XGBoost: A Scalable Tree Boosting System

by Tianqi Chen and Carlos Guestrin, presented by Scott Sievert and Zhenyu Zhang

2016-11-15, ECE 901, UW–Madison

Boosted trees

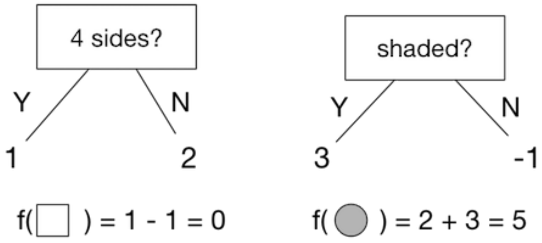


Figure 1: A boosted tree

- Can be used for regression and prediction with different losses
- Invariant to feature scaling
- Scales well as number of examples and feature dimension grow
- Performs well in practice (in 17/29 winning Kaggle solutions and in industry)

Boosted trees

- Given k decision trees evaluated on example i, the estimate of y_i is

$$\widehat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i)$$

where f_k is an independent tree structure with T leaves and weights w. $\mathbf{x}_i$ is one feature vector.

- One possible f_k would be $f_k(\mathbf{x}) = \begin{cases} 1 & \textit{if } \mathbf{x}_{i,0} > 5 \\ 0 & o.w. \end{cases}$.
- In this, there are $T = 2$ leaves and the leaf weights $\mathbf{w} = [1, 0]^T$.

Optimization

We have elements to optimize over; we want to minimize the error ℓ for each example and the complexity of each tree with $\Omega(f_k)$

$$\mathcal{L}(\phi) = \sum_i \ell(\widehat{y}_i, y_i) + \sum_k \Omega(f_k)$$
$$\text{where } \Omega(f) = \gamma T + \frac{\lambda}{2} \|w\|^2$$

- ℓ is differentiable convex loss function
- Ω measures the number of leaves T and the L2 norm of the weights w for each tree
- γ and λ are two regularization parameters

We can use other techniques (shrinkage method and feature subsampling) to prevent overfitting (Breiman 2001, Friedman 2002)

Global minimization

This is a challenging optimization problem.

- We are minimizing over *function* values, not parameters. Each function we're regularizing over has parameters associated with it.
- To find a global optimum, we have to list all possible trees.
  - Given discrete data this scales poorly
  - Given continuous data this is consuming to calculate even for small n

Local minimization

On the t-th iteration, we want to find the tree that minimizes the losses and has a low complexity.

Formally, at iteration t we want to minimize

$$\mathcal{L}^{(t)} = \sum_i \ell(y_i, \widehat{y}_i^{t-1} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

This is a greedy approach; it will converge to a local minima.

Expanding $\mathcal{L}$

When we define

- $g_i = \partial_{\widehat{y}_i^{t-1}} \ell(y_i, \widehat{y}^{t-1}) \in \mathbb{R}$
- $h_i = \partial^2_{\widehat{y}_i^{t-1}} \ell(y_i, \widehat{y}^{t-1}) \in \mathbb{R}$

we can use Taylor's thm to expand $\mathcal{L}$ as

$$\begin{aligned}\mathcal{L}^{(t)} &= \sum_i \ell(y_i, \widehat{y}_i^{t-1} + f_t(\mathbf{x}_i)) + \Omega(f_t) \\ &\approx \left(\sum_i \ell(y_i, \widehat{y}_i^{t-1}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right) + \gamma T + \frac{\lambda}{2} \|w\|^2\end{aligned}$$

If we remove the constant terms $\ell(y_i, \widehat{y}_i)$, we can say that

$$\mathcal{L}^{(t)} = \left(\sum_i g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right) + \gamma T + \frac{\lambda}{2} \|w\|^2$$

Expanding $\mathcal{L}$

- If we define I_j as the examples that “fall” to leaf j with $I_j = \left\{ i \,\middle|\, q(\mathbf{x}_i) = j \right\}$.
- Using this we can write $f_t(\mathbf{x}_i) = \sum_{k \in I_j} w_k$ because one example may fall to different leaves

Then we can rewrite the loss as

$$\begin{aligned}
\mathcal{L}^{t-1} &= \left(\sum_i g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right) + \gamma T + \frac{\lambda}{2} \sum_j w_j^2 \\
&= \left(\sum_{j=1}^{T} \left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right) + \gamma T
\end{aligned}$$

Optimal values for given tree structure

When $x \in \mathbb{R}$, $\operatorname{argmin}_x Gx + \frac{1}{2}Hx^2 = -\frac{G}{H}$ when $H > 0$.
We can use this fact when defining

- $G_j = \sum_{i \in I_j} g_i$
- $H_j = \sum_{i \in I_j} h_i$

Then given a fixed tree structure $q(\mathbf{x})$, we can compute the optimal weight $w_j^\star$ for leaf j by

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

Then the loss on this structure q is given by

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

Splitting

- We can only find the optimal weights for a *given* tree structure and can't compute all possible tree structures.
- Instead we start with one seed leaf and evaluate the cost of splitting that leaf into two leaves.
- If we denote the instances sets that are effected by the binary decision of splitting as I_L and I_R for left and right respectively, we can say that

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

Split finding

- Now we go over split finding – given a set of starting leaves, how can we find the best leave to make a decision at?
- There are an exact and approximate algorithm for this.
- The exact algorithm is used by many ML packages (scikit-learn, R's gdm). This isn't too bad with a discrete set of features.
- However very computationally inefficient when using these boosted trees with continuous features or the dataset doesn't fit into memory.

Approximate algorithm

Main idea:

1. Propose candidate splitting points based on the distribution of the features
2. Map these continuous features into buckets
3. Aggregate the statistics and find the best solution based on the aggregated statistics

There are two variants of this, either “global” or “local”.

- The global method is less refined (and hence needs to generate more split candidates).
- The local proposal refines the candidates after each split/level (which may be appropriate for deeper trees).

Proposing split candidates

- We'd like to split candidates to distribute evenly on the data: given any split, we want to have about half the examples go to each side.
- Formally, if we have a set of features k for example i as $\mathbf{x}_{ik} \in \mathbb{R}$ and second-order gradients as $h_i \in \mathbb{R}$, then the rank function $r_k : \mathbb{R} \to [0, \infty)$

$$r_k(z) = \frac{1}{\sum_i h} \sum_{i=0, \mathbf{x}_{ik} < z}^{n} h_i$$

- The rank function is scaled by the second gradients h because the loss can be rewritten as

$$\mathcal{L}^t = \frac{1}{2} \sum_i h_i (f_t(\mathbf{x}_i) - g_i/h_i)^2 + \Omega(f_t)$$

which means that as h_i gets larger the loss also gets larger.

Rank function objective

For some ϵ, we want to find a set of splitting points $\{s_{k,1}, s_{k,2}, \ldots, s_{k,l}\}$ such that

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon$$

where $s_{k,i} \in [\min_i \mathbf{x}_{i,k}, \max_i \mathbf{x}_{i,k}]$ and the $s_{k,i}$'s are ordered.

- We want bins $s_{k,i}$ and $s_{k,i+1}$ to have approximately the same number of items in them
- This means there are approximately $1/\epsilon$ buckets to bin the continuous features into.

Rank function objective

For some ϵ, we want to find a set of splitting points $\{s_{k,1}, s_{k,2}, \ldots, s_{k,l}\}$ such that

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon$$

where $s_{k,i} \in [\min_i \mathbf{x}_{i,k}, \max_i \mathbf{x}_{i,k}]$ and the $s_{k,i}$'s are ordered.

- We want bins $s_{k,i}$ and $s_{k,i+1}$ to have approximately the same number of items in them
- This means there are approximately $1/\epsilon$ buckets to bin the continuous features into.

Sparsity aware splits

- This ranking can be sparsity aware: they define default directions when a feature isn't present.
- They choose default directions by training the model with different candidate directions and choosing then one with the highest empirical accuracy

Solving this ranking problem

- They want *approximate* answer to "quantile" questions (e.g., "what are the best k items in the list?")
- They solve this problem with a weighted quantile sketch with provable theoretic guarantees.
- This relies on a data structure that supports the "merge" and "prune" operations. In the appendix they show that
  - A merge operation that combines two summaries with approximation error ϵ_1 and ϵ_2 together to create a merged summary with approximation error $\max\{\epsilon_1, \epsilon_2\}$
  - A prune operation that reduces the number of elements in the summary to $b+1$ and changes the approximation error ϵ to $\epsilon + \frac{1}{b}$
- They show the data structure they propose has the same bounds as existing work (Zhang 2007), allowing it to be used as a modular block

System design

- This ranking problem involves binning the features into different buckets based on magnitude.
- If done without any optimization, this involves sorting the features every time
- ... but we could also sort the features once and then do a binary search
- Binary searching n elements happens in $\mathcal{O}(\log n)$ time and the initial sorting only has to be done once

Which means that

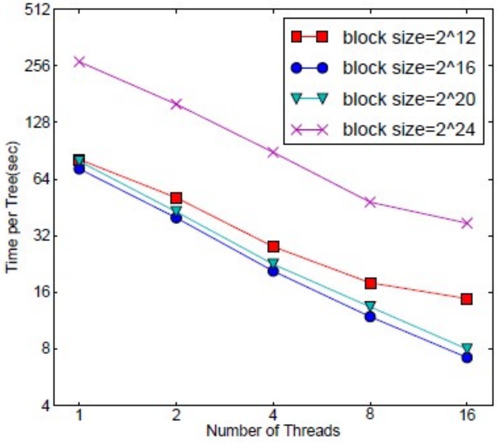
> *The most time consuming part of tree learning is to get the data into sorted order*

Column Block for Parallel Learning

- In order to reduce the time of sorting, data is stored in in-memory units.
- d is the maximum depth of the tree and K be the total number of trees.
- n examples and $\|x\|_0$ represents the number of non-missing enteries in the data
- Original time complexity is $\mathcal{O}(Kd \|x\|_0 \log n)$
- Tree boosting on the block structure cost $\mathcal{O}(\|x\|_0 (Kd + \log n))$ (sorting can parallelized)

Cache-aware Access

- Allocate an internal buffer in each thread, fetch gradient, then accumulate in a mini-batch manner.
- Balance the cache property and parallelization



(b) Higgs 10M

Blocks

Utilize disk space to handle data that does not fit into main memory.

Block Compression

- The block is compressed by columns.
- Subtract the row index by the beginning index of the block and use 16bit int to store each offset.

Block Sharding

- Shard the data onto multiple disks.
- One thread is assigned to each disk and fetch data into an in-memory buffer.
- The training thread alternatively reads the data from each buffer.

Evaluation

XGBoost is fast. A task is to classify whether an event correspond to Higgs boson

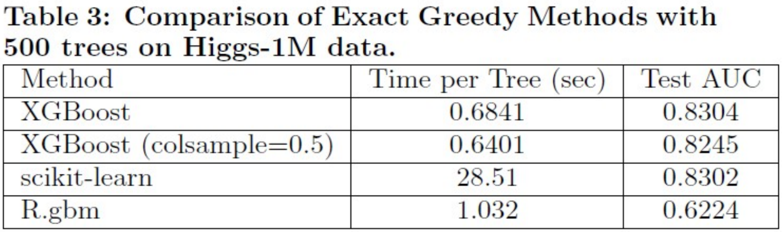


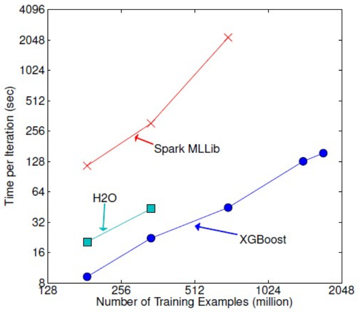
Table 3: Comparison of Exact Greedy Methods with 500 trees on Higgs-1M data.

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

Figure 3:

Evaluation

XGBoost is Scalable. Criteo terabyte click log dataset (1.7 billion) is used in distributed settings to evaluate the scaling property.



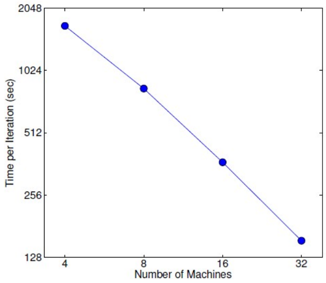


Figure 13: Scaling of XGBoost with different number of machines on criteo full 1.7 billion dataset.

Summary

1. They have a global optimization problem
2. This is difficult so they only consider converging to a local minimum
3. At each iteration they only consider splitting a leaf instead of adding the true minimizing leaf
4. Exact splitting is still computationally difficult so they consider approximate splitting
5. This is still a challenging problem that requires system design.

References

- **(Zhang 2007)**: Zhang, Qi, and Wei Wang. "A fast algorithm for approximate quantiles in high speed data streams." Scientific and Statistical Database Management, 2007. SSBDM'07, 2007.
- **(Breiman 2001)**: Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.
- **(Friedman 2002)**: Friedman, Jerome H. "Stochastic gradient boosting." Computational Statistics & Data Analysis 38.4 (2002): 367-378.