Lecture 23: Boosting

Wenbin Lu

Department of Statistics
North Carolina State University

Fall 2019

Outlines

- Adaboost
  - Strong learners vs Weak Learners
  - Motivations
  - Algorithms
- Additive Models
- Adaboost and Additive Logistic Regression
- L_2-Boost
- Boosting Trees
- XGBoost

Boosting Methods

Motivation: Model Averaging

They are methods for improving the performance of weak learners.

- strong learners: Given a large enough dataset, the classifier can arbitrarily accurately learn the target function with probability $1 - \tau$ (where $\tau > 0$ can be arbitrarily small)
- weak learners: Given a large enough dataset, the classifier can barely learn the target function with probability $\frac{1}{2} + \tau$
  - The error rate is only slightly better than a random guessing

Can we construct a strong learner from weak learners and how?

Boosting

Motivation: combines the outputs of many weak classifiers to produce a powerful "committee".

- Similar to bagging and other committee-based approaches
- Originally designed for classification problems, but can also be extended to regression problem.

Consider the two-class problem

- $Y \in \{-1, 1\}$, the classifier $G(\mathbf{x})$ has training error

$$\text{err} = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq G(\mathbf{x}_i))$$

- The expected error rate on future predictions is $E_{\mathbf{X},Y} I(Y \neq G(\mathbf{X}))$.

Classification Trees

Classifications trees can be simple, but often produce noise or weak classifiers.

- Bagging (Breiman 1996): Fit many large trees to bootstrap-resampled versions of the training data, and classify by a majority vote.
- Boosting (Freund & Shapire 1996): Fit many large or small trees to re-weighted versions of the training data. Classify by a weighted majority vote.

In general, Boosting $>$ Bagging $>$ Single Tree

- Breiman's comment "AdaBoost .... best off-the-shelf classifier in the world". (1996, NIPS workshop)

AdaBoost (Discrete Boost)

Adaptively resampling the data (Freund & Shapire 1997; winners of the 2003 Godel Prize)

1. sequentially apply the weak classification algorithm to repeatedly modified versions of the data (re-weighted data)
2. produces a sequence of weak classifiers

$$G_m(\mathbf{x}), \quad m = 1, 2, ..., M.$$

3. The predictions from G_m's are then combined through a weighted majority vote to produce the final prediction

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

Here $\alpha_1, ..., \alpha_M \geq 0$ are computed by the boosting algorithm.

AdaBoost Algorithm

1. Initially the observation weights $w_i = 1/n, i = 1, ..., n$.
2. For m=1 to M
   (a) Fit a classifier $G_m(\mathbf{x})$ to the training data using weights w_i.
   (b) Compute *the weighted error*

$$\text{err}_m = \frac{\sum_{i=1}^{n} w_i I(y_i \neq G_m(\mathbf{x}_i))}{\sum_{i=1}^{n} w_i}.$$

   (c) Compute *the importance* of G_m as

$$\alpha_m = \log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right)$$

   (d) Update $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(\mathbf{x}_i))], i = 1, ..., n$.
3. Output $G(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right)$.

Weights of Individual Weak Learners

In the final rule, the weight of G_m

$$\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right),$$

where err_m is the weight error of G_m.

- The weights α_m's weigh the contribution of each G_m.
- The higher (lower) err_m, the smaller (larger) α_m.
  - The principle is to give higher influence (larger weights) to more accurate classifiers in the sequence.

Data Re-weighting (Modification) Scheme

At each boosting, we impose the updated weights $w_1, ..., w_n$ to samples $(\mathbf{x}_i, y_i), i = 1, ..., n$.

- Initially, all weights are set to $w_i = 1/n$. The usual classifier.
- At step $m = 2, \ldots, M$, we modify weights for observations individually: increasing weights for those observations misclassified by $G_{m-1}(\mathbf{x})$ and decreasing weights for observations classified correctly by $G_{m-1}(\mathbf{x})$.
  - Samples difficult to correctly classify receive ever increasing influence
  - Each successive classifier is forced to concentrate on those training observations that are missed by previous ones in sequence
- The classification algorithm is re-applied to the weighted observations

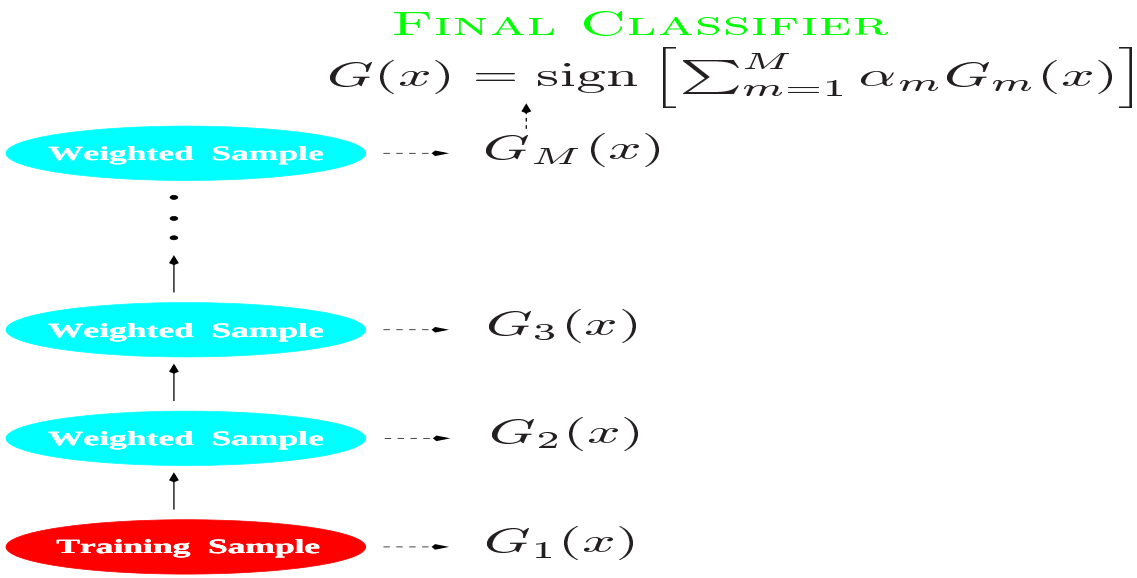

Figure 10.1: *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

Power of Boosting

- Ten features $X_1, ..., X_{10} \sim N(0, 1)$
- two-classes: $Y = 2 \cdot I(\sum_{j=1}^{10} X_j^2 > \chi_{10}^2(0.5)) - 1$
- sample size $n = 2000$, test size $10,000$
- weak classifier: stump (a two-terminal node classification tree)
- performance of boosting and comparison with other methods:
  - stump has 46% misclassification rate
  - 400-node tree has 26% misclassification rate
  - boosting has 12.2% error rate

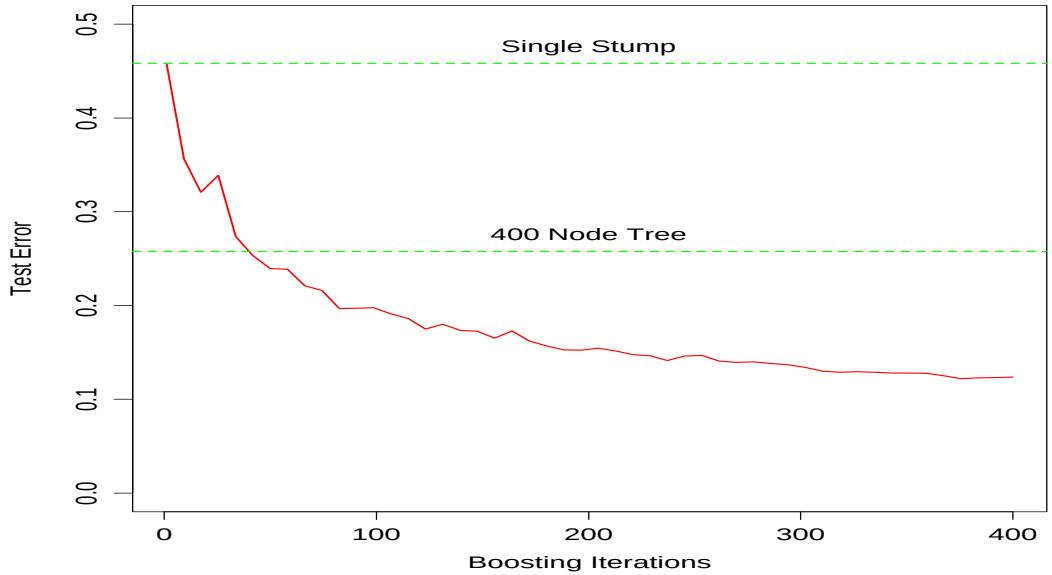

Figure 10.2: *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 400 node classification tree.*

Boosting And Additive Models

The success of boosting is not very mysterious. The key lies in

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

- Adaboost is equivalent to fitting an additive model using the exponential loss function (a very recent discovery by Friedman et al. (2000)).
- AdaBoost was originally motivated from a very different perspective

Introduction on Additive Models

An additive model typically assumes a function form

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

- β_m's are coefficients. $b(\mathbf{x}, \gamma_m)$ are basis functions of $\mathbf{x}$ characterized by γ_m.

The model $\hat{f}$ is obtained by minimizing a loss averaged over the training data

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{n} L\left(y_i, \sum_{m=1}^{M} \beta_m b(\mathbf{x}_i; \gamma_m)\right). \tag{1}$$

It is feasible to rapidly solve the sub-problem of fitting just a single basis.

Forward Stagewise Fitting for Additive Models

Forward stagewise modeling approximate the solution to (1) by

- sequentially adding new basis functions to the expansion without adjusting the parameters and coef. of those that have been added.
- At iteration m, one solves for the optimal basis function $b(\mathbf{x}, \hat{\gamma}_m)$ and corresponding coefficient $\hat{\beta}_m$, which is added to the current expansion $f_{m-1}(\mathbf{x})$.
  - Previously added terms are not modified.
- This process is repeated.

Squared-Error Loss Example

Consider the squared-error loss

$$L(y, f(\mathbf{x})) = [y - f(\mathbf{x})]^2$$

At the mth step, given the current fit $f_{m-1}(\mathbf{x})$, we solve

$$\min_{\beta,\gamma} \quad \sum_i L(y_i, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}, \gamma)) \Longleftrightarrow$$

$$\min_{\beta,\gamma} \quad \sum_i [y_i - f_{m-1}(\mathbf{x}_i) - \beta b(\mathbf{x}_i; \gamma)]^2 = \sum_i [r_{im} - \beta b(\mathbf{x}_i, \gamma)]^2.$$

The term $\hat{\beta}_m b(\mathbf{x}; \hat{\gamma}_m)$ is the best fit to the current residual.
This produces the updated fit

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \hat{\beta}_m b(\mathbf{x}; \hat{\gamma}_m).$$

Forward Stagewise Additive Modeling

1. Initialize $f_0(\mathbf{x}) = 0$.
2. For $m = 1$ to M:
   (a) Compute

$$(\hat{\beta}_m, \hat{\gamma}_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{n} L\left(y_i, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \gamma)\right).$$

   (b) Set $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \hat{\beta}_m b(\mathbf{x}; \hat{\gamma}_m)$

Additive Logistic Models and AdaBoost

Friedman et al. (2001) showed that AdaBoost is equivalent to forward stagewise additive modeling

- using the exponential loss function

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

- using individual classifiers $G_m(\mathbf{x}) \in \{-1, 1\}$ as basis functions

The (population) minimizer of the exponential loss function is

$$\begin{aligned} f^*(\mathbf{x}) &= \arg\min_f E_{Y|\mathbf{x}}[e^{-Yf(\mathbf{x})}] \\ &= \frac{1}{2}\log\frac{\Pr(Y=1|\mathbf{x})}{\Pr(Y=-1|\mathbf{x})}, \end{aligned}$$

which is equal to one half of the log-odds. So, AdaBoost can be regarded as an additive logistic regression model.

Forward Stagewise Additive Modeling with Exponential Loss

For exponential loss, the minimization at mth step in forward stagewise modeling becomes

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{n} \exp\{-y_i[f_{m-1}(x_i) + \beta b(x_i, \gamma)]\}$$

In the context of a weak learner G, this is

$$(\beta_m, G_m) = \arg\min_{\beta,G} \sum_{i=1}^{n} \exp\{-y_i[f_{m-1}(x_i) + \beta G(x_i)]\},$$

$$\text{or equivalently,} \quad (\beta_m, G_m) = \arg\min_{\beta,G} \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta G(x_i)\},$$

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

Iterative Optimization

In order to solve

$$\min_{\beta, G} \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta G(x_i)\},$$

we take the two-step (profile) approach

- first, fix $\beta > 0$ and solve for $\hat{G}$.
- second, solve for β with $G = \hat{G}$.

Recall that both Y and $G(\mathbf{x})$ take only two values $+1$ and -1. So

$$y_i G(\mathbf{x}_i) = +1 \Longleftrightarrow y_i = G(\mathbf{x}_i),$$

$$y_i G(\mathbf{x}_i) = -1 \Longleftrightarrow y_i \neq G(\mathbf{x}_i).$$

Solving for $\hat{G}_m$

$$
\begin{aligned}
(\beta_m, G_m) &= \arg\min_{\beta, G} \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta G(x_i)\} \\
&= \arg\min_{\beta, G} e^{\beta} \sum_{y_i \neq G(x_i)} w_i^{(m)} + e^{-\beta} \sum_{y_i = G(x_i)} w_i^{(m)} \\
&= \arg\min_{\beta, G} (e^{\beta} - e^{-\beta}) \sum_{y_i \neq G(x_i)} w_i^{(m)} + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)}
\end{aligned}
$$

For any $\beta > 0$, the minimizer $\hat{G}_m$ is a $\{-1, 1\}$-valued function

$$
\hat{G}_m = \arg\min_{G} \sum_{i=1}^{n} w_i^{(m)} I[y_i \neq G(x_i)],
$$

the classifier that minimizes training error for the weighted data.

Solving for $\hat{\beta}_m$

Define

$$\text{err}_m = \frac{\sum_{i=1}^n w_i^{(m)} I(y_i \neq \hat{G}_m(\mathbf{x}_i))}{\sum_{i=1}^n w_i^{(m)}}$$

Plugging $\hat{G}_m$ in the objective gives

$$\beta_m = \text{argmin}_\beta \{e^{-\beta} + (e^{\beta} - e^{-\beta})\text{err}_m\} \sum_{i=1}^n w_i^{(m)}$$

The solution is

$$\hat{\beta}_m = \frac{1}{2} \log \left(\frac{1 - \text{err}_m}{\text{err}_m} \right).$$

Connection to Adaboost

Since

$$-yG_m(x) = 2(I(y \neq G_m(x)) - 1,$$

we have

$$\begin{aligned} w_i^{(m+1)} &= w_i^{(m)} \exp\{-\beta_m y_i G_m(\mathbf{x}_i)\} \\ &= w_i^{(m)} \exp\{\alpha_m I(y_i \neq G(\mathbf{x}_i))\} \exp\{-\beta_m\} \end{aligned}$$

where $\alpha_m = 2\beta_m$ and $\exp\{-\beta_m\}$ is constant across the data points. Therefore

- the weight update is equivalent to line 2(d) of the AdaBoost
- line 2(a) of the Adaboost is equivalent to solving the minimization problem

Weights and Their Update

- At the mth step, the weight for the ith observation is

$$w_i^{(m)} = \exp\{-y_i f_{m-1}(\mathbf{x}_i)\},$$

  which depends only on $f_{m-1}(\mathbf{x}_i)$ but not on β or $G(\mathbf{x})$.
- At the $(m+1)$th step, we update the weight using the fact

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

It leads to the following update formula

$$\begin{aligned} w_i^{(m+1)} &= \exp\{-y_i f_m(\mathbf{x}_i)\} \\ &= \exp\{-y_i(f_{m-1}(\mathbf{x}_i) + \beta_m G_m(\mathbf{x}_i))\} \\ &= w_i^{(m)} \exp\{-\beta_m y_i G_m(\mathbf{x}_i)\}. \end{aligned}$$

Why Exponential Loss?

- Principal virtue is computational
- Exponential loss concentrates much more influence on observations with large negative margins $yf(x)$. It is especially sensitive to misspecification of class labels.
- More robust losses: computation is not as easy (Use Gradient Boosting).

Exponential Loss and Cross Entropy

Define $Y' = (Y+1)/2 \in \{0,1\}$.
The binomial negative log-likelihood loss function is

$$\begin{aligned} -l(Y, p(\mathbf{x})) &= -\left[Y' \log p(\mathbf{x}) + (1-Y') \log(1-p(\mathbf{x}))\right] \\ &= \log\left(1+\exp\{-2Yf(\mathbf{x})\}\right) \equiv -l(Y, f(\mathbf{x})), \end{aligned}$$

where $p(\mathbf{x}) = [1+e^{-2f(\mathbf{x})}]^{-1}$.

- The population minimizers of the deviance $E_{Y|\mathbf{x}}[-l(Y, f(\mathbf{x}))]$ and $E_{Y|\mathbf{x}}[e^{-Yf(\mathbf{x})}]$ are the same.
- The population minimizer is given by

$$f^*(\mathbf{x}) = \frac{1}{2} \log \frac{\Pr(Y=1|\mathbf{x})}{\Pr(Y=-1|\mathbf{x})}.$$

Loss Functions for Classification

Choice of loss functions matters for finite data sets.
The *Margin* of $f(\mathbf{x})$ is defined as $yf(\mathbf{x})$.
The classification rule is $G(\mathbf{x}) = \text{sign}[f(\mathbf{x})]$
The decision boundary is $f(\mathbf{x}) = 0$

- Observations with positive margin $y_i f(\mathbf{x}_i) > 0$ are correctly classified
- Observations with negative margin $y_i f(\mathbf{x}_i) < 0$ are incorrectly classified

The goal of a classification algorithm is to produce positive margins as frequently as possible

- Any loss function should penalize negative margins more heavily than positive margins, since positive margin are already correctly classified

Various Loss Functions

Monotone decreasing loss functions of the margin

- Misclassification loss: $I(\text{sign}(f(\mathbf{x})) \neq y)$
- Exponential loss: $\exp(-yf)$ (not robust against mislabeled samples)
- Binomial deviance: $\log\{1 + \exp(-2yf)\}$ (more robust against influential points)
- SVM loss: $[1 - yf]_+$

Other loss functions

- Squared error: $(y - f)^2 = (1 - yf)^2$ (penalize positive margins heavily)

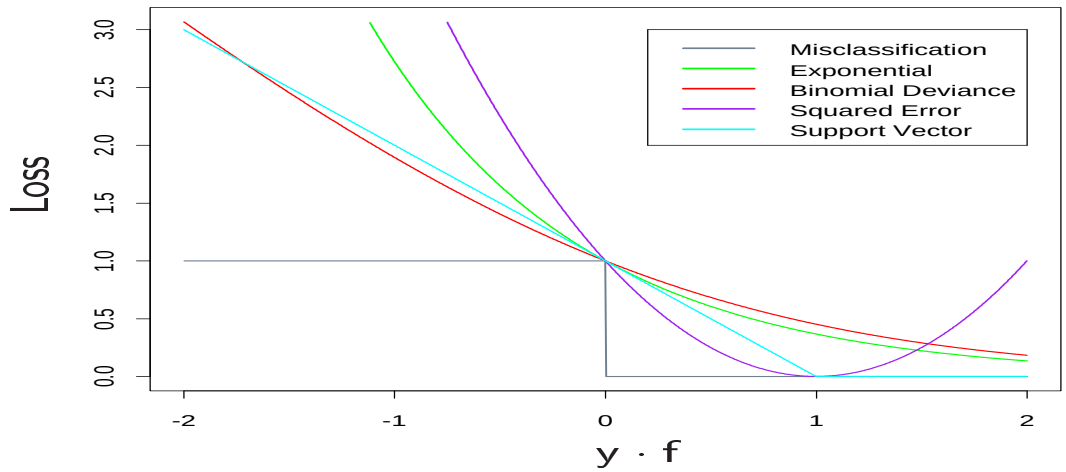

Figure 10.4: *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\text{sign}(f)$*. The losses are misclassification:* $I(\text{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1 + \exp(-2yf))$*; squared error:* $(y - f)^2$*; and support vector:* $(1 - yf) \cdot I(yf > 1)$ *(see Section 12.3). Each function has been scaled so that it passes through the point* $(0, 1)$*.*

Brief Summary on AdaBoost

- AdaBoost fits an additive model, where the basis functions $G_m(x)$ stage-wise optimize exponential loss
- The population minimizer of exponential loss is the log odds
- There are loss functions more robust than squared error loss (for regression problems) or exponential loss (for classification problems)

L_2 Boosting

- Proposed by Buhlmann & Yu (2003).
- Adopt the general framework of the gradient boosting machine developed by Friedman (2001).
- Data: (Y_i, X_i), $i = 1, \cdots, n$, where Y_i is the response and X_i are p-dimensional features.
- Problem: find $F : R^p \to R$ to minimize the expected loss $E[L\{Y_i, F(X_i)\}]$.
- Define empirical loss: $L_n(F) = \sum_{i=1}^{n} L\{Y_i, F(X_i)\}$.
- Estimate F nonparametrically using functional gradient descent.
- The procedure includes: initiation, evaluation of negative gradient of the loss function, projection of the gradient to a base learner, line search, update, and iterations.
- Common choice: component-wise cubic smoothing splines as the base learner.

Algorithm

- Step 1. Set $m = 0$ and $\widehat{F}_m(\cdot) = 0$.
- Step 2. Compute the negative gradient as

$$U_i^{(m)} = -\frac{\partial L(Y_i, F)}{\partial F} | F = \widehat{F}_m(X_i), \ i = 1, \cdots, n.$$

- Step 3. Projection: regress $U_i^{(m)}$ on $X_{i,k}$ using univariate cubic smoothing spline as the base learner, and choose the predictor $X_{k^{(m)}}$ such that

$$k^{(m)} = \arg \min_{1 \le k \le p} \sum_{i=1}^{n} \left(U_i^{(m)} - g_k(X_{i,k}) \right)^2,$$

where $g_k(\cdot)$ is the conventional univariate cubic smoothing spline fit.

Algorithm

- Step 4. Line search: find w_m to minimize

$$\sum_{i=1}^{n} L\{Y_i, \widehat{F}_m(X_i) + w g_k(X_{i,k^{(m)}})\}$$

  w.r.t. w.

- Step 5. Update: set

$$\widehat{F}_{m+1}(X_i) = \widehat{F}_m(X_i) + \rho w^{(m)} g_k(X_{i,k^{(m)}}),$$

  and increase m by one. Here, ρ is a small learning rate, usually chosen as $\rho = 0.05$ or 0.01.

- Step 6. Iteration: repeat Steps 2 to 5.

Discussions

- Tuning parameters: learning rate ρ and total number of iterations.
- Both can be tuned via cross-validation.
- Empirical findings: a smaller learning rate would often lead to a better predictive performance of the boosting algorithm; the predictive performance is not overly sensitive to the choice of ρ.
- Stopping rule: stop the iteration when the loss function flattens. This gives almost the same performance as cross-validation.
- Variable importance measure (Friedman, 2001):

$$I_j = \left[E_X \left\{ \frac{\partial \widehat{F}(X)}{\partial X_j} \right\}^2 \mathrm{Var}(X_j) \right]^{1/2}, \ j = 1, \cdots, p,$$

where the expectation with respect to the marginal distribution of X and can be computed through sample average.

Boosting Trees

- A tree can be expressed as

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

with parameters $\Theta = (\gamma_j, R_j)_1^J$.

- The parameters are found by minimizing the empirical risk

$$\widehat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j) = \arg\min_{\Theta} \sum_{i=1}^{N} L(y_i, T(x_i, \Theta)).$$

- Finding γ_j given R_j: it can be trivially done.
- Finding R_j: this is the difficult part. A typical strategy is to use a greedy, top-down recursive partitioning algorithm. A smooth function may be used to approximate L.

Boosted Tree Model

- The boosted tree model

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

  induced in a forward stagewise manner.

- At each step, we solve

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

  where $\Theta_m = (\gamma_{jm}, R_{jm}), j = 1, \cdots, J_m$.

Gradient Boosting

- Consider the numerical optimization

$$\arg \min_{f \in \{f_M\}} L(f) \equiv \sum_{i=1}^{N} L(y_i, f(x_i)).$$

- The solution is written as a sum of component vectors

$$f_M = \sum_{m=0}^{M} h_m, \qquad h_m \in R^N,$$

where $f_M = (f_M(x_1), \cdots, f_M(x_N))^T$ and h_m is the increment vector at the mth step.

- Steepest descent: choose $h_m = -\rho_m g_m$, where ρ_m is a scalar and the ith component of g_m is given by

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

Gradient Boosting

- The step length ρ_m is the solution to

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) - \rho g_m(x_i)).$$

- Solution update: $f_m(x_i) = f_{m-1}(x_i) - \rho_m g_m(x_i)$.
- Gradient boosting based on least squares:

$$\widetilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i, \Theta))^2.$$

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 $i = 1, \cdots, N$, compute $r_{im} = -g_{im}$.
   b. Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, ..., J_m$.
   c. For $j = 1, 2, ..., J_m$, compute

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

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

XGBoost

- e**X**treme **G**radient **B**oosting (XGBoost): a scalable tree boosting system. Capable to handle large data sets.
- a variant of gradient tree boosting, also known as gradient boosting machine (GBM) or gradient boosted regression tree (GBRT).
  - Builds an ensemble of weak learners in stage-wise manner.
  - Combines gradient descent with CART.
- successfully used in many machine learning and data mining challenges, such as Kaggle.
  - Among the 29 challenge winning solutions published at Kaggle's blog during 2015, 17 solutions used XGBoost.
  - In addition, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles.

Gradient Tree Boosting

- Data with n samples and m features: $\mathcal{D} = \{(x_i, y_i) : i = 1, \cdots, n\}$, $x_i \in R^m$ and $y_i \in R$.
- A tree ensemble model uses K additive functions:

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

where $\mathcal{F} = \{f(x) = w_{q(x)} : q : R^m \to T, w \in R^T\}$ is the space of regression trees (CART).

  - q: represents the structure of each tree;
  - T: number of leaves of each tree;
  - w: leaf weights (the score of each leaf).

Gradient Tree Boosting

- Minimize the following objective function:

$$L(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k),$$

where $\Omega(f) = \gamma T + (\lambda/2)||w||^2$.

- Consider $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$ and

$$L^{(t)}(\phi) = \sum_i l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t).$$

- Consider second-order approximation:

$$\begin{aligned} \tilde{L}^{(t)}(\phi) &= \sum_i [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \\ &= \sum_i [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + \text{constant}, \end{aligned}$$

where g_i and h_i are the first and second order derivatives of l w.r.t $\hat{y}^{(t-1)}$.

Recap of XGBoost

Algorithm:

1. Initialize $\hat{y}^{(0)}(x) = \hat{f}_0(x) = \arg\min_{f_0} \sum_{i=1}^{n} l(y_i, f_0(x_i)) + \Omega(f_0)$.
2. For $t = 1, \cdots, K$, do
   - Calculate g_i and h_i, $i = 1, \cdots, n$;
   - Fit a new tree $\hat{f}_t = \arg\min_{f_t} \sum_i [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$; (*)
   - Update $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \hat{f}_t(x_i)$.
   - Output $\hat{y}^{(K)}$.

Combination of Gradient Boosting with CART

To optimize (*):

- Define the instance set in leaf j as $I_j = \{i : q(x_i) = j\}$.
- Rewrite $\tilde{L}^{(t)}(\phi)$ as

$$\begin{aligned}\tilde{L}^{(t)}(\phi) &= \sum_i [g_i w_{q(x_i)} + \frac{1}{2} h_i w^2_{q(x_i)}] + \gamma T + \frac{\lambda}{2} \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T.\end{aligned}$$

- For a fixed structure q, the optimal weight w_j^* of leaf j is given by

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

Splitting Trees

- Define the optimal value of a given tree structure q

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

- The optimal value is like the impurity score for evaluating decision trees. A greedy algorithm can be used to build the tree (a single node is added at each step).
- Let I_L and I_R denote the instance set of left and right nodes after the split. Letting $I = I_L \cup I_R$. The loss reduction after the split is given by

$$L_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma.$$

- The above is usually used for evaluating the split candidates.

Splitting Algorithms

- Exact greedy algorithm: potentially time-consuming for continuous variables.
- Approximate greedy algorithm:
  - Proposes candidate splitting points according to percentiles of feature distribution.
  - Maps the continuous features into buckets split by these candidate points.
  - Aggregates the statistics and finds the best solution among proposals based on the aggregated statistics.
- Sparsity-aware split finding: features are sparse
  - presence of missing values in the data.
  - frequent zero entries in the statistics.

Refine Prediction

- Pruning
  - The gain of the split L_{split} can be negative.
  - Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain.
- Shrinkage: $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \epsilon \hat{f}_t(x_i)$, ϵ is the learning rate usually set at around 0.1.
- Feature subsampling: as used in random forests.
- XGBoost can be implemented in R ("xgboost" package), Python, and Julia.