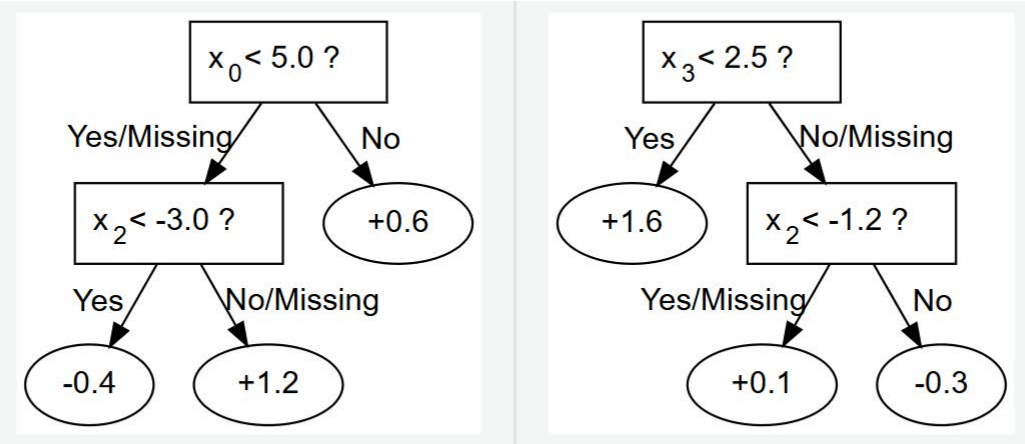




- `monotone_constraints`, default = `None`, type = multi-int, aliases: `mc`, `monotone_constraint`, `monotonic_cst`
  - used for constraints of monotonic features
  - `1` means increasing, `-1` means decreasing, `0` means non-constraint
  - you need to specify all features in order. For example, `mc=-1,0,1` means decreasing for the 1st feature, non-constraint for the 2nd feature and increasing for the 3rd feature

Feature Interaction Constraints

- Allow only subset of features to interact in models
  - Split feature into subsets [0, 1, 2, 3, 4] -> [0, 1], [2, 3, 4]
  - Only interaction within a subset is allowed



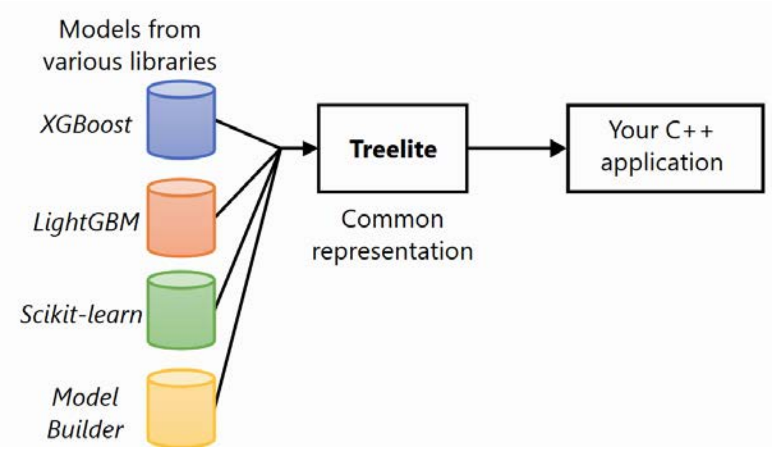
forbidden | allowed

- `interaction_constraints`, default = `""`, type = string
  - controls which features can appear in the same branch
  - by default interaction constraints are disabled, to enable them you can specify
    - for CLI, lists separated by commas, e.g. `[0,1,2],[2,3]`
    - for Python-package, list of lists, e.g. `[[0, 1, 2], [2, 3]]`
    - for R-package, list of character or numeric vectors, e.g. `list(c("var1", "var2", "var3"), c("var3", "var4"))` or `list(c(1L, 2L, 3L), c(3L, 4L))`. Numeric vectors should use 1-based indexing, where `1L` is the first feature, `2L` is the second feature, etc.
  - any two features can only appear in the same branch only if there exists a constraint containing both features

Inference Speedup – Compile to C/C++

- Treelite + TL2cgen

```python
model = treelite.Model.load("../LightGBM_model.txt", model_format="lightgbm")
tl2cgen.generate_c_code(model, dirpath="./code_dir", params={})
```




```c
void predict(union Entry* data, int pred_margin, double* result) {
  unsigned int tmp;
  if ( LIKELY( !(data[26].missing != -1) || (data[26].fvalue <= (double)1.0740148425100000234) ) ) {
    if ( UNLIKELY( !(data[26].missing != -1) || (data[26].fvalue <= (double)0.6540409028530002056) ) ) {
      if ( LIKELY( !(data[28].missing != -1) || (data[28].fvalue <= (double)0.8715623915195001015) ) ) {
        if ( LIKELY( !(data[6].missing != -1) || (data[6].fvalue <= (double)0.8728793561455000516) ) ) {
          if ( LIKELY( !(data[27].missing != -1) || (data[27].fvalue <= (double)0.7548422217370001075) ) ) {
            if ( UNLIKELY( !(data[25].missing != -1) || (data[25].fvalue <= (double)0.6902507841585000525) ) ) {
              if ( LIKELY( !(data[10].missing != -1) || (data[10].fvalue <= (double)1.0173631906550000249) ) ) {
                if ( LIKELY( !(data[27].missing != -1) || (data[27].fvalue <= (double)0.7035730481150000992) ) ) {
                  if ( LIKELY( !(data[28].missing != -1) || (data[28].fvalue <= (double)0.6658997833730001537) ) ) {
                    if ( LIKELY( !(data[14].missing != -1) || (data[14].fvalue <= (double)1.0648799538600000228) ) ) {
                      result[0] += 0.07753606129586194;
                    } else {
                      result[0] += 0.13399499888079566;
                    }
                  } else {
                    result[0] += 0.05036858247180605;
                  }
                } else {
                  if ( LIKELY( !(data[28].missing != -1) || (data[28].fvalue <= (double)0.7350653409955000273) ) ) {
                    result[0] += 0.14279365711277944;
                  } else {
                    result[0] += 0.08120157394259939;
                  }
                }
              } else {
                result[0] += 0.14092462566396044;
              }
            } else {
              if ( UNLIKELY( !(data[23].missing != -1) || (data[23].fvalue <= (double)0.7908223271370001806) ) ) {
                if ( LIKELY( !(data[1].missing != -1) || (data[1].fvalue <= (double)1.5426751971200000341) ) ) {
                  if ( UNLIKELY( !(data[22].missing != -1) || (data[22].fvalue <= (double)0.7982741296290001287) ) ) {
                    result[0] += 0.05666932064429683;
                  } else {
                    result[0] += 0.10392140214425577;
                  }
                } else {
                  result[0] += 0.14574560591441468;
                }
              } else {
```

Inference Speedup – Compile to LLVM

- lleaves

```
lgbm_model = lightgbm.Booster(model_file="NYC_taxi/model.txt")
%timeit lgbm_model.predict(df)
# 12.77s

llvm_model = lleaves.Model(model_file="NYC_taxi/model.txt")
llvm_model.compile()
%timeit llvm_model.predict(df)
# 0.90s
```

batchsize	10,000	100,000	678,000
LightGBM	95.14ms	992.47ms	7034.65ms
ONNX Runtime	38.83ms	381.40ms	2849.42ms
Treelite	38.15ms	414.15ms	2854.10ms
`lleaves`	5.90ms	56.96ms	388.88ms

```
define private double @tree_0(double %.1, double %.2, double %.3) {
node_0:
  %.5 = fcmp ule double %.2, 0x3FE768089A419B12    ; decimal = ~0.731
  br i1 %.5, label %node_1, label %node_2

node_1:                                           ; preds = %node_0
  %.7 = fcmp ule double %.3, 0x3FED06D4513F4FE5    ; decimal = ~0.907
  br i1 %.7, label %leaf_0, label %leaf_2

node_2:                                           ; preds = %node_0
  %.11 = fcmp ule double %.3, 0x3FEB60631F166F7A   ; decimal = ~0.856
  br i1 %.11, label %leaf_1, label %leaf_3

leaf_0:                                           ; preds = %node_1
  ret double 0x3FDFAFD3A55B8741                   ; decimal = ~0.495

leaf_2:                                           ; preds = %node_1
  ret double 0x3FE038704B651588                   ; decimal = ~0.507

leaf_1:                                           ; preds = %node_2
  ret double 0x3FE034DEA54DFC96                   ; decimal = ~0.506

leaf_3:                                           ; preds = %node_2
  ret double 0x3FDF62CFF241EA8B                   ; decimal = ~0.490
}
```

Reference

- [1] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. Annals of statistics, pages 1189–1232, 2001.
- [2] Ruoming Jin and Gagan Agrawal. Communication and memory efficient parallel decision tree construction. In Proceedings of the 2003.
- [3] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Proceedings of the 22Nd ACM SIGKDD International. Conference on Knowledge Discovery and Data Mining, pages 785–794. ACM, 2016.
- [4] Stephen Tyree, Kilian Q Weinberger, Kunal Agrawal, and Jennifer Paykin. Parallel boosted regression trees for web search ranking. In Proceedings of the 20th international conference on World wide web, pages 387–396. ACM, 2011.
- [5] Jerome H Friedman. Stochastic gradient boosting. Computational Statistics & Data Analysis, 38(4):367–378, 2002.
- [6] Zhi-Hua Zhou. Ensemble methods: foundations and algorithms. CRC press, 2012.
- [7] Haijian Shi. Best-first decision tree learning. PhD thesis, The University of Waikato, 2007.
- [8] Ke, et al. "Lightgbm: A highly efficient gradient boosting decision tree." Advances in Neural Information Processing Systems. 2017.
- [9] Ke, et al. "Lightgbm: A highly efficient gradient boosting decision tree." Advances in Neural Information Processing Systems. 2017.
- [10] Meng Q, Ke G, Wang T, et al. A communication-efficient parallel algorithm for decision tree[C]//Proceedings of the 30th International Conference on Neural Information Processing Systems. 2016: 1279-1287.
- [11] Popov S, Morozov S, Babenko A. Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data[C]//International Conference on Learning Representations. 2019.
- [13] Ke G, Xu Z, Zhang J, et al. DeepGBM: A deep learning framework distilled by GBDT for online prediction tasks[C]//Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019: 384-394.
- [14] Kadra A, Lindauer M, Hutter F, et al. Well-tuned Simple Nets Excel on Tabular Datasets[J]. Advances in Neural Information Processing Systems, 2021, 34.
- [15] Gorishniy Y, Rubachev I, Khrulkov V, et al. Revisiting deep learning models for tabular data[J]. Advances in Neural Information Processing Systems, 2021, 34.
- [16] Hegselmann S, Buendia A, Lang H, et al. TabLLM: few-shot classification of tabular data with large language models[C]//International Conference on Artificial Intelligence and Statistics. PMLR, 2023: 5549-5581.
- [17] Shi Y, Ke G, Chen Z, et al. Quantized Training of Gradient Boosting Decision Trees[C]//Advances in Neural Information Processing Systems.
- [18] Nam, Jaehyun, et al. "Optimized feature generation for tabular data via llms with decision tree reasoning." *Advances in Neural Information Processing Systems* 37 (2024): 92352-92380.