XGBoost: a Scalable Tree Boosting System

Date 7/10/17 No. 1/2

Tiangi Chen, Carlos Guestrin (2016)

Gradient tree boosting

Let a dataset with n examples and m features be denoted by

$$D = \{(x_i, y_i)\} \; (|D| = n, \; x_i \in \mathbb{R}^m, \; y_i \in \mathbb{R})$$

A tree ensemble with K additive functions is used to predict output

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F} \leftarrow \text{space of trees}$$

$$\mathcal{F} = \{ f(x) = w_{q(x)} \} \; (q : \mathbb{R}^m \rightarrow T, \; w \in \mathbb{R}^T)$$

q: maps example to leaf index

T: number of leaves

w: leaf weights

To learn the functions, we use the regularised objective:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \leftarrow \text{complexity penalty}$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Because this is a difficult optimisation, we use the greedy additive method.

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

Taylor expansion

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

$$\text{where } g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

$$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

- Let $I_j = \{ i \mid q(x_i) = j \}$ be the instance set of leaf j (i.e the set containing all examples in that leaf.

Removing constant terms and regrouping gives:

$$\mathcal{L}^{(t)} = \sum_{j=1}^{T} \Big[\big(\underbrace{\sum_{i \in I_j} g_i}_{G_j}\big) w_j + \frac{1}{2} \big(\underbrace{\sum_{i \in I_j} h_i}_{H_j} + \lambda\big) w_j^2 \Big] + \gamma T$$

Then for fixed structure $q(x)$, the optimal weights are

$$w_j^* = - \frac{G_j}{H_j + \lambda}$$

$$\Rightarrow \mathcal{L}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

This can be used as a scoring function for evaluating splits:

$$\text{gain} = \frac{1}{2} \Big[\underbrace{\frac{G_L^2}{H_L + \lambda}}_{\text{score of left child}} + \underbrace{\frac{G_R^2}{H_R + \lambda}}_{\text{score of right child}} - \underbrace{\frac{(G_L + G_R)^2}{H_L + H_R + \lambda}}_{\text{score if no split}} \Big] - \underbrace{\gamma}_{\text{complexity cost of split.}}$$

Exact greedy algorithm for split finding

I = instance set of current node

$G \leftarrow \Sigma_{i \in I}\, g_i$, $H \leftarrow \Sigma_{i \in I}\, h_i$

iterate over features

for $k = 1$ to m:

  $G_L = 0$, $H_L = 0$

  for j in sorted(I, by x_{jk}):

    $G_L \mathrel{+}= g_j$, $H_L \mathrel{+}= h_j$

    $G_R = G - G_L$, $H_R = H - H_L$

    score = max(score, gain)

  Left to right search for best split value

output the split with maximum score

We can also regularise with shrinkage, scaling new terms with a small parameter η. We can use column/row subsampling which has the added benefit of speeding up computation.

Alternative split finding algorithms

The exact greedy algorithm enumerates all possible splits and is thus computationally demanding. We can reduce the search space based on the weighted quantile sketch algorithm, which will propose a set of splits $S_K = \{s_{K1}, s_{K2}, s_{K3}, \dots s_{KL}\}$ for feature K.

Sparsity-aware split finding can be implemented by adding a default direction to each node, which will be learnt from non-missing data.

XGBoost system design

- To reduce the cost of sorting: before any learning is done, each column is sorted by feature value then stored in a block.
- The gradient statistics stay in place, referenced with pointers.
- Blocks $\Rightarrow$ parallelisation, and the column structure facilitates feature subsampling.
- Using the presorted blocks, finding quantiles is a linear scan.
- Too small a block size leads to inefficient parallelisation, while too large a block size means that not all of the gradient statistics will fit into the CPU cache. 2^{16} examples per block is a good compromise.
- Out-of-core computation, i.e disk reading/IO often takes a lot of computation time. This is improved by
  - block compression by column
  - block sharding onto multiple disks.