Boosting and Additive Trees

Yuting Zhang

Sun Yat-sen University, School of Mathematics

Boosting Method

Forward Stagewise Additive Modeling

L2Boosting

AdaBoost

LogitBoost

Boosting Trees

Numerical Optimization

- **Steepest Descent**
- **Gradient Boosting**
- **Regularization**
- **XGBoost**

Boosting Model

Different between bagging and boosting:

- Bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.

- Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees.

Boosting Model

Boosting is a way of fitting an additive expansion in a set of elementary "basis" functions.

Generally, basis function expansions take the form

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m),$$

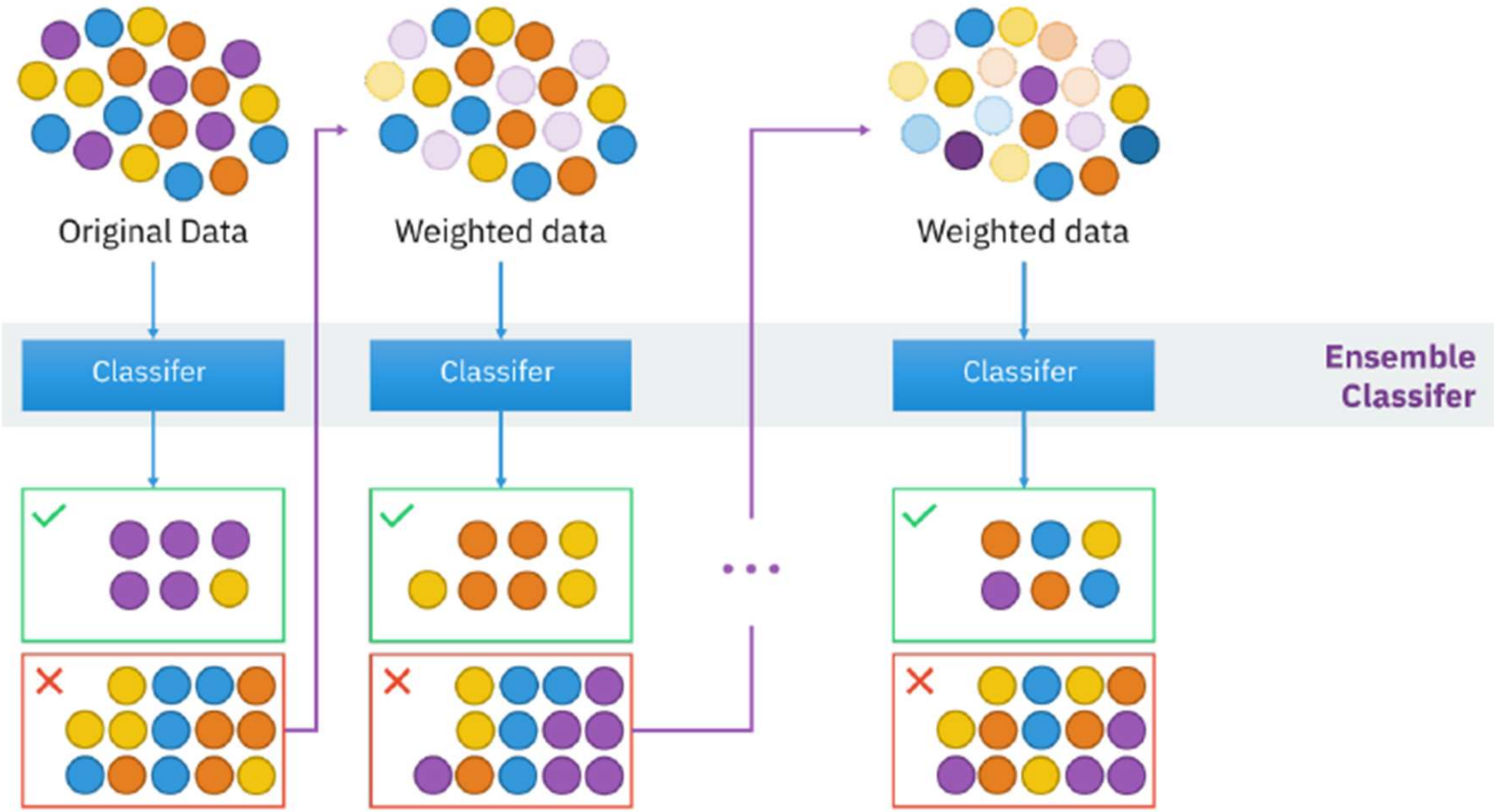
where $\beta_m, m = 1, \ldots, M$ are the expansion coefficients, and $b(x; \gamma) \in \mathbb{R}$ are usually simple functions of the multivariate argument x, characterized by a set of parameters γ.

Typically these models are fit by minimizing a loss function averaged over the training data,

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m))$$

中山大學
SUN YAT-SEN UNIVERSITY

Boosting Model



Forward Stagewise Additive Modeling

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute
   $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

L2Boosting

For squared-error loss

$$L(y, f(x)) = \frac{1}{2}(y - f(x))^2,$$

one has

$$L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)) = \frac{1}{2}(y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma))^2$$

$$= \frac{1}{2}(r_{im} - \beta b(x_i;\gamma))^2$$

AdaBoost

AdaBoost

Consider a two-class problem, with the output variable coded as $Y \in \{-1, 1\}$. Given a vector of predictor variables X, a classifier $G(X)$ produces a prediction taking one of the two values $\{-1, 1\}$. The error rate on the training sample is

$$\overline{\text{err}} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i)),$$

and the expected error rate on future predictions is $E_{XY} I(Y \neq G(X))$.

The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers $G_m(x), m = 1, 2, \dots, M$.

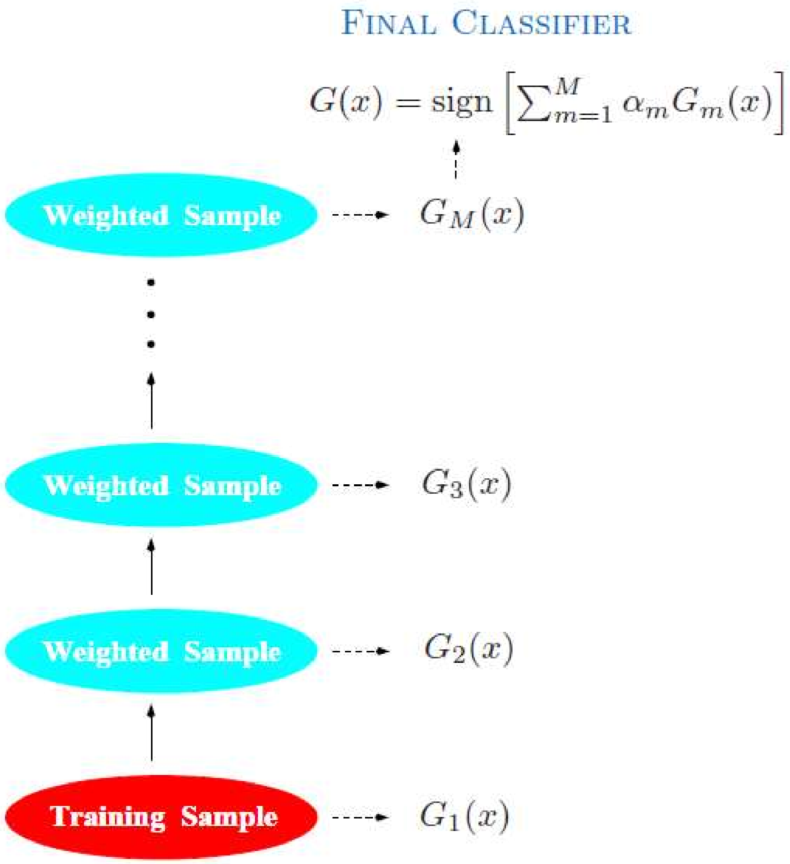
AdaBoost

AdaBoost

The predictions from all of them are then combined through a weighted majority vote to produce the final prediction：

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

Here $\alpha_1, \alpha_2, \ldots, \alpha_M$ are computed by the boosting algorithm, and weight the contribution of each respective $G_m(x)$.

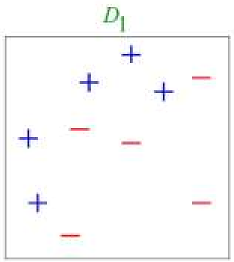


AdaBoost

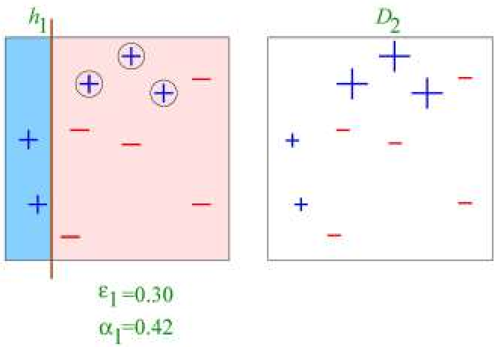
Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

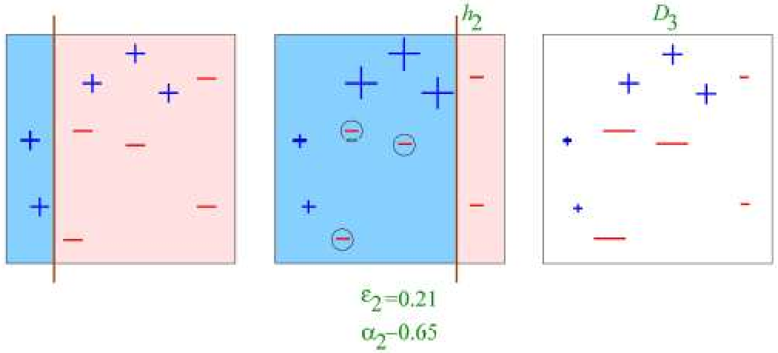
AdaBoost



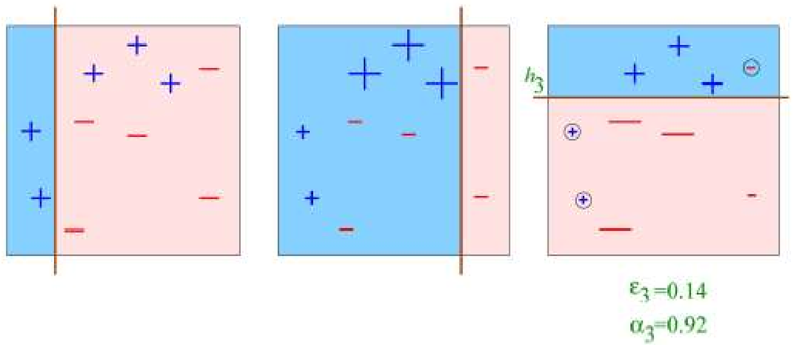
(a) 原始数据，蓝色+与红色–分别表示两类样本点



(b) 弱分类器1——h_1，针对分类正确的样本我们加大其权重，错分类样本减小权重

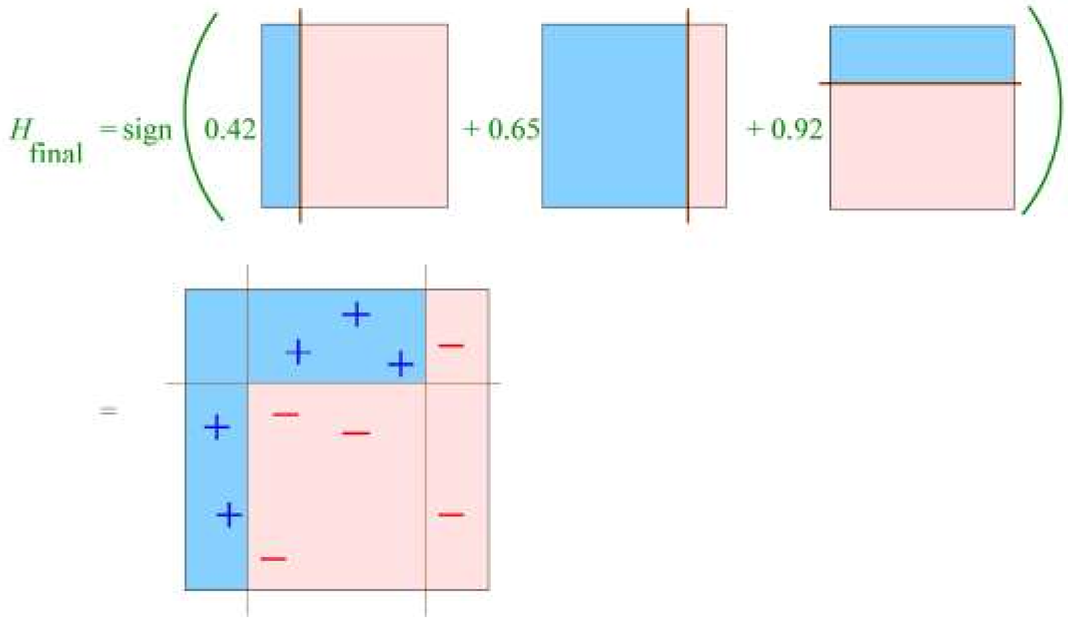


(c) 弱分类器2——h_2



(d) 弱分类器3——h_3

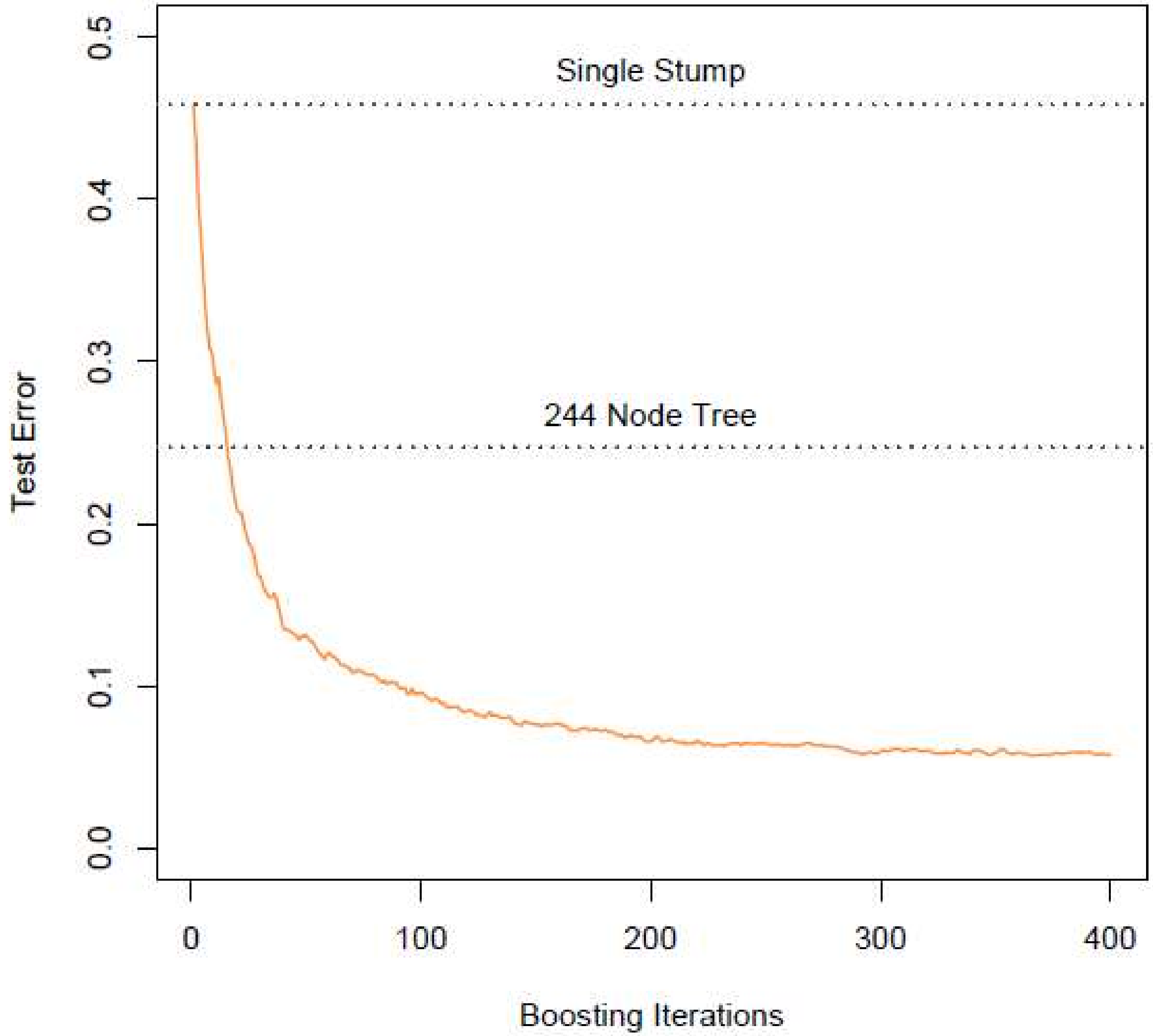
AdaBoost



(e) 总体Adaboost模型

SUN YAT-SEN UNIVERSITY

AdaBoost



SUN YAT-SEN UNIVERSITY 中山大學

Exponential Loss and AdaBoost

Exponential loss

$$L(y, f(x)) = \exp(-yf(x))$$

Using the exponential loss function, one must solve

$$(\beta_m, G_m) = \underset{\beta, G}{\operatorname{argmin}} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \beta G(x_i))]$$

$$\Downarrow$$

$$(\beta_m, G_m) = \underset{\beta, G}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta y_i G(x_i))$$

With $w_i^{(m)} = \exp(-y_i f_{m-1}(x))$.

Exponential Loss and AdaBoost

$$(\beta_m, G_m) = \underset{\beta, G}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta y_i G(x_i))$$

For any value of $\beta > 0$, the solution for $G_m(x)$

$$G_m = \underset{G}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))$$

Exponential Loss and AdaBoost

$$(\beta_m, G_m) = \underset{\beta, G}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta y_i G(x_i))$$

Plugging this Gm into the formula and solving for β one obtains

$$\beta_m = \frac{1}{2} \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m},$$

Where err_m is the minimized weighted error rate

$$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i^{(m)}}.$$

Exponential Loss and AdaBoost

The approximation is then updated

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x)$$

which causes the weights for the next iteration to be

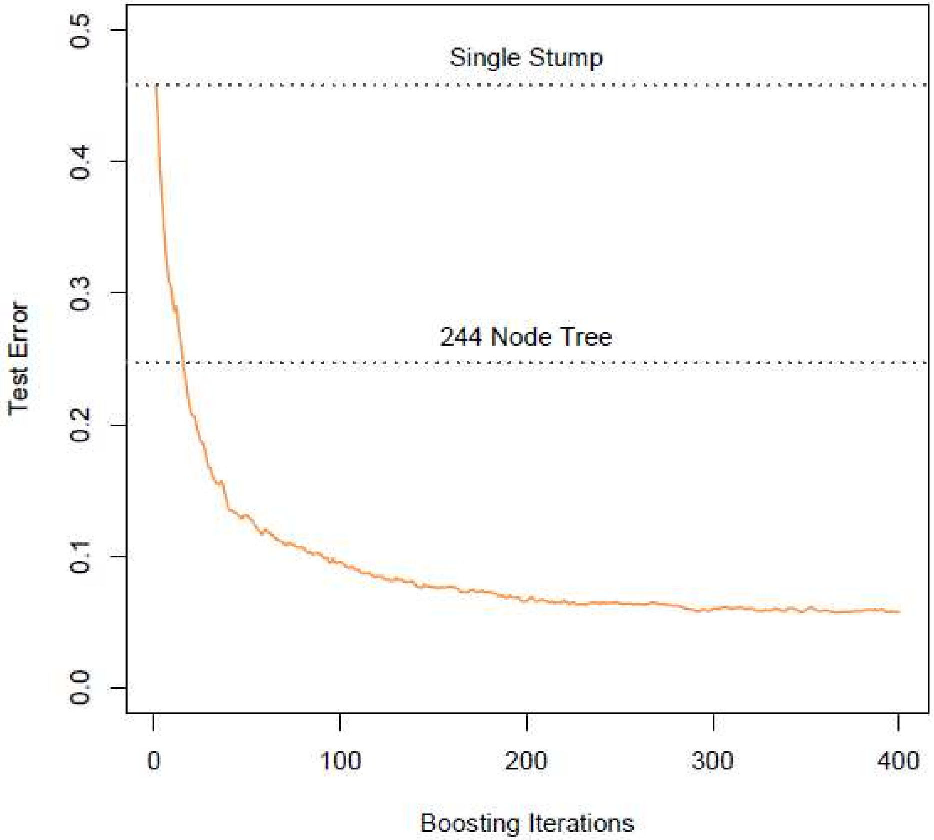
$$w_i^{(m+1)} = w_i^{(m)} \cdot e^{-\beta_m y_i G_m(x_i)}$$

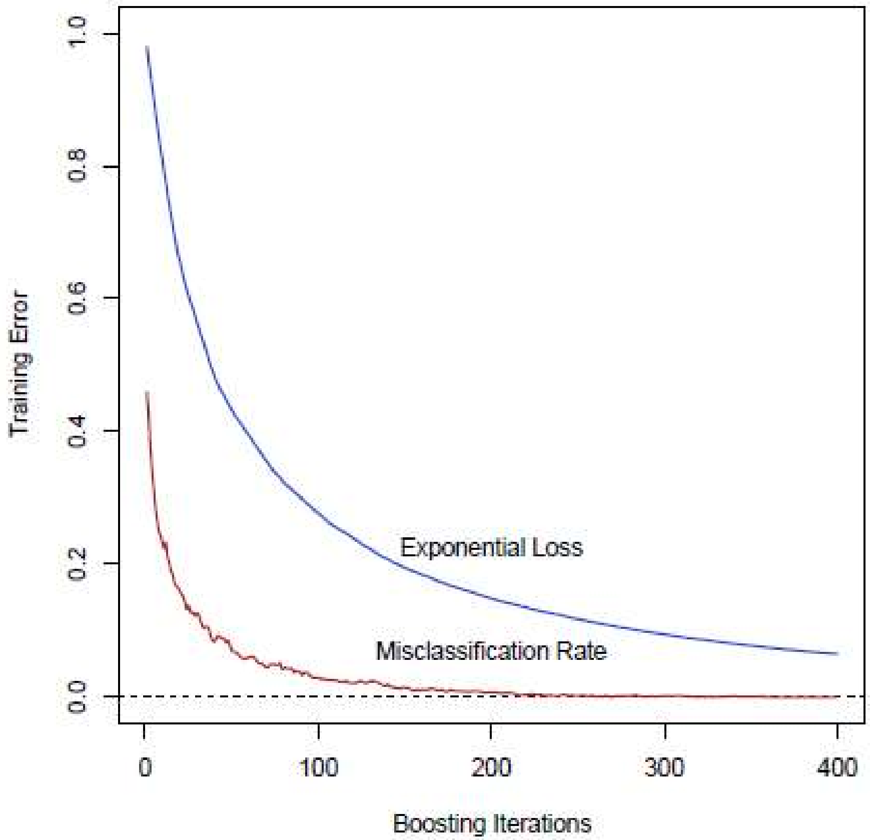
$$\Downarrow$$

$$w_i^{(m+1)} = w_i^{(m)} \cdot e^{\alpha_m I(y_i \neq G_m(x_i))} \cdot e^{-\beta_m},$$

where $\alpha_m = 2\beta_m$.

Exponential Loss and AdaBoost





Why Exponential Loss?

The additive expansion produced by AdaBoost is estimating one-half the log-odds of $P(y=1|x)$,

$$f^*(x) = \underset{f(x)}{\operatorname{argmin}} E_{Y|x}(e^{-Yf(x)}) = \frac{1}{2}\log\frac{P(y=1|x)}{P(y=-1|x)},$$

$$\Leftrightarrow P(y=1|x) = \frac{1}{1+e^{-2f^*(x)}}.$$

This justifies using its sign as the classification rule in $G(x) = \operatorname{sign}(\sum_{m=1}^{M} \alpha_m G_m(x))$.

LogitBoost

Binomial negative log-likelihood (deviance, or cross-entropy)

$$p(x) = P(y = 1|x) = \frac{e^{f(x)}}{e^{-f(x)} + e^{f(x)}} = \frac{1}{1 + e^{-2f(x)}}$$

$$l(p, y) = -(y \log p + (1 - y)\log(1 - p))$$

$$\Leftrightarrow l(y, f(x)) = -\log(1 + e^{-2yf(x)})$$

LogitBoost

Algorithm 16.3: LogitBoost, for binary classification with log-loss

$w_i = 1/N$, $\pi_i = 1/2$;
for $m = 1 : M$ **do**
  Compute the working response $z_i = \frac{y_i^* - \pi_i}{\pi_i(1-\pi_i)}$;
  Compute the weights $w_i = \pi_i(1 - \pi_i)$;
  $\phi_m = \text{argmin}_\phi \sum_{i=1}^N w_i(z_i - \phi(\mathbf{x}_i))^2$;
  Update $f(\mathbf{x}) \leftarrow f(\mathbf{x}) + \frac{1}{2}\phi_m(\mathbf{x})$;
  Compute $\pi_i = 1/(1 + \exp(-2f(\mathbf{x}_i)))$;
Return $f(\mathbf{x}) = \text{sgn}\left[\sum_{m=1}^M \phi_m(\mathbf{x})\right]$;

Boosting Tree

Regression and classification trees partition the space of all joint predictor variable values into disjoint regions $R_j, j = 1, 2, \ldots, J$, as represented by the terminal nodes of the tree. A constant γ_j is assigned to each such region and the predictive rule is

$$x \in R_j \Rightarrow f(x) = \gamma_j.$$

Thus a tree can be formally expressed as

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j),$$

with parameters $\Theta = \{R_j, \gamma_j\}_1^J$. The parameters are found by minimizing the empirical risk

$$\hat{\Theta} = \operatorname*{argmin}_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j).$$

Boosting Tree

Finding $\boldsymbol{\gamma_j}$ given $\boldsymbol{R_j}$:
Given the R_j, estimating the $\boldsymbol{\gamma_j}$ is typically trivial, and often $\boldsymbol{\hat{\gamma}_j} = \bar{y}_j$, the mean of the y_i falling in region R_j . For misclassification loss, $\hat{\gamma}_j$ is the modal class of the observations falling in region R_j.

Finding $\boldsymbol{R_j}$:
This is the difficult part, for which approximate solutions are found. Note also that finding the R_j entails estimating the γ_j as well. A typical strategy is to use a greedy, top-down recursive partitioning algorithm to find the R_j .

A strategy for classification trees.
The Gini index replaced misclassification loss in the growing of the tree (identifying the R_j).

Boosting Tree

The boosted tree model is a sum of such trees,

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$$

At each step in the forward stagewise procedure one must solve

$$\widehat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

Given the regions R_{jm}, then

$$\hat{\gamma}_{jm} = \underset{\gamma_{jm}}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

Boosting Tree

In particular, if the trees $T(x;\Theta_m)$ are restricted to be scaled classification trees, then the solution of $\widehat{\Theta}_m$ is the tree that minimizes the weighted error rate

$$\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq T(x_i;\Theta_m)),$$

Where $w_i^{(m)} = e^{-y_i f_{m-1}(x_i)}$. By a scaled classification tree, we mean $\beta_m T(x;\Theta_m)$, with the restriction that $\gamma_{jm} \in \{-1, 1\}$.

Without this restriction, $\widehat{\Theta}_m$ still simplifies for exponential loss to a weighted exponential criterion for the new tree:

$$\widehat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \exp[-y_i T(x_i;\Theta_m)].$$

Boosting Tree

Given the regions R_{jm}, we can show that $\hat{\gamma}_{jm}$ is the weighted log-odds in each corresponding region

$$\hat{\gamma}_{jm} = \frac{1}{2}\log\frac{\sum_{x_i \in R_{jm}} w_i^{(m)} I(y_i = 1)}{\sum_{x_i \in R_{jm}} w_i^{(m)} I(y_i = -1)}$$

Numerical Optimization

$$\hat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

The loss in using $f(x)$ to predict y on the training data is

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$$

It can be viewed as a numerical optimization

$$\hat{\mathbf{f}} = \underset{\mathbf{f}}{\operatorname{argmin}}\, L(\mathbf{f})$$

where $\mathbf{f} \in \mathbb{R}^N$ are the values of the approximating function $f(x_i)$ at each of the N data points x_i :

$$\mathbf{f} = \{f(x_1), f(x_2), \ldots, f(x_N)\}^T$$

Numerical Optimization

Numerical optimization:

$$\mathbf{f} = \sum_{m=1}^{M} \mathbf{h}_m, \qquad \mathbf{h}_m \in \mathbb{R}^N$$

where $\mathbf{f}_0 = \mathbf{h}_0$ is an initial guess, and each successive $\mathbf{f}_m$ is induced based on the current parameter vector $\mathbf{f}_{m-1}$.

Numerical optimization methods differ in their prescriptions for computing each increment vector $\mathbf{h}_m$.

Steepest Descent

Steepest descent chooses $\mathbf{h}_m = -\rho_m \mathbf{g}_m$ where ρ_m is a scalar and $\mathbf{g}_m \in \mathbb{R}^N$ is the gradient of $L(\mathbf{f})$ evaluated at $\mathbf{f} = \mathbf{f}_{m-1}$. The components of the gradient $\mathbf{g}_m$ are

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$$

The step length ρ_m is the solution to

$$\rho_m = \underset{\rho}{\operatorname{argmin}}\, L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m)$$

The current solution is then updated

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho \mathbf{g}_m$$

Gradient Boosting

Tree components:

$$\mathbf{t_m} = \{T(x_1; \Theta_m), \dots, T(x_N; ; \Theta_m)\}$$

Induce a tree $T(x; \Theta_m)$ at the mth iteration whose predictions tm are as close as possible to the negative gradient. Using squared error to measure closeness:

$$\widehat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^{N} (-g_{im} - T(x_i; \Theta))^2$$

Gradient Boosting

TABLE 10.2. *Gradients for commonly used loss functions.*

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

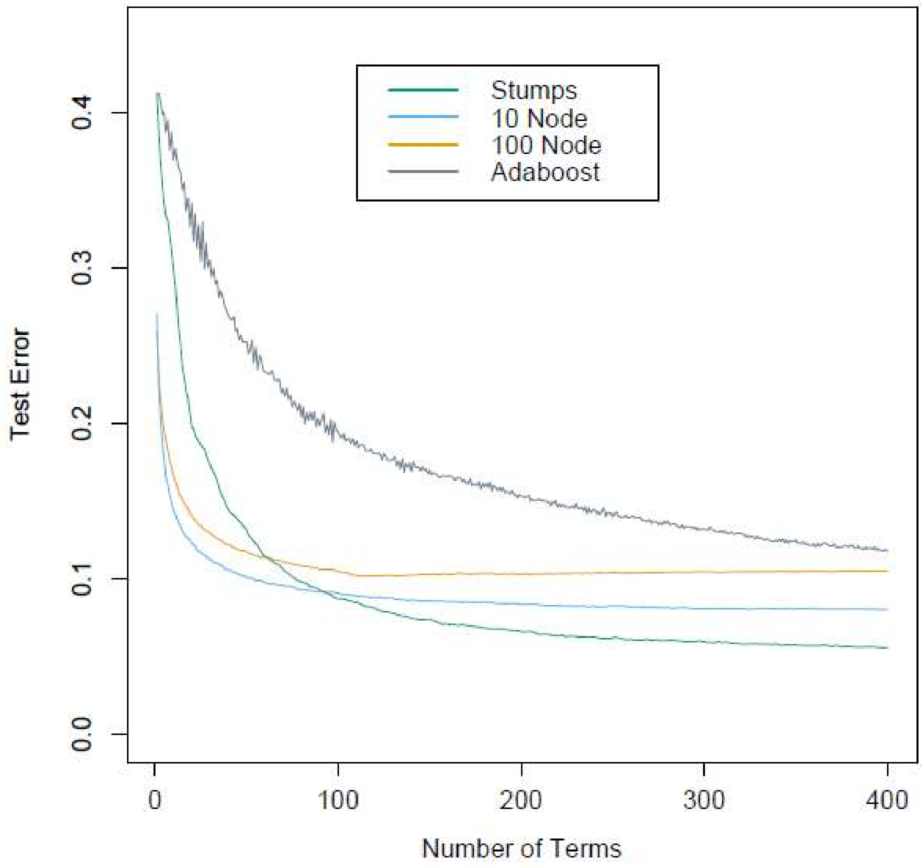
Implementations of Gradient Boosting

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

Right-Sized Trees for Boosting

Restrict all trees to be the same size, $J_m = J, \forall m$.



Regularization

The number of boosting iterations *M*

There is an optimal number M^* minimizing future risk that is application dependent. The value of M that minimizes this risk is taken to be an estimate of M^*.

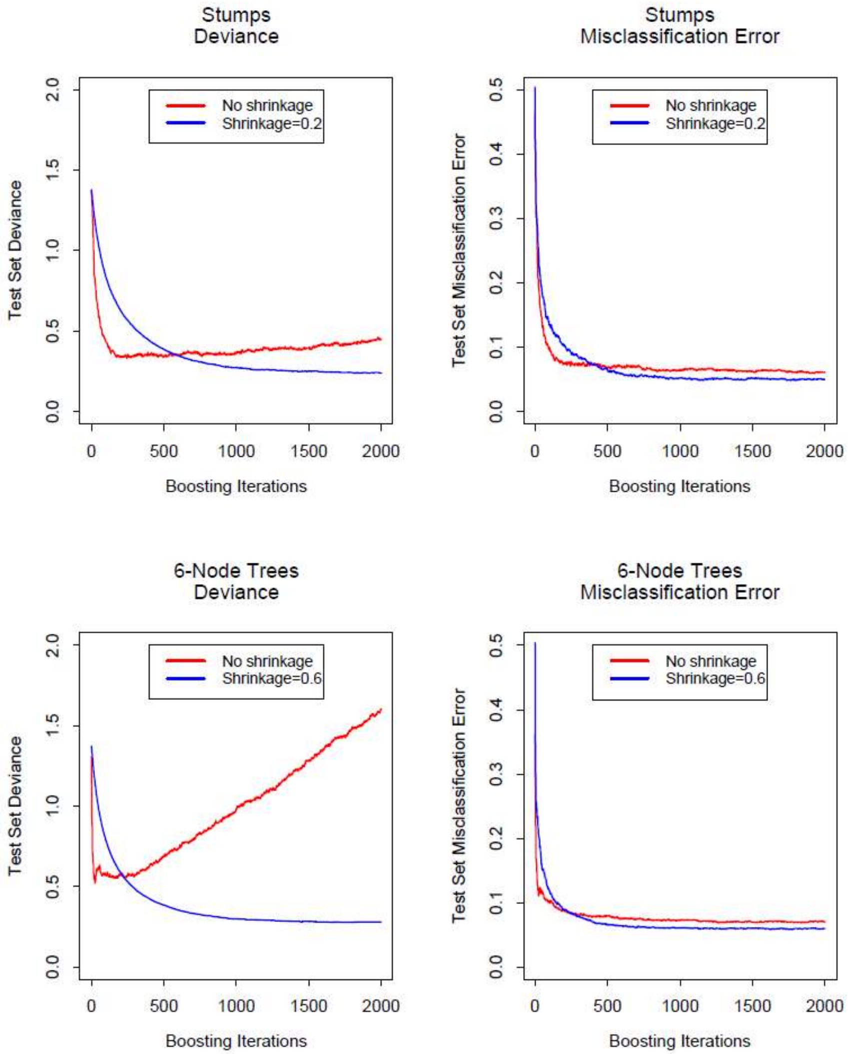
Shrinkage

Scale the contribution of each tree by a factor $0 < \nu < 1$ when it is added to the current approximation

$$f_m(x) = f_{m-1}(x) + \nu \cdot \sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm})$$

中山大學
SUN YAT-SEN UNIVERSITY

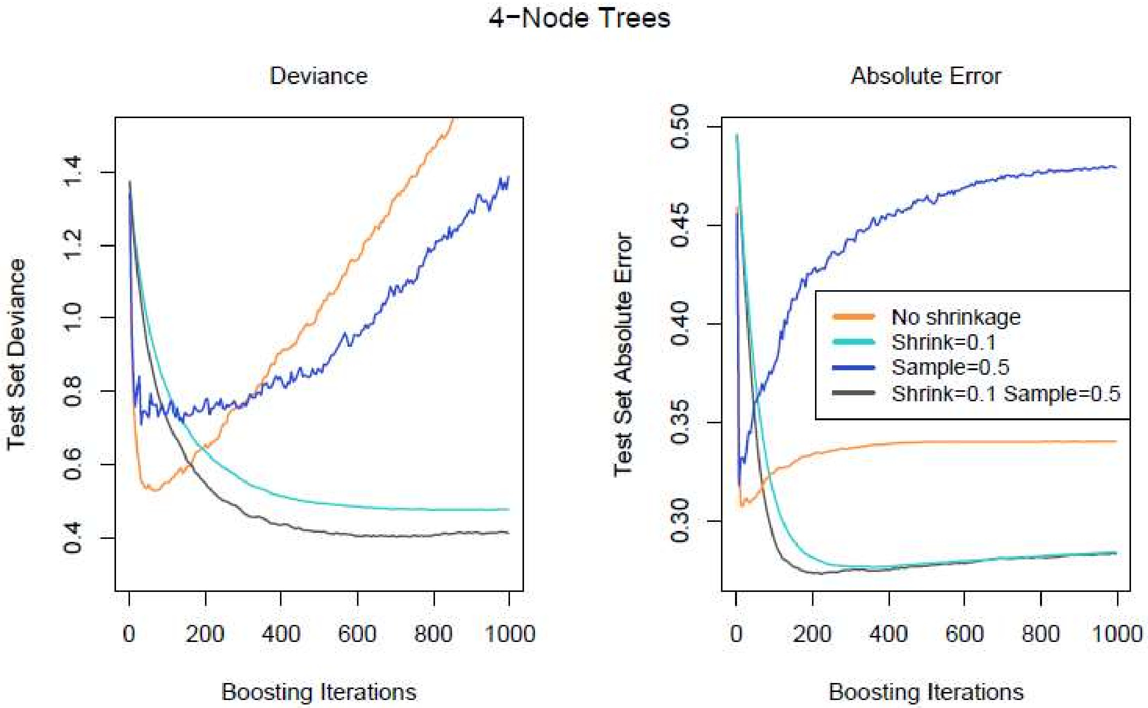
Regularization



Regularization

Subsampling

With stochastic gradient boosting, at each iteration we sample a fraction η of the training observations (without replacement), and grow the next tree using that subsample. A typical value for η can be $\frac{1}{2}$.



XGBoost

Objective

$$Obj = \sum_{i=1}^{N} \underbrace{L(y_i, F(x_i))}_{\text{Training loss}} + \sum_{m=1}^{M} \underbrace{\Omega(h_m)}_{\text{Complexity of the trees}}$$

where

$$\Omega(h) = \lambda_J J + \frac{1}{2}\lambda_\omega \|\omega\|^2$$

XGBoost

$$Obj^{(m)} = \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + h_m(x_i)) + \Omega(h_m)$$

Taylor expansion

$$Obj^{(m)} \approx \sum_{i=1}^{N} \left[L(y_i, F_{m-1}(x_i)) + f_{i,m-1} h_m(x_i) + \frac{1}{2} g_{i,m-1} h_m^2(x_i) \right] + \Omega(h_m)$$

where

$$f_{i,m-1} = \frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}, \qquad g_{i,m-1} = \frac{\partial^2 L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}^2(x_i)}$$

For squared loss,

$$f_{i,m-1} = 2(F_{m-1}(x_i) - y_i), \qquad g_{i,m-1} = 2$$

XGBoost

$$\widetilde{Obj}^{(m)} = \sum_{i=1}^{N}\left[f_{i,m-1}h_m(x_i) + \frac{1}{2}g_{i,m-1}h_m^2(x_i)\right] + \Omega(h_m)$$

$$= \sum_{i=1}^{N}\left[f_{i,m-1}h_m(x_i) + \frac{1}{2}g_{i,m-1}h_m^2(x_i)\right] + \lambda_J J + \frac{1}{2}\lambda_\omega\|\omega\|^2$$

$$= \sum_{j=1}^{J}\left[\left(\sum_{i\in I_j} f_{i,m-1}\right)\omega_j + \frac{1}{2}\left(\sum_{i\in I_j} g_{i,m-1} + \lambda_\omega\right)\omega_j^2\right] + \lambda_J J$$

$$\Longrightarrow \omega_j^* = -\frac{\sum_{i\in I_j} f_{i,m-1}}{\sum_{i\in I_j} g_{i,m-1} + \lambda_\omega}$$

XGBoost

$$\widetilde{Obj}^{(m)} = -\frac{1}{2}\sum_{i=1}^{N}\frac{\left(\sum_{i\in I_j} f_{i,m-1}\right)^2}{\sum_{i\in I_j} g_{i,m-1} + \lambda_\omega} + \lambda_J J$$

Let $I_L = \{\text{leaf of left subtree}\}$, $I_R = \{\text{leaf of right subtree}\}$, $I = I_L \cup I_R$.

Define

$$\text{Gain} = \frac{1}{2}\sum_{i=1}^{N}\left[\frac{\left(\sum_{i\in I_L} f_i\right)^2}{\sum_{i\in I_L} g_i + \lambda_\omega} + \frac{\left(\sum_{i\in I_R} f_i\right)^2}{\sum_{i\in I_R} g_i + \lambda_\omega} + \frac{\left(\sum_{i\in I} f_i\right)^2}{\sum_{i\in I} g_i + \lambda_\omega}\right] - \lambda_J$$

XGBoost

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

XGBoost

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.

Thanks !