2023
Machine Learning Odyssey

Seungeun Lee

Dept. of Mathematics

Korea University

Part 3

XGBoost: A Scalable Tree Boosting System

Introduction

- End-to-End Learning
- Scalable Machine Learing: An algorithm that can handle large amounts of data without consuming significant resources (e.g. memory) for any given amount of data.
- Weighted Quantile Sketch
- Sparsity-aware Algorithm for Parallel Tree Learning
- Effective Cache-aware block structure for Out-of-Core Tree Learning

Tree Boosting in a Nutshell

Dataset

- n examples, m features
- $D = \{(x_i, y_i)\}, \quad |D| = n$
- $x_i \in \mathbb{R}^m, y_i \in \mathbb{R}$

Tree Ensemble Model

- A tree ensemble model uses K additive functions to predict the output.

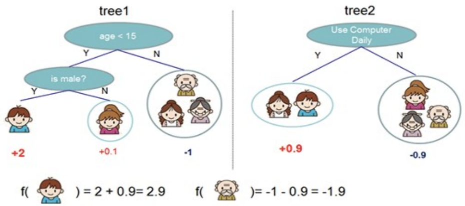


Figure 1: Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree.

Tree Boosting in a Nutshell

- $\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \;\; f_k \in F$
- $F = \{f(x) = w_{q(x)}\} \;\; (q : \mathbb{R}^m \to T, w \in \mathbb{R}^T)$
- F: space of regression trees (CART)
- q: structure of each tree that maps an example to the corresponding leaf index
- T: the number of leaves in the tree
- f_k: independent tree structure q and leaf weights w
- w_i: continuous score on i-th leaf
- Calculate the final prediction by summing up the score in the corresponding leaves

Decision Tree Algorithm

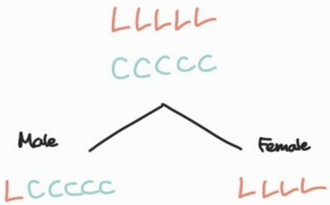
- Automatically discovering data through learning to create tree-based classification rules
- ”How to split the tree?”
- Distribute the data to create the most homogeneous (highest purity) dataset possible

CART [Classification And Regression Tree]

- Gini Index: $G(S) = 1 - \sum_{i=1}^{C} {p_i}^2$
- 1st measurement: chance, at least 2 measurements are needed for accuracy
- Ensuring entities with the same characteristics are grouped together
- Drawback: Overfitting, Regression: RSS
- e.g.) LC [Loyal Customer] *vs.* CC [Churm Customer]

Decision Tree Algorithm

#1. Gender



$$G(\text{parent node}) = 1 - \left(\left(\frac{5}{10}\right)^2 + \left(\frac{5}{10}\right)^2 \right) = 0.5$$

$$G(\text{Male}) = 1 - \left(\left(\frac{5}{6}\right)^2 + \left(\frac{1}{6}\right)^2 \right) = 0.278$$

$$G(\text{Female}) = 1 - \left(\left(\frac{0}{4}\right)^2 + \left(\frac{4}{4}\right)^2 \right) = 0$$

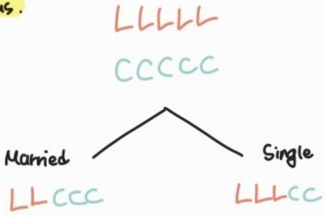
$$G(\text{Gender}) = \frac{6}{10} \times 0.278 + \frac{4}{10} \times 0 = 0.167$$

weight

entropy. uncertainty ↓

Decision Tree Algorithm

#2. Marital Status.



$$G(\text{parent node}) = 1 - \left(\left(\tfrac{5}{10}\right)^2 + \left(\tfrac{5}{10}\right)^2 \right) = 0.5$$

$$G(\text{Married}) = 1 - \left(\left(\tfrac{2}{5}\right)^2 + \left(\tfrac{3}{5}\right)^2 \right) = 0.48$$

$$G(\text{Single}) = 1 - \left(\left(\tfrac{3}{5}\right)^2 + \left(\tfrac{2}{5}\right)^2 \right) = 0.48$$

$$G(\text{Marital Status}) = \tfrac{5}{10} \times 0.48 + \tfrac{5}{10} \times 0.48 = 0.48$$

Gradient Tree Boosting

- Ensemble Model Objective
- $L(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_i \Omega(f_k)$,
- where $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda||w||^2$ (L2 regularization)
- Additional regularization term helps to smooth the final learnt weights to avoid overfitting
- $L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + Constant$
- Sum of the regularization terms up to the (t-1)-th iteration. It is a constant for the current iteration as it does not change, which is why it is omitted in the paper.
- (second-order approximation) $L^{(t)} \simeq \sum_{i=1}^{n}[l(y_i, \hat{y}_i^{(t-1)} + g_i f_t(x_i) + \frac{1}{2}h_i(f_t)^2(x_i)] + \Omega(f_t)$
- $\tilde{L}^{(t)} \simeq \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i(f_t)^2(x_i)] + \Omega(f_t)$

Gradient Tree Boosting

- **More about Gradient Statistics**

  Second-order approximation through Taylor Expansion

  ↳ advantages of 2nd-order approx :

  Regardless of the types of loss functions, we can easily optimize the given functions by g_i and h_i in #t iteration.

  $l'(y) = l(y_i, y + f_t(x_i))$; $y = \hat{y}_i^{t-1} - f_t(x_i)$

  $\left(f(x) \approx f(a) + f'(a)(x-a) + \frac{1}{2} f''(a)(x-a)^2 \right)$

  $$L^{(t)} = \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) \right] + \Omega(f_t) + \cancel{\text{Constant}}$$

  $$\simeq \sum_{i=1}^{n} \left[\underbrace{l(y_i, \hat{y}_i^{(t-1)})}_{\text{Constant (already calculated in the previous iteration)}} + \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \underline{f_t(x_i)} + \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) f_t^2(x_i) \right] + \Omega(f_t)$$

  $$\simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

- $\sum_{i=1}^{n} \frac{1}{2} h_i \left(f_t(x_i) - \left(-\frac{g_i}{h_i} \right) \right)^2 + \text{constant}$

$$= \sum_{i=1}^{n} \frac{1}{2} h_i \left(f_t^2(x_i) + 2 \frac{f_t(x_i) g_i}{h_i} + \left(\frac{g_i}{h_i} \right)^2 \right) = \sum_{i=1}^{n} \left(g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) + \frac{g_i}{2h_i} \right)$$

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

Gradient Tree Boosting

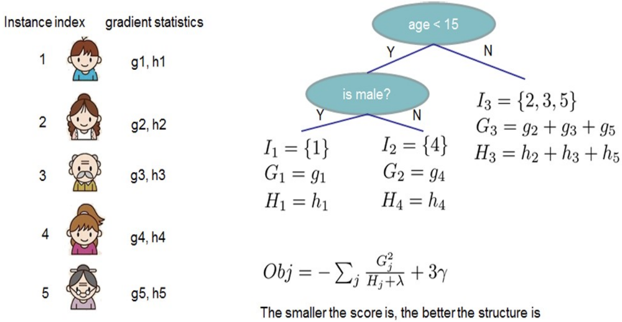


Figure 2: Structure Score Calculation. We only need to sum up the gradient and second order gradient statistics on each leaf, then apply the scoring formula to get the quality score.

Gradient Tree Boosting

- Normally it is impossible to enumerate all the possible tree structures. A greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used instead.
- The goal is to find a split point that minimizes the loss function as much as possible
- $I = I_L \bigcup I_R$, where I_L and I_R are the instance sets of left and right nodes after the split
- $L_{split} = \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma$
- (Loss function before split) - (Loss function after split)

Shrinkage and Column Subsampling

- Introduced to prevent overfitting
- Shrinkage: reduces the influence of each individual tree and leaves space for future trees to improve the model
- scales newly added weights by a factor $\eta \in (0, 1)$ after each step of tree boosting. similar to a learning rate.
- Column Subsampling: user feedback (prevents overfitting even more so than the traditional row subsampling), speed up!

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

Split Finding Algorithms

Basic Exact Greedy Algorithm

- GBM, Single machine version of XGBoost
- enumerate all the possible splits for continuous features
- first sort the data according to the feature values and visit the data in sorted order to accumulate the gradient statistics for the structure score (gain)

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

Approximate Algorithm

- (exact greedy algorithm) What if data does not fit entirely into memory?
- select a subset of candidate split points and choose the best split point from within that subset, rather than considering all possible split points
- Global variants: propose all the candidate splits during the initial phase of tree construction
- Local variants: re-propose after each split (iteration), require more computation
- smaller eps, more candidates (continued in section ”Weighted Quantile Sketch”)

Split Finding Algorithms

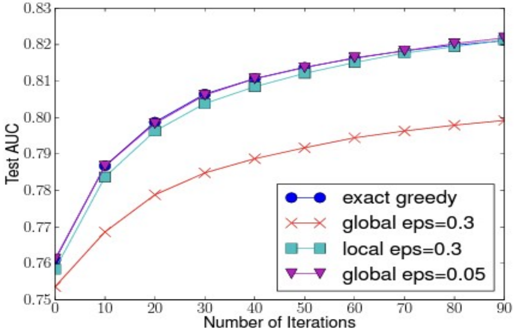
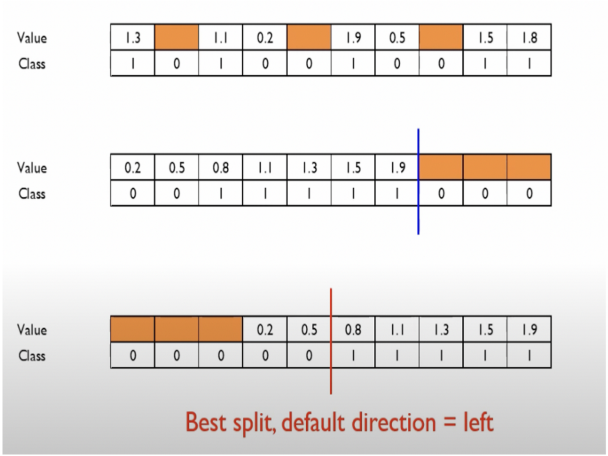


Figure 3: Comparison of test AUC convergence on Higgs 10M dataset. The eps parameter corresponds to the accuracy of the approximate sketch. This roughly translates to 1 / eps buckets in the proposal. We find that local proposals require fewer buckets, because it refine split candidates.

Sparsity-aware Split Finding

- Missing values, frequent zero entries, artifacts of feature engineering (e.g. one-hot encoding)
- Youtube lecture by Prof. Pilsung Kang @DSBA Lab, Korea Univeristy



Split Finding Algorithms

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

Sparsity-aware Split Finding

- Placing all missing values once on the right and once on the left to find the split point.
- (example above) By placing all missing values on the left, we can find a better split point. Therefore, in that branch, the default direction for classifying missing data is set to the left leaf.

To Be Continued...

- Weighted Quantile Sketch, System Design, How to Tune XGBoost, Visualization