CONSECUTIVE OPTIMIZER FOR XGBOOST

A PREPRINT

Renjun Hu
Institute of Operations Research and Analytics
National University of Singapore
Singapore, 117602
orahr@nus.edu.sg

November 24, 2019

ABSTRACT

XGBoost is among the most popular tree boosting algorithm for prediction tasks. First purposed in 2015, the significance of XGBoost was well recognized in the Kaggle's 2015 blog where 19 out 27 winning solution used XGBoost. The success of XGBoost partly accounts for its utilizing both gradient and hessian information of the residual which increases convergence speed and reduces over-fitting. The structure of loss function with respect to gradient and hessian implies a close connection between XGBoost tree training and the consecutive optimizer in combinatorial optimization. Based on this connection, We purpose a refined XGBoost algorithm with time complexity $\mathcal{O}(KT[d\log n + mn^2])$.

***K*eywords** Machine Learning, Combinatorial Optimization

1 XGBoost Problem Setting and Optimal Partitioning

We follow the prediction task setting in [3]. For a given data set of n examples and d features $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, $\mathbf{x}_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$, a tree ensemble model uses K additive functions to predict the output.

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F}, \tag{1}$$

where $\mathcal{F} = \{f(\mathbf{x} = (w)_{q(\mathbf{x})}(q : \mathbb{R}^d \to T, \mathbf{w} \in \mathbb{R}^T)\}$ is the space of regression trees. Here q represents the structure of each tree that maps an example to the corresponding leaf index. T is the number of leaves in the tree. Each f_k corresponds to a tree structure and leaf weights $\mathbf{w} = (w_1, ..., w_T)$. To learn the set of functions used in the model, we minimized the following regularized objective:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k), \quad where\ \ \Omega(f) = \gamma T + \frac{1}{2}\lambda\|\mathbf{w}\|^2,\ \lambda \geq 0 \tag{2}$$

Here l is a differential convex loss function that measures the difference between the prediction $\hat{y}_i$ and the target y_i. The additional regularization terms helps to smooth the learnt weight to prevent over-fitting. In tree boosting algorithm, the prediction model $\phi(\cdot)$ is trained in an additive manner. Let $\hat{y}_i^{(t)}$ be the prediction of the i-th instance at the t-th iteration, we aim to add f_t to minimize the following objective

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t) \tag{3}$$

We employ second order Taylor expansion to express the loss at t-th iteration. We set $g_i = \partial_{\hat{y}^{t-1}} l(y_i, \hat{y}^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{t-1}} l(y_i, \hat{y}^{(t-1)})$. By eliminating the constant term, the optimizing objective becomes

$$\tilde{\mathcal{L}}^{(t)} \simeq \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{4}$$

Define $I_j = \{i|q(\mathbf{x}_i = j)\}$ as the set of instances in leaf j. We can rewrite Eq(4) by

$$\tilde{\mathcal{L}}^{(t)} = \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T \tag{5}$$

For a fixed tree $q(\mathbf{x})$, we can compute the optimal weight w_j^* of leaf j by

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda}, \tag{6}$$

with the optimal objective value given by

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T. \tag{7}$$

Eq(7) can be viewed as the optimal loss given any structure q. The rest of optimization task becomes finding the best tree q^*, or the heuristically best one (That is what the original XGBoost algorithm does).

We notice that the a decision tree consists of branches as the predicting criterion and the leaves as MECE(mutually exclusive, collectively exhaustive) partitions of instances. We relax the tree to a MECE partition $\{I_j\}_{j=1}^T$ of instances. The optimization goal of the relaxed problem becomes finding a optimal partition, which is generally NP-hard if we don't have prior knowledge of the objective function. However, the prediction loss is in good shape such that we can employ the consecutive optimizer tool to find the optimal partion in polynomial time, which I will state in section 2.

2 Consecutive Optimizer on Relaxed XGBoost

We introduce the concept of consecutive optimizer

Definition 1. *Let $\mathcal{N} = \{1, .., n\}$ be the set of n integers. A subset $\mathcal{I} \subseteq \mathcal{N}$ is called consecutive if its elements are consecutive integers. A MECE partition $P = \{I_1, ..., I_T\}$ is called consecutive if I_t is consecutive for all $t \in \{1, .., T\}$*

We give an example of here: $I_1 = \{1, 2, 3\}$ is a consecutive subset of $\mathcal{N}_{(n=5)}$ and $P = \{I_1, \{4, 5\}\}$ is a consecutive partition. It was proved in [2] that for some combinatorial problem there exists a consecutive partition which is also the global optimum. We state the result formally as the lemma below.

Lemma 1. *Let $\{(a_i, b_i)\}_{i=1}^n$ be the set of real number pairs such that $b_i \geq 0\ \forall i$ and $b_1/a_1 \leq ... \leq b_i/a_i \leq b_{i+1}/a_{i+1} \leq b_n/a_n$. Denote $a_I = \sum_{i\in I} a_i$ and $b_I = \sum_{i\in I} b_i$. Given a real-valued function g satisfying separable condition $g(I_1, ..., I_T) = \sum_{j=1}^m h(a_{I_j}, b_{I_j})$ and is concave for each of the $2T$ variables, then there exist an consecutive partition $P^* = \{I_1^*, ..., I_T^*\}$ that minimizes $g(\cdot)$, and this partition could be found by standard dynamic programming in $\mathcal{O}(Tn^2)$ time.*

We refer readers to [1] for the details of finding an optimal consecutive partition in $\mathcal{O}(Tn^2)$ time. We apply the lemma to the XGBoost relaxation problem. We denote $G_j = \sum_{i\in I_j} g_i$ and $H_j = \sum_{i\in I_j} h_i$. It is easy to see that $\tilde{L}^{(t)}(q)$ is separable for each j and concave for each G_j, H_j. Besides, the convexity of the loss function $l(\cdot, \cdot)$ guarantees that $h_i \geq 0$ hence $H_j \geq 0$. We conclude that:

Theorem 1. *For any convex loss function, the learning objective equation (7) has a global optimal consecutive partitions which can be found in $\mathcal{O}(Tn^2)$ time*

We give an illustrate view why the (or one) global minimum is consecutive in terms of g_i/h_i. Rewrite Eq(4) as

$$\sum_{i=1}^{n}\frac{1}{2}h_i(f_t(\mathbf{x_i}) - g_i/h_i)^2 + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 + constant \tag{8}$$

Intuitively, we should allocate some i with similar g_i/h_i in a leaf of value w_j such that $f_t(\mathbf{x}_i) - g_i/h_i = w_j - g_i/h_i$ is near zero.

3 Refining the XGBoost

Although the partition given by theorem gives a global minimum to the loss function, we do not build any decision tree that given a new instance $\mathbf{x_{n+1}}$, which partition j should it be allocated such that $\hat{y}_{n+1} := w_j$. Indeed, it is possible that there is no decision tree whose leaves are the sets in the partition. We give a simple example below.

Example 1. *Suppose* $\mathbf{x}_1 = (-1, 0), \mathbf{x}_2 = (1, 0), \mathbf{x}_3 = (0, 0)$ *and the optimal partition is* $\mathcal{P}^* = \{\{1, 2\}, \{3\}\}$. *For any split in a decision tree* $\mathbf{q}$ *, it should be on either the first or second dimension. We show that such tree does not exist. In the first dimension,* $\mathbf{x}_3$ *lies in between* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *hence the split does not exist. In the second dimension, their values are the same such that the split does not exist either.*

Although the points might be linearly separable in hyperplane, it is costly to do so. Besides, it is not wise to convert a simple regression problem to an aggregation of multi-class classification.

Nevertheless, the optimal partition is informative. Since the decision tree trained by heuristic split method should be close to the optimal partition, we investigate the causation of the small difference of two partitions. We examine the position of the points in each partition. If there is a point whose prediction value w_j is disparate in each of the partition, then this point is the causation of partition's difference and highly likely to be an outlier that contaminate prediction function. By examining the points and eliminating the outliers, we can essentially reduce the level of over-fitting in each tree hence improve the overall performance of the tree ensembles. We present the algorithm below.

Algorithm 1: Refined XGBoost

Input : Training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, Initial prediction function $f_0(\cdot)$, size of tree ensemble K, Outlier threshold α

Output : XGBoost tree ensemble $\phi(\cdot)$

$\hat{y}_i^0 = f_0(\mathbf{x}_i)$;

for $k \leftarrow 1$ **to** K **do**

- $g_i^k, h_i^k \leftarrow \partial_{\hat{y}_i^{k-1}} l(y_i, \hat{y}_i^{(k-1)}), \partial^2_{\hat{y}^{k-1}} l(y_i, \hat{y}_i^{(k-1)})$;
- $\bar{W}^k = \{\bar{w}_1^k, ..., \bar{w}_n^k\} \leftarrow$ weights from heuristic XGBoost w.r.t $\mathbf{g}^k, \mathbf{h}^k$;
- $\tilde{W}^k = \{\tilde{w}_1^k, ..., \tilde{w}_n^k\} \leftarrow$ weights from consecutive optimizer w.r.t $\mathbf{g}^k, \mathbf{h}^k$;
- $d_i^k \leftarrow |\bar{w}_i^k - \tilde{w}_i^k|$;
- $\{\sigma^k(1), ..., \sigma^k(n)\} \leftarrow$ permutation by sorting $\{1, ..., n\}$ with ascending d_i^k;
- $I_{good}^k \leftarrow$ first $(1 - \alpha) \times 100\%$ proportion of elements in the above permutation;
- $f_k \leftarrow$ XGBoost heuristic prediction function trained on instances in I_{good}^k;
- $\hat{y}_i^k \leftarrow f_k(\mathbf{x}_i)$;

end

$\phi = \sum_{k=0}^K f_k$

It is known that the time complexity of XGBoost algorithm is $\mathcal{O}(KTd \log n)$ (Chen and Guestrin, 2016). Combining this with theorem 1, we have:

Theorem 2. *Algorithm 1 has* $\mathcal{O}(KT(d \log n + mn^2))$ *time complexity. Here* m *is the average number of sets in all partitions*

In practice, we tune the outlier threshold α by cross validation method. Intuitive, we should apply higher α for noisy data. There could be other tuning techniques, for example, performing heuristic XGBoost in the first in the first K_s tree while applying algorithm 1 for the rest of them. This algorithm is potentially applicable to outlier detection problem since the eliminated points are likely to be contaminated or doesn't come from the same nature as other points does.

4 Computaitonal Result

I have to stay honest. The result is not satisfying, otherwise this draft will become a conference paper in ICML or NIPS. I give the explanations below.

First, the benchmark is high. The XGBoost package `https://github.com/dmlc/xgboost` has been well-optimized and it is hard to do inverse-engineering to see what are the techniques to apply. My code is built on the toy model `https://github.com/eriklindernoren/ML-From-Scratch/tree/master/mlfromscratch/supervised_learning` that is far from fine tuning.

Second, I am a math undergraduate and my supervisor in this project in Professor Teo Chung-Piaw, a scholar in combinatorial optimization and operations research. He can evaluate my idea at the angle of optimization but don't know how CS literature will see this work. If you are interested with this idea, please contact me and we can possibly collaborate on this.

References

[1] Chakravarty, A. K., J. B. Orlin, and U. G. Rothblum. A partitioning problem with additive objective with an application to optimal inventory groupings for joint replenishment. In Operations Research. 30, no. 5 (1982): 1018-1022.

[2] Chakravarty, Amiya K., James B. Orlin, and Uriel G. Rothblum. Consecutive optimizers for a partitioning problem with applications to optimal inventory groupings for joint replenishment. *Operations Research.* 33, no. 4 (1985): 820-834.

[3] Chen, Tianqi, and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785-794. ACM, 2016.