XGBoost: A Scalable Tree Boosting System

Daniil Fishman

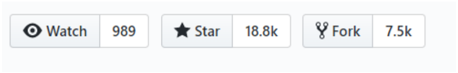
Novosibirsk State University

23.04.2020

What is xgboost?

- ~~Algorithm~~
- ~~Optimization technique~~
- ~~Method of ensemble~~
- Library

What is xgboost? Library

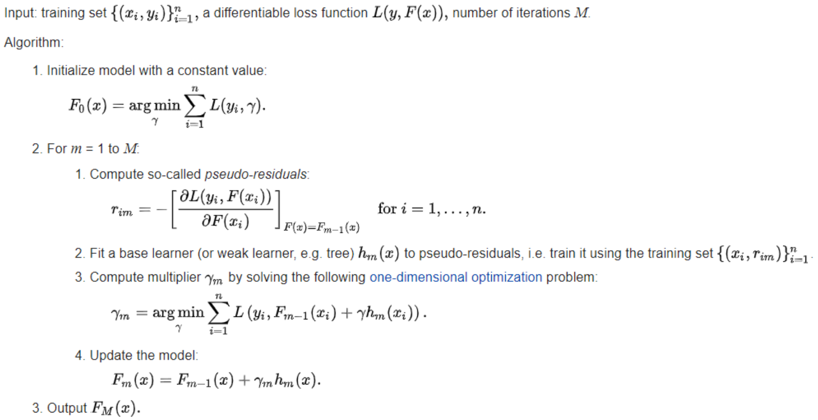


- 90 000+ lines of C++ code.
- API for R, Python, Java, Scala
- Multithred and multinode versions

Why is it so fast?

- Highly scalable end-to-end tree boosting system.
- A novel sparsity-aware algorithm for parallel tree learning.
- Effective cache-aware block structure for out-of-core tree learning.

Gradient boosting idea



Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:
   $$F_0(x) = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L(y_i, \gamma).$$
2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:
      $$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$
   2. Fit a base learner (or weak learner, e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:
      $$\gamma_m = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$
   4. Update the model:
      $$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$
3. Output $F_M(x)$.

Xgbsoost gradient boosting idea

1. Change loss function: $L = l + \Omega$, where Ω is special regularization component combining L_1 and L_2 regularization, l - original loss.

2. Usually, for best split finding one can use either Gini or Entropy criteria. In xgboost, we can obtained more efficient split criteria using second order derivative approximation:

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

Shrinkage and Column Subsampling

Over-fitting preventing techniques:

1. **Shrinkage** scales newly added weights by a factor η after each step of tree boosting. Similar to a learning rate.
2. **Column (feature) subsampling**. According to user feedback, using column sub-sampling prevents over-fitting even more so than the traditional row sub-sampling (which is also supported).

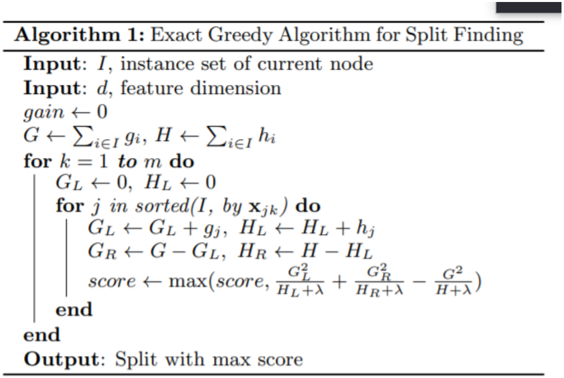
Split finding algorithms

The most time and resource consuming operation is finding best splits. To optimize it following algorithms were proposed:

- Exact Greedy
- Approximate Algorithm
- Histogram-based

Exact Greedy

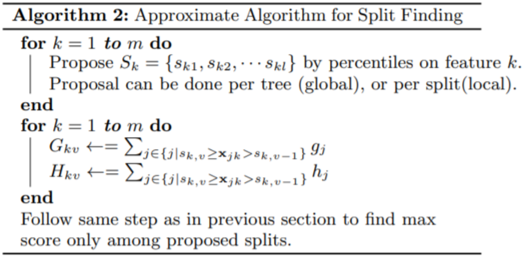
"Naive approach". Simple but highly inefficient in terms of computation power and memory.



```
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
```

Approximate Algorithm

Continuous features are bucketed into discrete bins. It costs $O(\mathit{bin} * \mathit{feature})$ for split point finding.

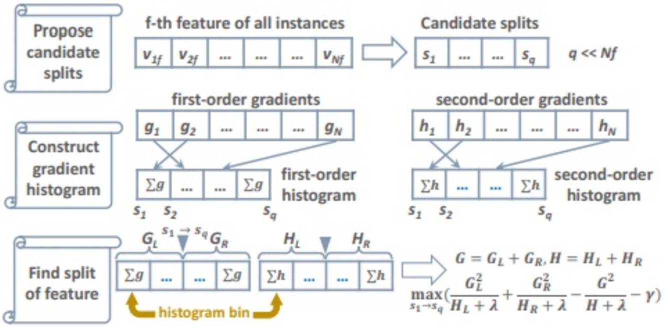


Algorithm 2: Approximate Algorithm for Split Finding

```
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.
```

Histogram-based

Similar to Approximate algorithm, but for each bin we first construct statistic histogram. Then use it to find best split.


Propose candidate splits
f-th feature of all instances
v_{1f} v_{2f} ... ... v_{Nf}
Candidate splits
s_1 ... ... s_q
$q << Nf$
Construct gradient histogram
first-order gradients
g_1 g_2 ... ... g_N
second-order gradients
h_1 h_2 ... ... h_N
Σg ... ... Σg
first-order histogram
Σh ... ... Σh
second-order histogram
s_1 s_2 s_q
Find split of feature
G_L $s_1 \rightarrow s_q$ G_R
H_L H_R
histogram bin
$G = G_L + G_R, H = H_L + H_R$
$\max_{s_1 \rightarrow s_q}\left(\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda} - \gamma\right)$

To propose bin candidates special technique **"Weighted Quantile Sketch"** is used. It aggregates gradient statistics to create uniform bins. More details could be found in the paper.
Another optimization step is **"Sparsity-aware Split Finding"**. Default direction in each tree node added. When a value is missing in the sparse matrix x, the instance is classified into the default direction.

System design

- Column Block for Parallel Learning (Special use of memory, in order to perform better task parallelisation)
- Cache-aware Access (Similar to mini-batch in neural network, fit several objects in memory and perform fast operations in memory)
- Blocks for Out-of-core Computation (Special technique to store and load data from dist in order to share information between operations)

Results

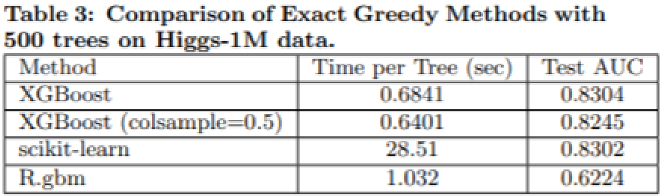


Table 3: Comparison of Exact Greedy Methods with 500 trees on Higgs-1M data.

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

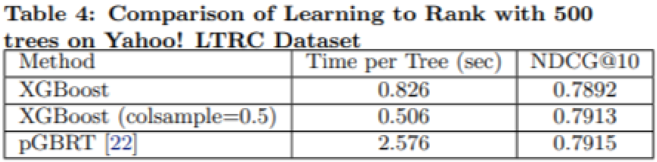


Table 4: Comparison of Learning to Rank with 500 trees on Yahoo! LTRC Dataset

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