A VARIANT OF XGBOOST WITH DYNAMIC SCALING FACTORS

Zhihan Liu
PB18051211

January 12, 2021

1 Introduction

XGBoost is one of the most powerful models in the Machine Learning and widely applied and achieves state-of-art performance on different real word task.

However, there still exist some problems with XGBoost when the data set is large and complex:

- High complexity of space, for many trees are needed to store.
- XGboost just add a new tree to narrow the gap between the present prediction and the real response rather than changing the former trees which make it hard to do the transferring work.
- Like deep neural network, the model interpretation is poor if we ensemble too many trees.
- In the ordinary tree, we cannot delete a *bad* tree but only can offset the effect by introducing a new tree.

And in this paper, we propose a new variant of XGBoost in order to solve the below problems.

2 Related Work

XGBoost The main idea of XGBoost is to convey gradient among different layers of the gradient tree so that our combined predicition can be closer to the true value. The prediction of a tree ensemble model with K addictive functions can be formulated in:

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

where $\mathcal{F} = \left\{f(\mathbf{x}) = w_{q(\mathbf{x})}\right\} \left(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T\right)$ is the space of regression trees T is the number of leaves in the tree. Each f_k corresponds to an independent tree structure q and leaf weights w.

And to train the model, we minimize the following regularized objective.

$$\begin{aligned} &\mathcal{L}(\phi) = \textstyle\sum_i l\left(\hat{y}_i, y_i\right) + \sum_k \Omega\left(f_k\right) \\ &\text{where } \Omega(f) = \gamma T + \tfrac{1}{2}\lambda\|w\|^2 \end{aligned} \tag{2}$$

And another important factor of the success of XGBoost is the use of regularization and the pruning of the tree. While XGboost prune every tree by backwarding, it also pursuit sparsity in the tree split by greedy algorithm.

Sparse Structure Selection for Deep Neural Networks To pursuit a sparse deep neural structure, Zehao Huang providing a fancy idea by introducing a scaling factor on the output of block or neuron and enhance the sparsity by exploiting the property of l_1 penalty and thresfold functions. Combine with the **Accelerated Proximal Gradient (APG)** , we can also take gradient of the scaling factor very efficiently and derive our desired sparse structure.

3 Methodology

3.1 Motivations

Our motivation is also to prune the model of XGBoost, but consider every tree as a block thus the whole XGBoost model consists of certain blocks. Then our goal is to make every block sparse, i.e. when we have a lot of blocks in our model, we just make some of them instead of all of them contribute to our whole model. We desire to control the sparsity by introducing a scaling factor to the output of each tree.

3.2 XGBoost with Sparse Structure Selection

For we scale each layer of the tree model with a responding scaling factor $\beta_k (0 \leq \beta_k \leq 1)$, then we can rewrite the prediction formula as:

$$\hat{y}_i = \phi\left(\mathbf{x}_i\right) = \sum_{k=1}^{K} \beta_k f_k\left(\mathbf{x}_i\right), \quad f_k \in \mathcal{F} \tag{3}$$

Except for β_k, every notes are the same with (1).

Also we have our new objective function to minimize:

$$\begin{gathered} \mathcal{L}(\phi) = \sum_{i=1}^{n} l\left(\hat{y}_i, y_i\right) + \mathcal{R}(\mathbf{W}) + \mathcal{R}_s(\boldsymbol{\beta}) + \gamma T \\ \text{where } \mathcal{R}(\mathbf{W}) = \tfrac{1}{2}\lambda \|w\|_2^2, \quad \mathcal{R}_s(\boldsymbol{\beta}) = \gamma_0 \sum_{i=1}^{K} ||\beta_k||_1 \\ \text{For simplicity, we denote } \Omega\left(f\right) := \mathcal{R}(\mathbf{W}) + \gamma T \end{gathered} \tag{4}$$

For we use additive manner, and denote $\hat{y}_i^{(t)}$ to be the prediction of the i -th instance at the t -th iteration, then we just need to add f_t to minimize the following objective function.

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t\left(\mathbf{x}_i\right)\right) + \Omega\left(f_t\right) + \mathcal{R}_s(\boldsymbol{\beta}) \tag{5}$$

Similar to the ordinary XGBoost, we use second-order approximation.

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} \left[l\left(y_i, \hat{y}^{(t-1)}\right) + g_i f_t\left(\mathbf{x}_i\right) + \frac{1}{2} h_i f_t^2\left(\mathbf{x}_i\right) \right] + \Omega\left(f_t\right) + \mathcal{R}_s(\boldsymbol{\beta}) \tag{6}$$

where $g_i = \partial_{\hat{y}(t-1)} l\left(y_i, \hat{y}^{(t-1)}\right)$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right)$ are first and second order gradient statistics on the loss function.

And if we define $I_j = \{i \mid q\left(\mathbf{x}_i\right) = j\}$ as the instance set of leaf j. We can rewrite (6) by expanding Ω, then follow the steps as the ordinary XGBoost, for a fixed structure $q(\mathbf{x})$, we can compute the optimal weight w_j^* of leaf j by

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

As for the splitting of tree we can use the same method with the ordinary XGBoost by greedy algorithm. We can define I_L and I_R are the instance sets of left and right nodes after the split, and derive the reduction loss which is used to evaluate the optional split.

$$\mathcal{L}_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma \tag{7}$$

It is worthy of explaining why our scaling factor are not appear explicitly on the last two formulas before, for the reason each tree are independent and just interacts with the prediction and its prediction loss thus the scaling factor effect w and a new tree splitting the by effecting the prediction loss generated by the previous trees.

And another important thing is to decide *when to delete a tree*. In fact, the scaling factor $beta_k$ has already gives us a optional answer. For the sparsity of $\boldsymbol{\beta}$, we can naturally delete the k^{-th} tree when $\beta_k = 0$, which help prune our XGboost model.

3.3 Adaptive Tuning Scaling Factor

To update $\boldsymbol{\beta}$, we can use Accelerated Proximal Gradient (APG) method to solve the following optimization problem:

$$\min_{\boldsymbol{\lambda}} \mathcal{G}(\boldsymbol{\beta}) + \mathcal{R}_s(\boldsymbol{\beta}) \tag{8}$$

$$\text{where } \mathcal{G}(\boldsymbol{\beta}) := \sum_{i=1}^{n} l\left(\hat{y}_i, y_i\right)$$

And we can calculate its gradient. And the following inductions are similar with the work by Zehao Huang except for some changes for our problem setting.

$$\nabla \mathcal{G}(\boldsymbol{\beta}) = \sum_{i=1}^{n} (f_1\left(\mathbf{x}_i\right), ..., f_K\left(\mathbf{x}_i\right))$$

Thus we have

$$\begin{aligned} \mathbf{d}_{(t)} &= \boldsymbol{\beta}_{(t-1)} + \frac{t-2}{t+1}\left(\boldsymbol{\beta}_{(t-1)} - \boldsymbol{\beta}_{(t-2)}\right) \\ \mathbf{z}_{(t)} &= \mathbf{d}_{(t)} - \eta_{(t)} \nabla \mathcal{G}\left(\mathbf{d}_{(t)}\right) \\ \boldsymbol{\beta}_{(t)} &= \operatorname{prox}_{\eta(t)} \mathcal{R}_s\left(\mathbf{z}_{(t)}\right) \end{aligned} \tag{9}$$

where $\mathbf{v}_{(t-1)} = \boldsymbol{\lambda}_{(t-1)} - \boldsymbol{\lambda}_{(t-2)}$, $\eta_{(t)}$ is the learning rate and $\mu_{(t-1)} = \frac{t-2}{t+1}$

Finally if we additionally define $\boldsymbol{\beta'}_{(t-1)} := \boldsymbol{\beta}_{(t-1)} + \mu_{(t-1)}\mathbf{v}_{(t-1)}$ and expand the proximal gradient operator $\operatorname{prox}_{\eta(t)}$, we can get a simpler updating form:

$$\begin{aligned} \mathbf{z}_{(t)} &= \boldsymbol{\beta}'_{(t-1)} - \eta_{(t)} \nabla \mathcal{G}\left(\boldsymbol{\beta}'_{(t-1)}\right) \\ \mathbf{v}_{(t)} &= \mathcal{S}_{\eta_{(t)}\gamma_0}\left(\mathbf{z}_{(t)}\right) - \boldsymbol{\beta}'_{(t-1)} + \mu_{(t-1)}\mathbf{v}_{(t-1)} \\ \boldsymbol{\beta}'_{(t)} &= \mathcal{S}_{\eta_{(t)}\gamma_0}\left(\mathbf{z}_{(t)}\right) + \mu_{(t)}\mathbf{v}_{(t)} \end{aligned} \tag{10}$$

where $\mathcal{S}_\gamma(z)$ is the soft-threshold operator which can compress the parameter to zero.

$$\mathcal{S}_\gamma(z) = \operatorname{sign}(z)(|z| - \gamma)_+ = \begin{cases} z - \gamma & \text{if } z > 0 \text{ and } \gamma < |z| \\ z + \gamma & \text{if } z < 0 \text{ and } \gamma < |z| \\ 0 & \text{if } \gamma \geq |z| \end{cases} \tag{11}$$

However, updating formula(10) can not be applied directly on our tree model for the dimension of $\boldsymbol{\lambda}$ is always increasing. One solution is to normalize $\boldsymbol{\beta}$ and use ($\frac{1}{K+1}$) padding trick when a k-th tree is being added.

$$\beta_i \leftarrow \frac{K}{K+1}\frac{\beta_i}{\sum_{j=1}^{K}\beta_j} \quad \forall i \leq K \quad \text{and} \quad \beta_{K+1} \leftarrow \frac{1}{K+1} \tag{12}$$

And to smooth the updating process and speed up the training, we can update the scaling factors every mini-batch rather than iteration.

In summary, we can propose a variant of ordinary XGBoost, which we call **XGBoost with Dynamic Scaling Factors**.

- We assign a scaling factor $\boldsymbol{\beta}$ to each tree, and we use the weighted sum of trees to serve as the model prediction and train our model every iteration similar with ordinary XGBoost
- We update $\boldsymbol{\beta}$ of each tree every iteration or mini-batch by APG according to (10).
- Every time a new tree is added, we reformulate $\boldsymbol{\beta}$ by (12).
- Whenever $\beta_k = 0$, we delete the k -th tree.

3.4 Analysis

Compared with the original XGBoost, our variant has the following properties:

- Reduce overfitting of XGBoost for we use less trees.
- Enable people to reuse the model from others or previous work, easier for transferring work. This because when we can download the existing XGBoost model form Internet with the similar problem.During our training the core feature will be remain while the other features not relevant to our present problem will be automatically deleted for the property of our structure.
- Less complexity of space. Owing to the sparsity of β, we will not need to store too many trees.
- Higher model interpretation.Simpler models mean higher model interpretation.
- But more complexity of time, for we need to take the gradient on the scaling factor $\boldsymbol{\beta}$. But this issue can be coped with by updating β every mini-batch instead of iteration and using the accelerating algorithm like APG.

4 Conclusions

In this paper, we first discuss the problems with ordinary XGBoost and combining another work by Zehao Huang, we propose a new variant of XGboost. We analyze our variant theoretically and will do the relevant experiment in the future.

References

[1] Chen, Tianqi and Guestrin, Carlos Data-Driven Sparse Structure Selection for Deep Neural Networks *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining DOI:10.1145/2939672.2939785*, 2016.

[2] Zehao Huang and Naiyan Wang Data-Driven Sparse Structure Selection for Deep Neural Networks *ECCV arXiv:1707.01213*, 2018.

[3] Johanna Sommer and Dimitrios Sarigiannis and Thomas Parnell Learning to Tune XGBoost with XGBoost *eprint 1909.07218*, 2019.