10. Boosting and Additive Trees

오영민

Introduction

- **AdaBoost. M1.**
- **Exponential Loss and AdaBoost**
- **Loss Functions and Robustness**
- **Boosting Trees (Gradient Boosting Trees)**
- **Interpretation**

AdaBoost.M1.

- The motivation for boosting was a procedure that combines the outputs of **many "weak" classifiers** to produce a powerful "**committee**."

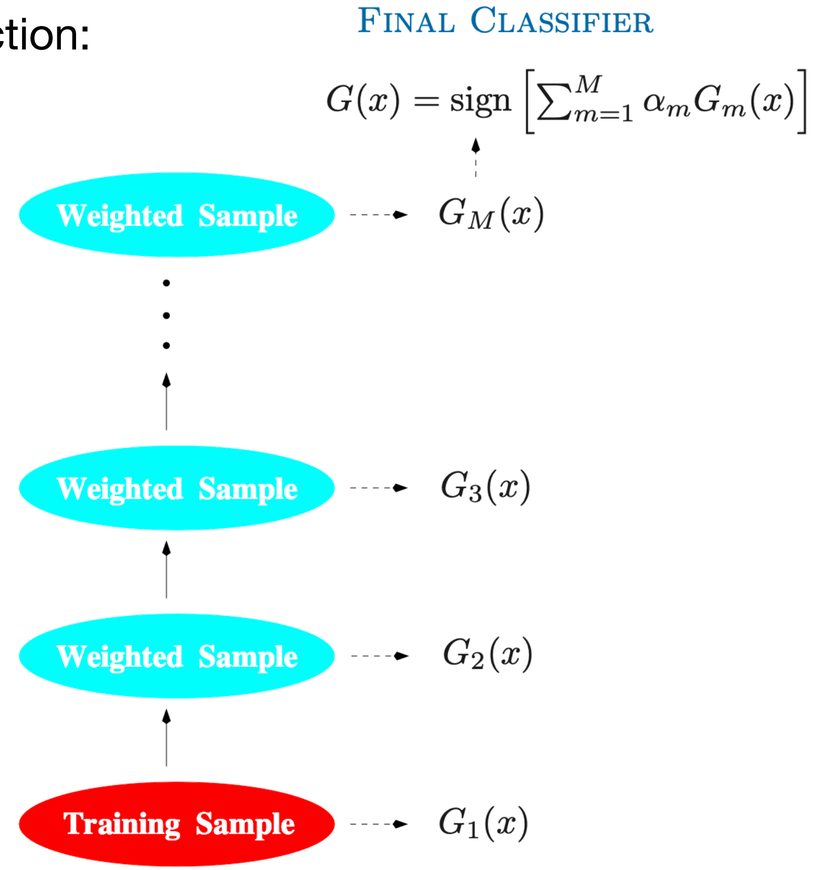
- Consider binary classification with response $Y \in \{-1,1\}$.

- Producing a sequence of weak classifiers $G_m(x)$, $m = 1,...,M$, then the final prediction:

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

where $\alpha_1, \ldots, \alpha_M$ are weights the contribution of each respective $G_m(x)$

- At each step, applying weights $w_1, \ldots, w_M$ to each $\{x_i, y_i\}_{i=1}^{N}$



AdaBoost.M1.

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: It looks similar to Forward stage-wise procedure

   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.

   (b) Compute

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

   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$. Log odds on $1 - err_m$

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

AdaBoost.M1.

- Boosting is a way of fitting an **additive expansion** in a set of elementary “basis” functions.

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$ where β_m are the expansion coefficients, $b_m = b(\cdot\,; \gamma_m)$ is basis functions

Single-hidden-layer neural network: $b(x; \gamma) = \sigma(\gamma_0 + \gamma_1^T x)$

MARS: b_m is truncated power spline basis functions

Trees: $b_m(x) = I(x \in R_m)$

AdaBoost.M1.: $b_m(x) = G_m(x)$

- **Optimization**:

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x; \gamma_m)\right)$$ which is hard to compute ⟶ **Forward Stage-wise Additive Modeling**

Exponential Loss and AdaBoost

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 $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

AdaBoost.M1. minimizes the exponential-loss criterion via a forward stagewise additive modeling.

For exponential loss $L(y, f(x)) = \exp(-yf(x))$, step 2-(a) can be written as

$$(\beta_m, G_m) = \underset{\beta,G}{argmin} \sum_{i=1}^{N} \exp\left(-y_i G_{m-1}(x_i)\right) \exp\left(-y_i \beta G(x_i)\right) = \underset{\beta,G}{argmin} \sum_{i=1}^{N} w_i^{(m)} \left[I(y_i = G(x_i))e^{-\beta} + I(y_i \neq G(x_i))e^{\beta}\right]$$

$$= \underset{\beta,G}{argmin} \left[(e^{\beta} - e^{-\beta}) \cdot \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)) + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)}\right]$$

Exponential Loss and AdaBoost

- For m-th step, (β_m, G_m) solves

$$\underset{\beta,G}{argmin}\left[(e^{\beta}-e^{-\beta})\cdot\sum_{i=1}^{N}w_i^{(m)}I(y_i\neq G(x_i))+e^{-\beta}\sum_{i=1}^{N}w_i^{(m)}\right] \text{ where } w_i^{(m)}=\exp\left(-y_iG_{m-1}(x_i)\right)$$

- First, for any $\beta>0$, $G_m=\underset{\beta,G}{argmin}\sum_{i=1}^{N}w_i^{(m)}I(y_i\neq G(x_i))$

- Plugging G_m into criterion, and solving for β, $\underset{\beta}{argmin}\sum_{i=1}^{N}w_i^{(m)}\left[I(y_i=G_m(x_i))e^{-\beta}+I(y_i\neq G_m(x_i))e^{\beta}\right]$

$$\left[\frac{\partial}{\partial\beta}\sum_{i=1}^{N}w_i^{(m)}\left[I(y_i=G_m(x_i))e^{-\beta}+I(y_i\neq G_m(x_i))e^{\beta}\right]\right]_{\beta=\beta_m}=0,\ \beta_m=\frac{1}{2}\log\frac{1-err_m}{err_m}=\frac{1}{2}\alpha_m \text{ in AdaBoost.M1.}$$

- Update $f_m(x)=f_{m-1}(x)+\beta_mG_m(x)$ and $w_i^{(m+1)}=w_i^{(m)}\cdot\exp(-y_i\beta_mG_m(x))=w_i^{(m)}\cdot\exp(-\beta_m(2I(y_i\neq G_m(x_i)-1))$

$$=w_i^{(m)}\cdot\exp(-\alpha_mI(y_i\neq G_m(x_i))\cdot\exp(-\beta_m)$$

Exponential Loss and AdaBoost

- **Population minimizer for the exponential loss:**

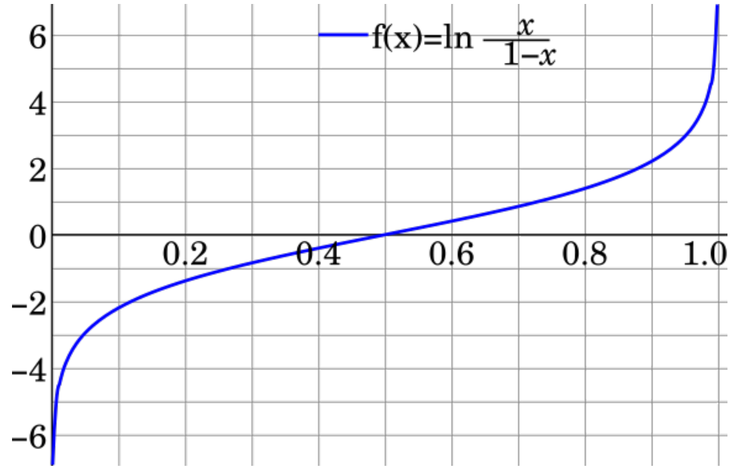
$$f^*(x) = \underset{f(x)}{argmin}\, \mathbb{E}_{Y|x} \exp(-Yf(x)) = \underset{f(x)}{argmin} \left[\exp(-f(x))Pr(Y=1\,|\,X=x) + \exp(f(x))Pr(Y=-1\,|\,X=x)\right]$$

$$= \frac{1}{2}\frac{\log Pr(Y=1\,|\,X=x)}{\log Pr(Y=-1\,|\,X=x)}$$

This justifying using its sign as classification rule in AdaBoost.M1.

AdaBoost.M1. can be interpreted as a stagewise estimation for fitting an additive logistic model.

or $Pr(Y=1\,|\,X=x) = \dfrac{1}{1+\exp(-2f^*(x))}$



Logit function

- **Population minimizer for the binomial deviance:**

Let $p(x) = Pr(Y=1\,|\,X=x) = \dfrac{1}{1+\exp(-2f(x))}$

Define $Y' = (Y+1)/2 \in \{0,1\}$

Then $-l(Y,p(x)) = -\,Y'\log p(x) - (1-Y')\log(1-p(x)) = \log(1+\exp(-2Yf(x)))$

And we know that $\underset{f(x)}{argmin}\, \mathbb{E}_{Y|x} \log(1+\exp(-2Yf(x))) = \underset{f(x)}{argmin}\, \mathbb{E}_{Y|x}\left[-l(Y,f(x))\right] = Pr(Y=1\,|\,X=x)$

Exponential Loss and AdaBoost

- **Direct optimization of the binomial deviance:** $p(x) = Pr(Y' = 1 \,|\, X = x) = \dfrac{1}{1 + \exp(-2f(x))} = \dfrac{e^{f(x)}}{e^{f(x)} + e^{-f(x)}},\ Y' \in \{0,1\}$

  → **Newton step** for fitting an additive logistic model by maximum likelihood:

  For step m, update $f_{m-1}(x) + f(x)$ by the expected log-likelihood

  $$\mathbb{E}[l(Y', f_{m-1}(x) + f(x))] = \mathbb{E}\left[2Y'(f_{m-1}(x) + f(x)) - \log[1 + \exp(2(f_{m-1}(x) + f(x))]\right]$$

  $$s(x) = \left[\frac{\partial \mathbb{E}[l(Y', f_{m-1}(x) + f(x))]}{\partial f(x)}\right]_{f(x)=0} = 2\mathbb{E}[Y' - p(x) \,|\, x],$$

  $$H(x) = \left[\frac{\partial^2 \mathbb{E}[l(Y', f_{m-1}(x) + f(x))]}{\partial f(x)^2}\right]_{f(x)=0} = -4\mathbb{E}[p(x)(1 - p(x)) \,|\, x]$$

  $$f_m(x) \leftarrow f_{m-1}(x) - H(x)^{-1}s(x) = f_{m-1}(x) + \frac{1}{2}\mathbb{E}_{w(x)}\left[\frac{Y' - p(x)}{p(x)(1 - p(x))} \,|\, x\right]$$

  where $\mathbb{E}_{w(x)}(g(x,y) \,|\, x) = \dfrac{\mathbb{E}[w(x)g(x,y) \,|\, x]}{\mathbb{E}[w(x) \,|\, x]}$,

  $w(x) = p(x)(1 - p(x))$

  Weighted least square estimator on $\{x_i, z_i\}_{i=1}^N$ where $z_i = \dfrac{y_i' - p(x_i)}{p(x_i)(1 - p(x_i))}$

Exponential Loss and AdaBoost

- **Direct optimization of the binomial deviance using Newton step:**

LogitBoost (two classes)

1. Start with weights $w_i = 1/N$ $i = 1, 2, \ldots, N$, $F(x) = 0$ and probability estimates $p(x_i) = \frac{1}{2}$.
2. Repeat for $m = 1, 2, \ldots, M$:

   (a) Compute the working response and weights

$$z_i = \frac{y_i^* - p(x_i)}{p(x_i)(1 - p(x_i))},$$

$$w_i = p(x_i)(1 - p(x_i)).$$

   (b) Fit the function $f_m(x)$ by a weighted least-squares regression of z_i to x_i using weights w_i.

   (c) Update $F(x) \leftarrow F(x) + \frac{1}{2} f_m(x)$ and $p(x) \leftarrow (e^{F(x)})/(e^{F(x)} + e^{-F(x)})$.

3. Output the classifier $\text{sign}[F(x)] = \text{sign}[\sum_{m=1}^{M} f_m(x)]$.

Exponential Loss and AdaBoost

- **Newton steps for minimizing exponential loss in forward stagewise additive model:**

Gentle AdaBoost

1. Start with weights $w_i = 1/N, i = 1, 2, \ldots, N, F(x) = 0$.
2. Repeat for $m = 1, 2, \ldots, M$:
   (a) Fit the regression function $f_m(x)$ by weighted least-squares of y_i to x_i with weights w_i.
   (b) Update $F(x) \leftarrow F(x) + f_m(x)$.
   (c) Update $w_i \leftarrow w_i \exp(-y_i f_m(x_i))$ and renormalize.
3. Output the classifier $\text{sign}[F(x)] = \text{sign}[\sum_{m=1}^{M} f_m(x)]$.

Fit $f_m(x)$ by weighted least-squares i.e. $f_m(x) = Pr_w(Y = 1 \mid x) - Pr_w(Y = -1 \mid x)$ rather than $\frac{1}{2} \frac{\log Pr_w(Y = 1 \mid x)}{\log Pr_w(Y = -1 \mid x)}$.

Logit function can be numerically unstable. Gentle AdaBoost outperforms AdaBoost.M1. when stability is an issue.

Loss Functions and Robustness

- **Robust Loss Functions for Classification**

- **Margin**: $y \cdot f(x)$, the margin plays a role analogous to the residuals in regression.

- The classification rule $G(x) = sign[f(x)]$ implies that observations with positive margin $y_i f(x_i) > 0$ are classified correctly whereas those with negative margin $y_i f(x_i) < 0$ are misclassified.

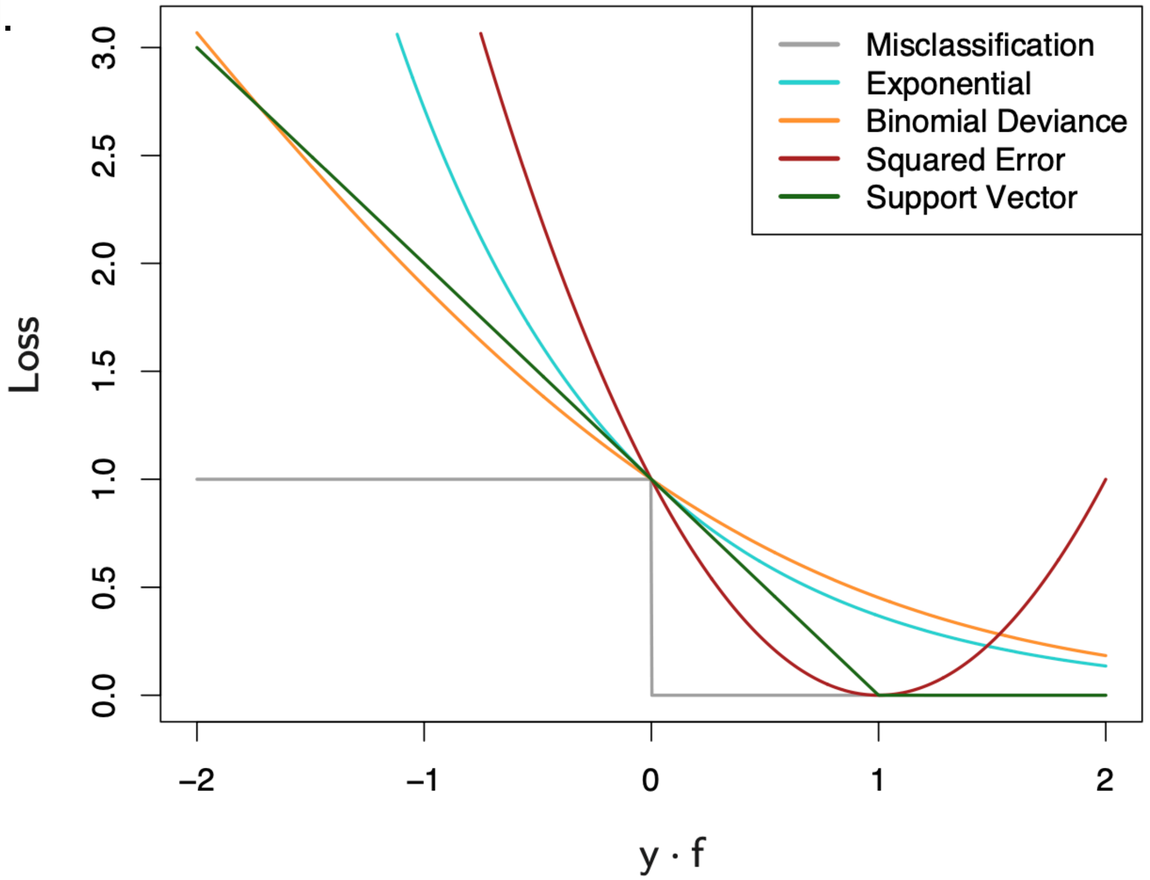
- K**-class classification** $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_K\}$:

  $G(x) = \mathcal{G}_k$ where $k = \arg\max_l p_l(x)$ e.g. softmax function

  $\underset{f(x)}{argmin} \mathbb{E}_{Y|x} L(y, p(x))$ subject to $\sum_{k=1}^{K} f_k(x) = 0$

  Multinomial deviance loss:

$$L(y, p(x)) = -\sum_{k=1}^{K} I(y = \mathcal{G}_k) \log p_k(x)$$



Exponential Loss and AdaBoost

- **Exponential loss in K-class classification:**

$L(\mathbf{y}, \mathbf{f}) = \exp(-\frac{1}{K}\mathbf{y}^T\mathbf{f})$ where $\mathbf{y} = (y_1, \ldots, y_k)$, $y_k = 1$ if $y = \mathcal{G}_k$, $-\frac{1}{K-1}$ o.t. and $\mathbf{f} = (f_1, \ldots, f_k)$ with $\sum_k f_k = 0$

Using Lagrange multipliers, we obtain the population minimizer:

$$f_k^*(x) = (K-1)\left(\log Pr(y = k \,|\, X = x) - \frac{1}{K}\sum_{k'=1}^{K} \log Pr(y = k' \,|\, X = x) \right), \; k = 1,...,K$$

Thus, $\arg\max_k f_k^*(x) = \arg\max_k Pr(y = k \,|\, X = x)$

or $Pr(y = k \,|\, X = x) = \dfrac{\exp(\frac{1}{K-1} f_k^*(x))}{\sum_{l=1}^{K} \exp(\frac{1}{K-1} f_l^*(x))}, \; k = 1,...,K$

Exponential Loss and AdaBoost

- **Forward stagewise additive modeling with K-class exponential loss:**

$$(\beta_m, \mathbf{g}_m) = \underset{\beta,\mathbf{g}}{argmin} \sum_{i=1}^{N} \exp\left(-\frac{1}{K}\mathbf{y}_i^T \mathbf{f}_{m-1}(x_i)\right) \exp\left(-\frac{1}{K}\beta \mathbf{y}_i^T \mathbf{g}(x_i)\right) = \underset{\beta,\mathbf{g}}{argmin} \sum_{i=1}^{N} w_i \exp\left(-\frac{1}{K}\beta \mathbf{y}_i^T \mathbf{g}(x_i)\right)$$

Note. $\mathbf{g}(x) = (g_1(x), \ldots, g_K(x))$ with $g_k(x) = 1$ if $G(x) = k$, $-\frac{1}{K-1}$ o.t. Therefore, solving $\mathbf{g}_m(x)$ is equivalent to finding $G_m(x)$

Similarly to binary case, we can show that

$$G_m = \underset{\beta,G}{argmin} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)), \quad \beta_m = \frac{(K-1)^2}{K}\left(\log\frac{1-err_m}{err_m} + \log(K-1)\right) =: \frac{(K-1)^2}{K}\alpha_m$$

Exponential Loss and AdaBoost

Algorithm 1: SAMME

1. Initialize the observation weights $w_i = 1/N, i = 1, 2, ..., N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute

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

   (c) Compute

$$\alpha_m = \log(\frac{1 - \text{err}_m}{\text{err}_m}) + \log(K - 1).$$

   (d) Set, for $i = 1, ..., N$

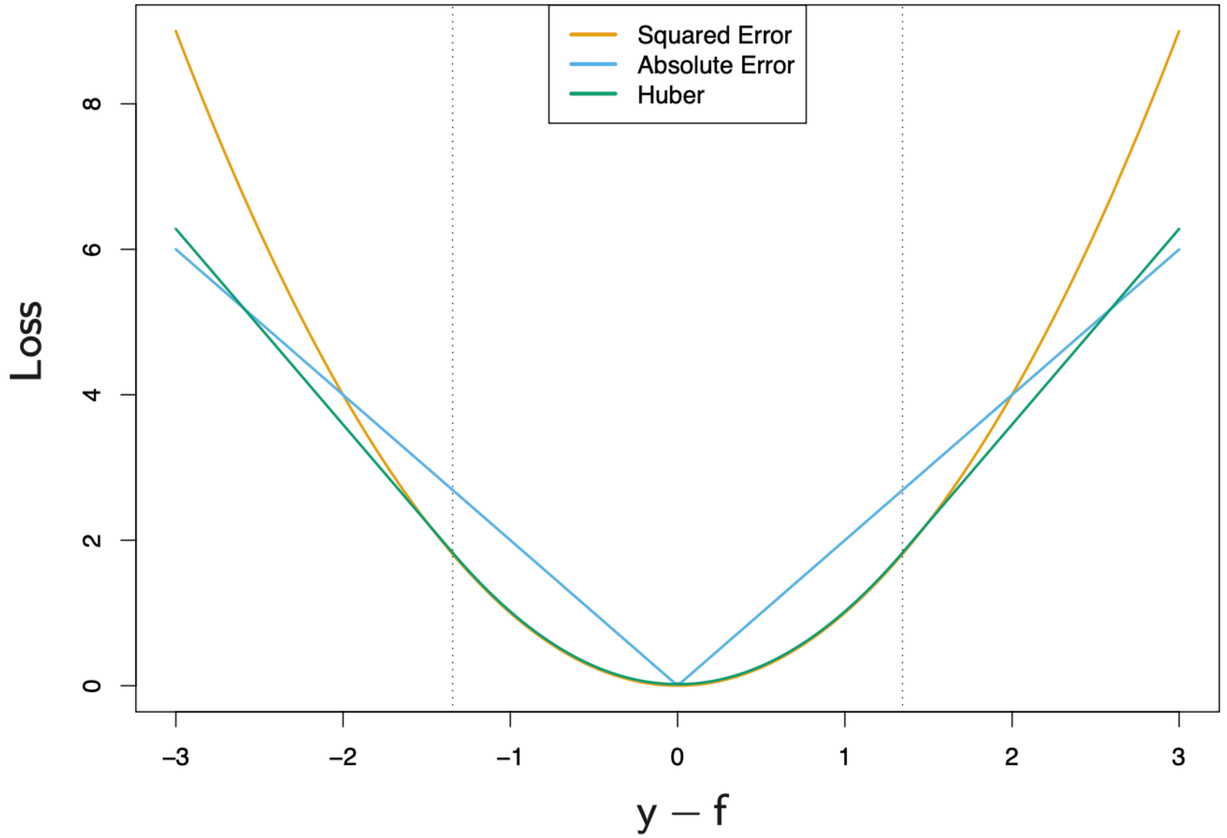
$$w_i \leftarrow w_i \cdot \exp\left(\alpha_m \cdot \mathbf{1}(y_i \neq G_m(x_i))\right)$$

   (e) Output

$$G(x) = \arg\max_k \sum_{m=1}^{M} \alpha_m \cdot \mathbf{1}(G_m(x) = k).$$

Loss Functions and Robustness

- **Robust Loss Functions for Regression**
- **Huber loss:** $L(y, f(x)) = \begin{cases} [y - f(x)]^2 & \text{for } |y - f(x)| \le \delta, \\ 2\delta|y - f(x)| - \delta^2 & \text{otherwise.} \end{cases}$



Boosting Trees

TABLE 10.1. *Some characteristics of different learning methods. Key:* ▲ *= good,* ◆ *=fair, and* ▼ *=poor.*

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

Trees have one aspect that prevents them from being the ideal tool for predictive learning, namely **inaccuracy**.

Boosting decision trees improves their accuracy, often dramatically. At the same time it maintains most of their desirable properties for data mining.

Boosting Trees

- **CART:** $T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j)$ where $\Theta = \{\gamma_j, R_j\}_{j=1}^{J}$ is parameter, and J is meta-parameter
- Θ is found by $\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j)$

  → we usually settle for approximate suboptimal solutions: 1. Finding γ_j given R_j.
  2. Finding R_j

- **The boosted tree model**: $f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m)$ induced in a forward stagewise manner.

  At each step, solve $\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i;\Theta_m))$

Gradient Boosting Trees

- **Numerical optimization in parameterized model:** $\hat{f}(x;\Theta) = \arg\min_{f(x;\Theta)} \mathbb{E}_{Y|x} L(Y, f(x;\Theta)),\ \Theta = \{\theta_j\}_{j=1}^{p}$

  ⟶ $f(x;\Theta)$ is a member of a parameterized class of functions.

  ⟶ Change the function optimization problem to **one of parameter optimization**:

  $\Theta = \arg\min_{\Theta} \mathbb{E}_{Y|x} L(Y, f(x;\Theta)),\ \Theta^* = \sum_{m=0}^{M} \mathbf{h}_m$ where $\mathbf{h}_0$ is initial guess and $\mathbf{h}_m$ is defined by the optimization method.

- **Steepest Descent:** $\mathbf{g}_m = \{g_{jm}\}_{j=1}^{p} = \left\{ \left[\frac{\partial \mathbb{E}_{Y|x} L(Y, f(x;\Theta))}{\partial \theta_j} \right]_{\Theta=\Theta_{m-1}} \right\}_{j=1}^{p}$ where $\Theta_{m-1} = \sum_{i=0}^{m-1} \mathbf{h}_i$

  Then $\mathbf{h}_m = -\rho_m \mathbf{g}_m$ where $\rho_m = \arg\min_{\rho} \mathbb{E}_{Y|x} L(Y, f(x;\Theta = \mathbf{h}_{m-1} - \rho_m \mathbf{g}_m))$

  $-\mathbf{g}_m$: steepest descent direction, ρ_m: line search

Gradient Boosting Trees

- **Nonparametric model:** $\hat{f} = \arg\min_f \mathbb{E}_{Y|x} L(Y, f(x))$ and $(x, y) \sim \mathscr{P}$ is estimated by a finite data sample $\{x_i, y_i\}_{i=1}^N$.

  → $\hat{f}$ can not be estimated accurately, strength must be borrowed from nearby data points by **imposing smoothness** on the solution.

  → Assume a parameterized form and do parameter optimization: $f(x; \Theta) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$, $\Theta = \{\beta_m, \gamma_m\}_{m=1}^M$

  $$\Theta = \{\beta_m, \gamma_m\}_{m=1}^M = \arg \min_{\{\beta'_m, \gamma'_m\}_{m=1}^M} \sum_{i=1}^N L(y_i, \sum_{m=1}^M \beta'_m b(x_i; \gamma'_m))$$

  → Greedy-stagewise approach: $\{\beta_m, \gamma_m\} = \arg\min_{\{\beta,\gamma\}} \sum_{i=1}^N L(y_i, f_{m-1}(x) + \beta b(x_i; \gamma))$, **: It's difficult to calculate for robust criteria!**

  $$f_m(x) = f_{m-1}(x) + \beta_m b(x_i; \gamma_m)$$

  The best greedy step towards the data based estimate of $\hat{f}$,
  under the constraint that the step direction be a member of $b(x; \gamma)$

  → By analogue of the unconstrained negative gradient:

  $$-\mathbf{g}_m = \{g_m(x_i)\}_{i=1}^N = \left\{ \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}} \right\}_{i=1}^N$$

  However, $-\mathbf{g}_m$ is defined only at $\{x_i\}_{i=1}^N$ and cannot be generalized.

  -> Choose the γ_m s.t. produces $\{b(x_i; \gamma_m)\}_{i=1}^N$ most parallel to $-\mathbf{g}_m$.

Gradient Boosting Trees

- $f_m(x) = f_{m-1}(x) + \beta_m b(x_i; \gamma_m)$ where $\{\beta_m, \gamma_m\} = \arg\min_{\{\beta,\gamma\}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x) + \beta b(x_i; \gamma))$,

  → Choose the γ_m s.t. produces $\{b(x_i; \gamma_m)\}_{i=1}^{N}$ most parallel to $-\mathbf{g}_m$. This is the $b(x; \gamma)$ most highly correlated with $g_m(x)$ over the data distribution.

  $\gamma_m = \arg\min_{\gamma,\beta} \sum_{i=1}^{N} [-g_m(x_i) - \beta b(x_i; \gamma)]^2 \approx \arg\min_{\gamma,\beta} \mathbb{E}[-g_m(x) - \beta b(x; \gamma)]^2$: Fitting $b(x; \gamma)$ to the **"pseudo-responses"** $\{r_{im} = -g_m(x_i)\}_{i=1}^{N}$

  This permits the replacement of the difficult function minimization to least square function minimization.

  → The line search: $\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \rho b(x_i; \gamma_m))$

  → Update: $f_m(x) = f_{m-1}(x) + \rho_m b(x_i; \gamma_m)$

- **In regression tree,** $b(x; \gamma) = T(x; \Theta = \{\gamma_j, R_j\}_1^J) = \sum_{j=1}^{J} \gamma_j I(x \in R_j)$

  The update can be expressed as $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

  where $\{R_{jm}\}_{j=1}^{J_m}$ is J_m-terminal node tree given $\{x_i, r_{im}\}_{i=1}^{N}$, $\{\gamma_{jm}\}_{j=1}^{J} = \left\{ \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma) \right\}_{j=1}^{J}$

Gradient Boosting Trees

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: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Gradient Boosting Trees

- **K-class classification:** $\mathcal{G} = \{\mathcal{G}_1, \ldots \mathcal{G}_K\}$

$L(\{y_k, f_k(x)\}_{k=1}^K) = -\sum_{k=1}^K y_k \log p_k(x)$ where $y = \{y_k\}_{k=1}^K$ is dummy variable, $p_k(x) = Pr(y_k = 1 \,|\, X = x) = \dfrac{\exp(f_k(x))}{\sum_{l=1}^K \exp(f_l(x))}$

Then $r_{ki} = -g_k(x_i)$ is $\left[\dfrac{\partial L(\{y_{il}, f_l(x_i)\}_{l=1}^K)}{\partial f_k(x_i)}\right]_{f_k = f_{km}} = y_{ik} - p_{k,m-1}(x_i)$

Thus, K-trees are induced at each step using: Each tree T_{km} has J_{km}-terminal nodes, with corresponding region $\{R_{jkm}\}_{j=1}^{J_{km}}$ given $\{x_i, r_{ki}\}_{i=1}^N$

$$\gamma_{jkm} = \arg\min_{\{\gamma_{jk}\}} \sum_{i=1}^N \sum_{k=1}^K \phi\left(y_{ik},\, f_{k,m-1}(x_i) + \sum_{j=1}^{J_{km}} \gamma_{jk} I(x_i \in R_{jkm})\right) \text{ where } \phi(y_k, f_k) = -y_k \log p_k$$

Above has no closed-form. We approximate this with a single Newton-Raphson using diagonal approximation to Hessian,

$$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jkm}} r_{ik}}{\sum_{x_i \in R_{jkm}} |r_{ik}|(1 - |r_{ik}|)}$$

The final estimates: $G(x) = \mathcal{G}_{\hat{k}(x)}$ where $\hat{k}(x) = \arg\min_k \sum_{l=1}^K L_{lk} p_{lM}(x)$,

$L_{kk'} = loss(\hat{y} = k' \,|\, y = k)$

Gradient Boosting Trees

- **Regularization:** hyper-parameter $(M, \eta, \nu, \{J_m\}_1^M)$

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 **(Subsampling)** Let $N' = \eta N$ (typically $\eta = 0.5$). For $i = n_1, \ldots, n_{N'}$ where $1 \le n_1 < \ldots < n_{N'} \le N$

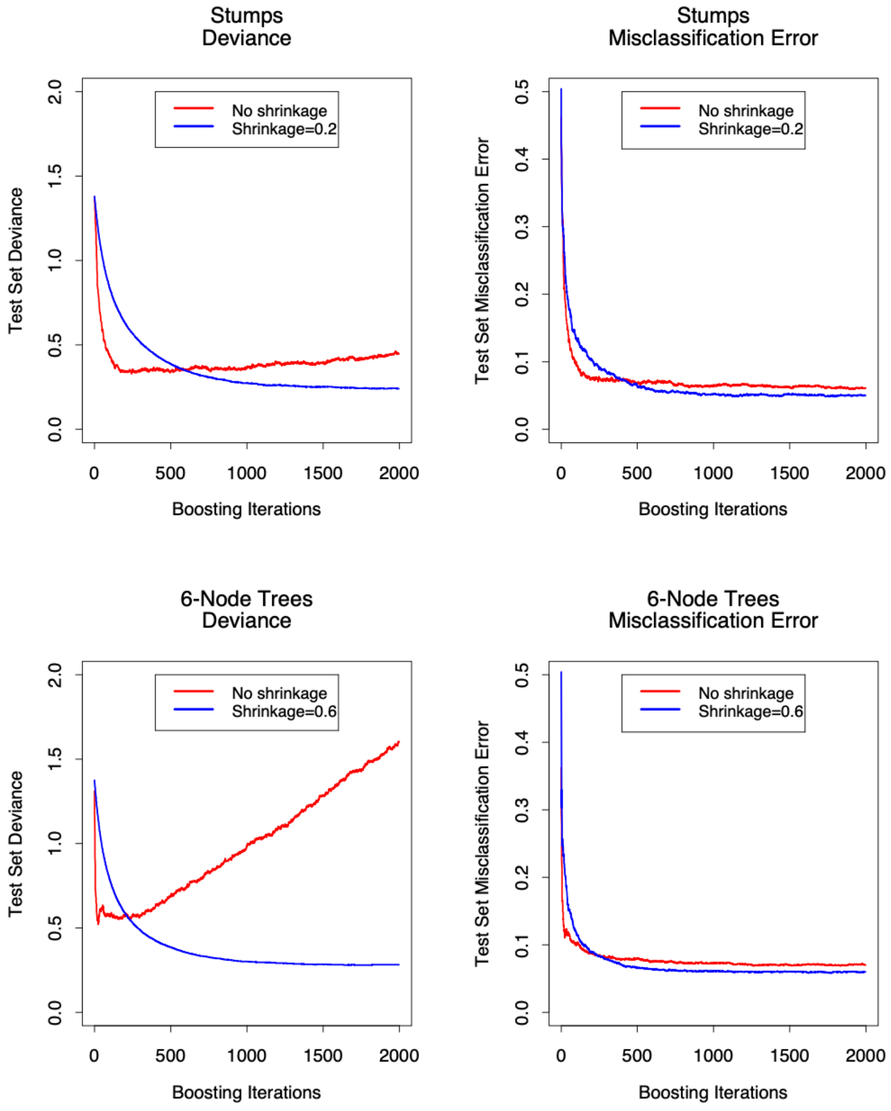
$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$

   (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

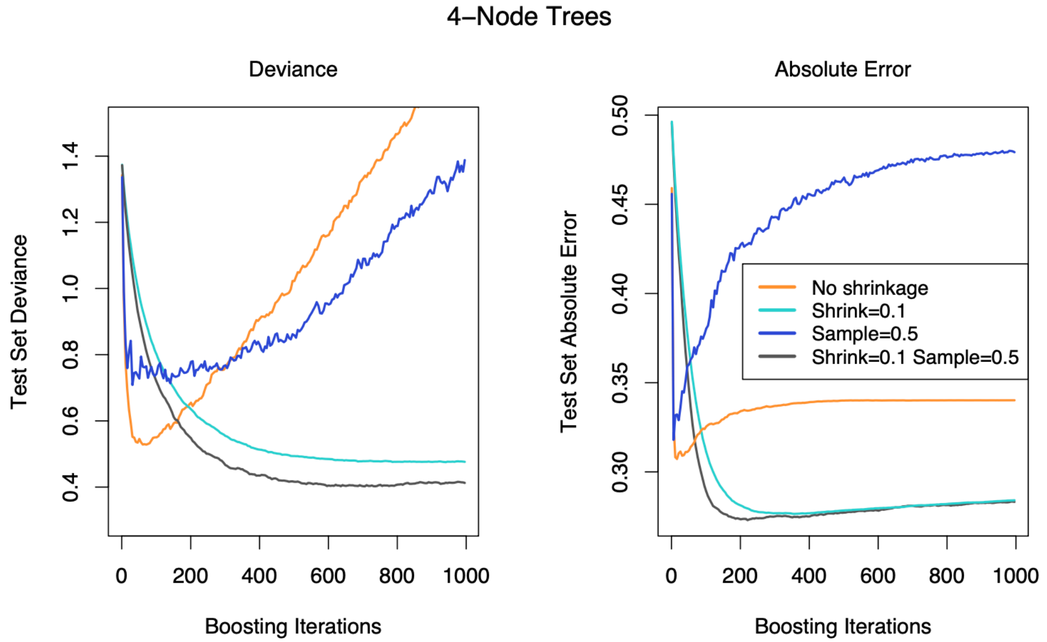
$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$. **(Shirinkage)** $f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm}), \nu \in (0,1)$
3. Output $\hat{f}(x) = f_M(x)$.

Gradient Boosting Trees



Shrinkage



Shirinkage and Subsampling

Gradient Boosting Trees

- **Right-Sized Trees for Boosting:**

  Pruning assumes implicitly that each tree is the final estimator. Except perhaps for the very last tree, this is clearly a very poor assumption.

  → The result is that trees tend to be much too large, especially during the early iterations.
  This substantially degrades performance and increases computation.

  To restrict all trees to be the same size $J = J_m, \forall m$

- A tree with J-terminal nodes produces a function with **interaction order** at most $J - 1$:

$$f(x) = \sum_j f_j(x_j) + \sum_{j,k} f_{jk}(x_j, x_k) + \sum_{jkl} f_{jkl}(x_j, x_k, x_l) + \dots$$

  Experience so far indicates that $J \in \{4,5,6,7,8\}$ works well in the context of boosting.

Interpretation

- **Relative Importance of Predictor Variables:**

For single decision tree, $\mathcal{I}_l^2(T) = \sum_{t=1}^{J-1} i_t^2 I(v(t) = l)$ as a measure of relevance for each $X_j, j = 1,...,p$.

For J- terminal nodes tree, there exists $J-1$ internal nodes of the tree.

At each node t, $X_{v(t)}$ is used to partition the region associated with that node into two subregions

$i_t^2(R_l, R_r) = \frac{w_l w_r}{w_l + w_r}(\bar{y}_l - \bar{y}_r)^2$: The least-squares improvement criterion. w_l, w_r are corresponding sum of the weights.

For M-additive tree expansions $\{T_m\}_1^M$, $\mathcal{I}_l^2 = \frac{1}{M}\sum_{m=1}^{M} \mathcal{I}_l^2(T_m),\ l = 1,...,p$

For K-classification with M-additive tree expansions $\{T_{km}\}$,

$\mathcal{I}_{lk}^2 = \frac{1}{M}\sum_{m=1}^{M} \mathcal{I}_l^2(T_{km}),\ k = 1,...,K$: The relevance of X_l in separating the class k from the other classes.

$\mathcal{I}_l^2 = \frac{1}{K}\sum_{k=1}^{K} \mathcal{I}_{lk}^2,\ l = 1,...,p$: The overall relevance of X_l.

Interpretation

- **Partial Dependence Plots:** After the most relevant variables have been identified, the next step is to attempt to understand the nature of the **dependence** of the approximation $f(X)$ on their joint values.

  Let $X = (X_1, \ldots, X_p)$, $\mathcal{S} \subset \{1,...,p\}$ with $\mathcal{S} \cup \mathcal{C} = \{1,...,p\}$. Then $f(X) = f(X_\mathcal{S}, X_\mathcal{C})$.

  Partial dependence of $f(X)$ on $X_\mathcal{S}$: $f_\mathcal{S}(X_\mathcal{S}) = \mathbb{E}_{X_\mathcal{C}} f(X_\mathcal{S}, X_\mathcal{C})$

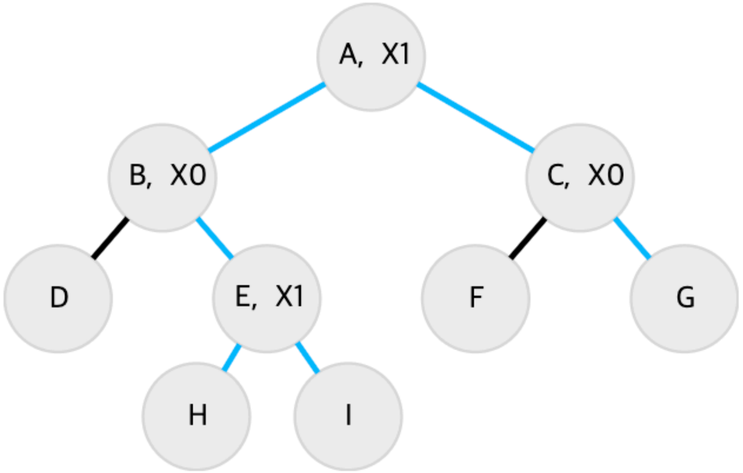
  Partial dependence function can be estimated by $\bar{f}_\mathcal{S}(X_\mathcal{S}) = \frac{1}{N} \sum_{i=1}^{N} f(X_\mathcal{S}, x_{i\mathcal{C}})$

  This can be can be computationally intensive. Fortunately with decision tree, $\bar{f}_\mathcal{S}(X_\mathcal{S})$ can be rapidly computed from the tree itself without reference to the data.

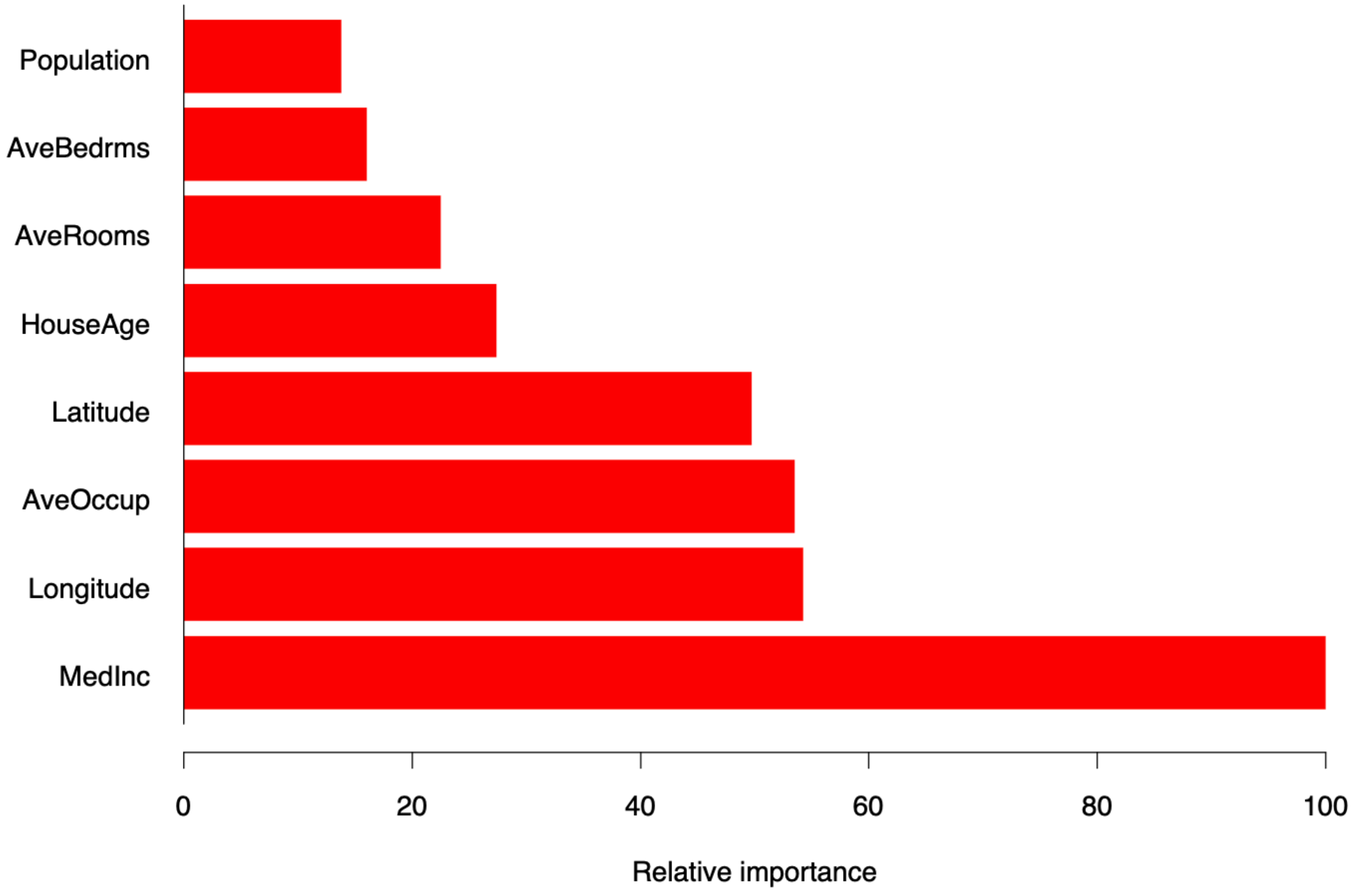
  Ex: In blue are the paths that the fake samples have followed:

  We then just computed the proportion of fake samples landing in each terminal nodes (H, I and G), and averaged the predictions.

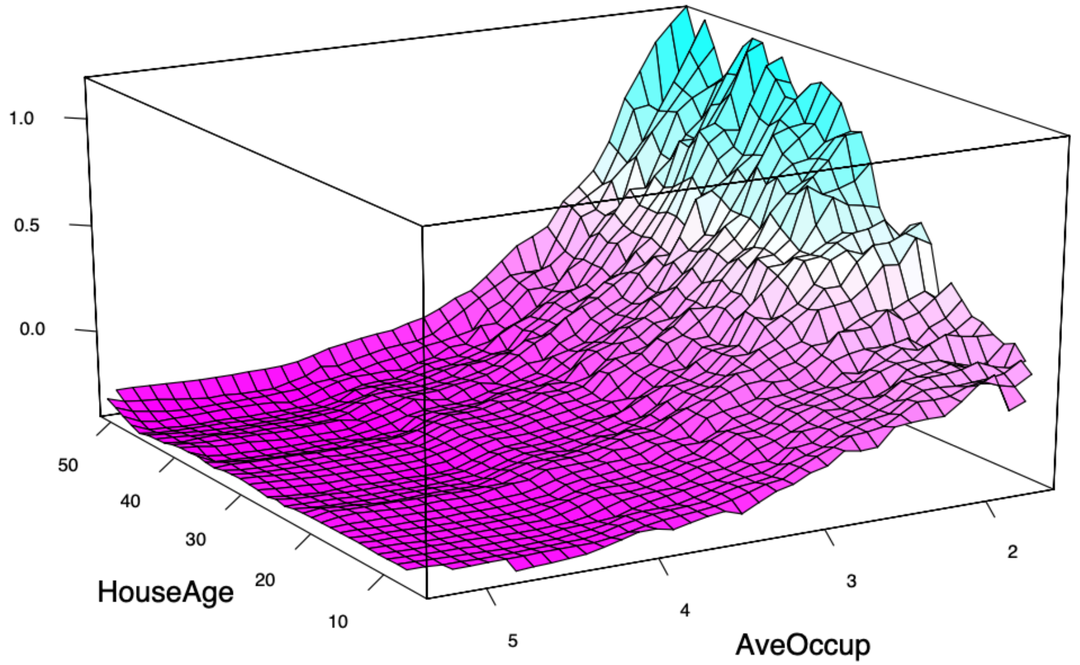
  In decision tree, we can compute the proportion of node **without having to create the samples.-> Tree node search: DFS algorithm**



Interpretation



Relative importance of the predictors for the California housing data.



Partial dependence of house value on median age and aver- age occupancy.

XGBoost

T Chen & Guestrin(2016)

- **Each step, f_m minimizes the following penalized loss function:**

$$\sum_{i=1}^{N} L(y_i, f_{m-1}(x) + T(x_i; \Theta_m)) + \gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} \gamma_{jm}^2, \text{ where } T(x; \Theta_m) = \sum_{j} = 1^{J} \gamma_{jm} I(x \in R_{jm}),\ \Theta_m = \{\gamma_{jm}, R_{jm}\}_{j=1}^{J}$$

- **Approximate Algorithm for Split Finding (parallel algorithm):**

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

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.

Compute the gradients for each bucket and find the best split

XGBoost

- **Sparsity-aware Split Finding:**

  Handing missing data or sparse data.

Algorithm 3: Sparsity-aware Split Finding

Input: I, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
Input: d, feature dimension
Also applies to the approximate setting, only collect statistics of non-missing entries into buckets
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I}, g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  // enumerate missing value goto right
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I_k,$ *ascent order 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
  // enumerate missing value goto left
  $G_R \leftarrow 0,\ H_R \leftarrow 0$
  for j *in sorted*$(I_k,$ *descent order by* $\mathbf{x}_{jk})$ **do**
    $G_R \leftarrow G_R + g_j,\ H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R,\ H_L \leftarrow H - H_R$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split and default directions with max gain

- **Column Subsampling**

  This technique is used in RandomForest.

  Column subsampling **prevents over-fitting** even more so than the traditional row sub-sampling (which is also supported). The usage of column sub-samples also speeds up computations of the parallel algorithm.