An Overview of Gradient Boost Machine and XGBoost

anhminhphanle@gmail.com

February 2018

- Jerome H Friedman, Greedy function approximation: A gradient boosting machine, 1999
- Tianqi Chen and Carlos Guestrin, XGBoost: A Scalable Tree Boosting System, 2016 https://arxiv.org/abs/1603.02754

1 Gradient Boosting Machine (GBM)

Given a training data $\{\mathbf{x_i}, y_i\}_{i=1}^N$, we need to find a model $F(\mathbf{x})$ that map $\mathbf{x}$ to y. We will measure the mapping effectiveness by choosing a loss function $L(y, F(\mathbf{x}))$ and then minimize it:

$$F^* = \underset{F}{\operatorname{argmin}} E_{y,\mathbf{x}} L(y, F(\mathbf{x})) \tag{1.1}$$

The model $F(\mathbf{x})$ is constructed in the form:

$$F(\mathbf{x}) = F_0(\mathbf{x}) + \sum_{m=1}^{M} \rho_m h_m(\mathbf{x}) \tag{1.2}$$

(1.2) is called boosting where:

- $F_0(\mathbf{x})$ is a initial guess, the first starting point of the model.
- $h_m(\mathbf{x})$ is called "weak learner" or "base learner".

We start the process by choosing $F_0(\mathbf{x})$, then measure the error between target y and prediction $F_0(\mathbf{x})$: $y - F_0(\mathbf{x})$. Next, we try to find a new function $h_0(\mathbf{x})$ so that:

$$F_0(\mathbf{x}) + \rho_0 h_0(\mathbf{x}) = y$$

The role of h_0 is to reduce the error of model $F(\mathbf{x})$. This step is proceeded recursively, we can add up another $h_1, \ldots, h_m$ until the best F^* is found. Each new h_m will try to correct errors made by previous h_{m-1} so that:

$$F_m(\mathbf{x}) + \rho_m h_m(\mathbf{x}) = y \tag{1.3}$$

One way to obtain that is to use "steepest descent" to find the right direction for h_m to follow:

$$g_m(\mathbf{x_i}) = -\left[\frac{\partial L\big(y_i, F(\mathbf{x_i})\big)}{\partial F(\mathbf{x_i})}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, i = 1, \ldots, N \tag{1.4}$$

The negative gradient $-g_m(\mathbf{x_i})$ is called the "linear search" along that direction in order for $\rho_m h_m$ to obtain "best greedy step" towards $F^*(\mathbf{x})$ in (1.2). We know that $\rho_m h_m$ has to output $g_m(\mathbf{x_i})$ allowing itself to "step" exactly in the direction of the negative gradient. Using a "rough" solution by only fitting h_m into the training data $\{\mathbf{x_i}, g_m(\mathbf{x_i})\}_{i=1}^{N}$:

$$h_m = \underset{h}{\operatorname{argmin}} \sum_{i=1}^{N} [-g_m(\mathbf{x_i}) - h(\mathbf{x_i})]^2 \tag{1.5}$$

The step size is optimized through linear search that satisfies:

$$\rho_m = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{N} L\big(y_i, F_{m-1}(\mathbf{x}) + \rho h_m(\mathbf{x})\big) \tag{1.6}$$

From (1.5) we can see that $h_m(\mathbf{x}) \sim g_m(\mathbf{x})$, using (1.4) we examine (1.2) under the form:

$$F(\mathbf{x}) \sim F_0(\mathbf{x}) - \rho_m \left[\frac{\partial L\big(y_i, F(\mathbf{x_i})\big)}{\partial F(\mathbf{x_i})}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \tag{1.7}$$

How can new h_m correct errors made by prior h_{m-1}? From (1.7) we can see what it does is similarly to Gradient Descent. At each iteration, h_m will continue to explore the path that its predecessor h_{m-1} has already followed and try to reach closer to the local minimum of loss function by following the negative gradient direction using the "steepest descent". The difference between Gradient Boosting and Gradient Descent in each step is: the Gradient Boosting adds **a new function to the model** that moves along the negative gradient direction. Whereas Gradient Descent **update its parameter** along the negative gradient direction. Both also try to reach a local minimum of the loss function.

1.1 Gradient Tree Boosting

We'll examine Gradient Tree Boosting's base learner $h(\mathbf{x})$, it is a decision tree with a special form:

$$h(\mathbf{x}) = \sum_{j=1}^{T} w_j \mathbf{1}(\mathbf{x} \in R_j) \tag{1.8}$$

- Indicator **1** has value of 1 if its argument is true, 0 otherwise.
- T is the number of its leaves.
- $\{R_j\}_1^T$ are disjoint regions correspond to the terminal nodes or leaves of the tree with it's predicted value $\{w_j\}_1^T$.

Algorithm 1 Friedman's Gradient Boost

```
Input: Training data $\{\mathbf{x_i}, y_i\}_{i=1}^N$, loss function $L\big(y, F(\mathbf{x})\big)$, number of iterations $M$,
base-learner model $h(\mathbf{x})$
Input: A model $F(\mathbf{x}) = \sum_{m=1}^M F_m(\mathbf{x})$
1: Initialize: Constant $F_0(\mathbf{x}) = \underset{F}{\text{argmin}} \sum_{i=1}^N L(y_i, \rho)$
2: for $m = 1$ to $M$ do
3:     Compute pseudo-residuals:
         $g_m(\mathbf{x_i}) = -\Big[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}\Big]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, i = 1, \dots, N$
4:     Fit $h(x)$ to pseudo-residuals using training set $\{\mathbf{x_i}, g_m(\mathbf{x_i})\}_{i=1}^N$
5:     Compute $\rho_m$ by solving linear-search problem:
         $\rho_m = \underset{\rho}{\text{argmin}} \sum_{i=1}^N L\big(y_i, F_{m-1}(\mathbf{x}) + \rho h_m(\mathbf{x})\big)$
6:     Update the model:
         $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h_m(\mathbf{x})$
7: end for
8: Output $F(\mathbf{x})$
```

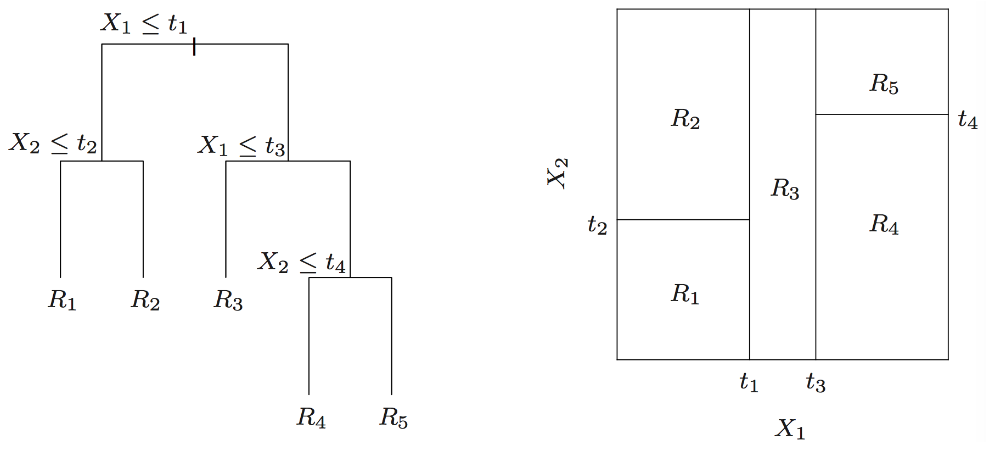

Figure 1: Tree and its Disjoint Regions

Let $g_i = g(\mathbf{x_i})$, the model's loss function is measured by the error of the base learner $h(\mathbf{x})$:

$$\begin{aligned} L\big(y, F(\mathbf{x})\big) &= \sum_{i=1}^N [-g_i - h(\mathbf{x_i})]^2 \\ &= \sum_{i=1}^N [g_i^2 + 2g_i h(\mathbf{x_i}) + h^2(\mathbf{x_i})] \\ &= \sum_{j=1}^T [\sum_{i \in I_j} g_i^2 + 2w_j \sum_{i \in I_j} g_i + n_j w_j^2] \end{aligned} \tag{1.9}$$

where n_j is the number of indices in region R_j. Let I_j is the set of indices that belongs to region R_j, meaning $\mathbf{x}_i \in R_j$ for $i \in I_j$. Let $G_j = \sum_{i \in I_j} g_i$, we further reduce to:

$$L\big(y, F(\mathbf{x})\big) = \sum_{j=1}^{T} [2G_j w_j + n_j w_j^2] + \text{constant} \tag{1.10}$$

With a fixed tree's structure, we compute the optimal weight $w*$ and replace it to (1.10):

$$\begin{aligned} w* &= -\frac{G_j}{n_j} \\ L\big(y, F(\mathbf{x})\big) &= -\sum_{j=1}^{T} \frac{G_j^2}{n_j} + \text{constant} \end{aligned} \tag{1.11}$$

From (1.11) we derive similarly to (2.13):

$$\begin{aligned} \mathcal{L}_{split} &= \frac{G_L^2}{n_L} + \frac{G_R^2}{n_R} - \frac{(G_L + G_R)^2}{n_L + n_R} \\ &= \frac{G_L^2}{n_L} + \frac{G_R^2}{n_R} - \frac{G^2}{n} \end{aligned} \tag{1.12}$$

2 XGBoost A Scalable Tree Boosting System

2.1 Tree Ensemble

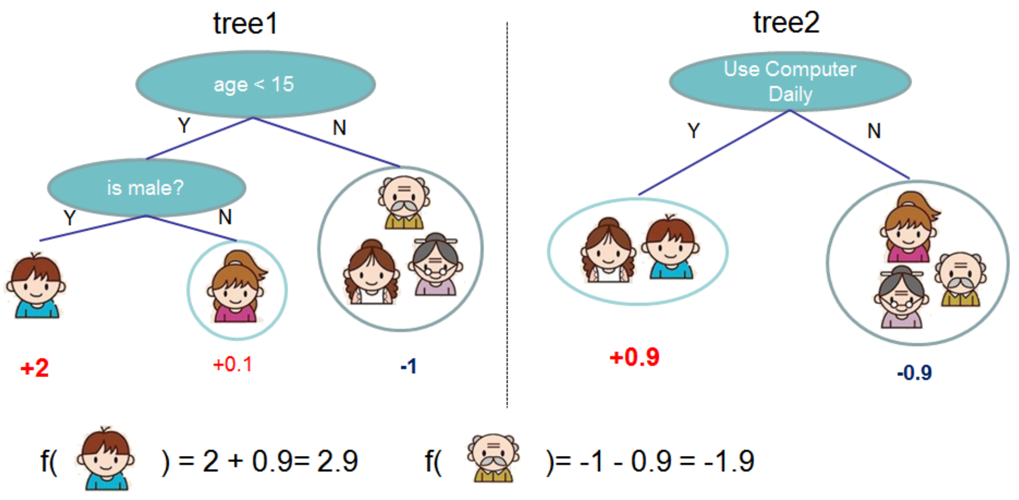

Figure 2: Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree.

Consider our model:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F} \tag{2.1}$$

K is the number of trees, $\mathcal{F}$ is the space of regression tree, the prediction scores of each individual tree f_k in space $\mathcal{F}$ are summed up to get the final prediction $\hat{y}_i$ from data x_i.

We need to optimize $\mathcal{L}$:

$$\mathcal{L}(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{2.2}$$

- l is the training loss function.
- Ω is regularization term.

2.2 Gradient Boost Tree

As definition of our model:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \tag{2.3}$$

Replace (2.3) to (2.2):

$$\mathcal{L}(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{k=1}^{K} \Omega(f_k) \tag{2.4}$$

Using Taylor expansion into MSE loss function:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t), \tag{2.5}$$

where:

$$\begin{aligned} g_i &= \frac{\partial l(y_i, F)}{\partial F} \\ h_i &= \frac{\partial^2 l(y_i, F)}{\partial^2 F}, F = \hat{y}_i^{(t-1)} \end{aligned} \tag{2.6}$$

Remove all constants:

$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{2.7}$$

2.3 Structure Score

Definition of tree $f(x)$:

$$f(x) = w_{q(x)}, q : \mathbb{R}^m \to T, w \in \mathbb{R}^T \tag{2.8}$$

Consider regularization function:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \tag{2.9}$$

- w is leaf score.
- q is a function assigning each data point to the corresponding leaf, forming the structure of each tree.
- T is the number of leaves in the tree.
- f_k corresponds to an independent tree structure q and leaf weights w.

Rewrite (2.7):

$$\begin{aligned}\tilde{\mathcal{L}}^{(t)} &\approx \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2}h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T\end{aligned} \tag{2.10}$$

Where $I_j = \{i|q(x_i) = j\}$ is the set of indices of data points assigned to the leaf j. Let $G_j = \sum_{i\in I_j} g_i$ and $H_j = \sum_{i\in I_j} h_i$. We reduce further (2.10):

$$\tilde{\mathcal{L}}^{(t)} = \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T \tag{2.11}$$

For fixed structure $q(x)$ we can compute the optimal weight $w*$ of leaf j. Then replace $w*$ to (2.11) to get corresponding value $\tilde{\mathcal{L}}^{(t)}$:

$$\begin{aligned}w_j^* &= -\frac{G_j}{H_j + \lambda} \\ \tilde{\mathcal{L}}^{(t)}(q) &= -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T\end{aligned} \tag{2.12}$$

2.4 Split candidate

$$\begin{aligned}\mathcal{L}_{split} &= \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \\ &= \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma \\ &= \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda}\right] - \gamma\end{aligned} \tag{2.13}$$

- $\mathcal{L}_{split}$ is the different score between the *tree's nodes* to the *left and right's nodes after split.*
- I is the set of indices of data points assigned to this node.
- I_L and I_R are the sets of indices of data points assigned to two new leaves, $I = I_L \cup I_R$.

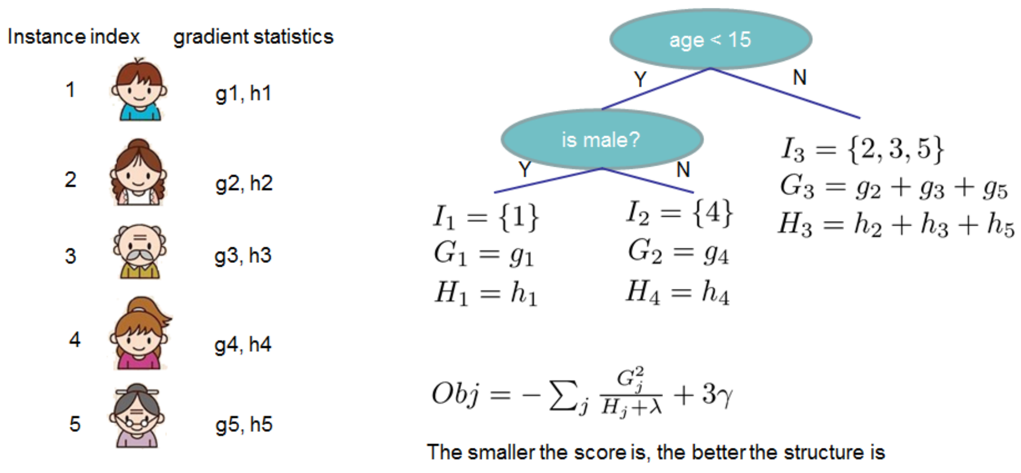

Figure 3: Structure Score Calculation. We only need to sum up the gradient and second order gradient statistics on each leaf, then apply the scoring formula to get the quality score.

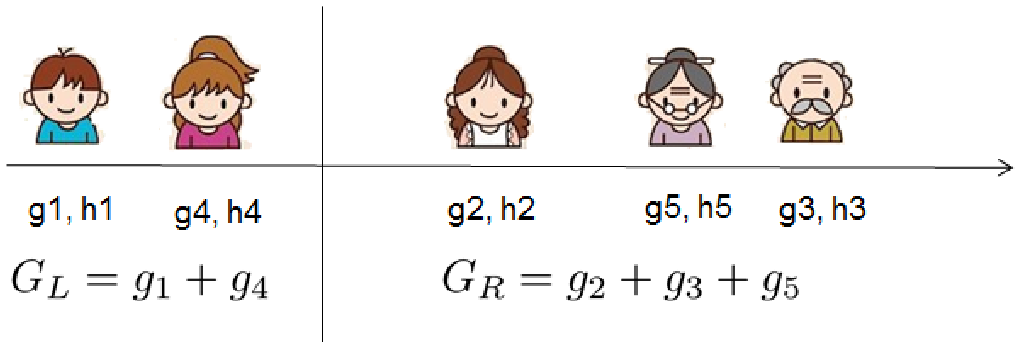

Figure 4: A left to right scan to find best split.

3 Differentiate between these two methods

- XGBoost can prevent overfitting better since its regularization is more advanced. This helps XGBoost to learn a finer tree structure compare to GBM.

$$\begin{cases} F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu\rho_m h_m(\mathbf{x}),\ 0 < \nu \le 1, & \text{GBM} \\ \Omega(f) = \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2, & \text{XGBoost} \end{cases}$$

- GBM is more generalized to be used in diverse applications. GBM's regularization is just a shrinkage value ν while XGBoost's regularization function $\Omega(f)$ requires to be differentiable for first and second derivatives.

- XGBoost is faster when calculating $\mathcal{L}_{split}$.

$$\mathcal{L}_{split} = \begin{cases} \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right] - \gamma, & \text{GBM} \\ \frac{G_L^2}{n_L} + \frac{G_R^2}{n_R} - \frac{G^2}{n}, & \text{XGBoost,} \end{cases}$$

  Also the weight of GBM is the average value of the gradients, that means GBM has to do linear search, scanning through all nodes in a region. While XGBoost only needs to scale the gradients by its hessians. But because XGBoost uses approximation to calculate the loss function, GBM's weight calculation is more accurate.

$$w_j = \begin{cases} -\frac{G_j}{n_j}, & \text{GBM} \\ -\frac{G_j}{H_j+\lambda}, & \text{XGBoost} \end{cases}$$