Boosting

Xiaogang Su, Ph.D.
Department of Mathematical Sciences
University of Texas at El Paso (UTEP)
xsu@utep.edu

April 28, 2018

Contents

1 Introduction

The boosting algorithm has attracted much attention in the machine learning and statistics community. Breiman (1998) crowned it as "the best off-the-shelf classifier" (noting that this was before his proposal of random forest). Off-the-shelf methods are generally referred to those that can be applied directly to data without requiring a great deal of time consuming data preprocessing or careful tuning of the learning procedure.

Figure 1: Stump as a ‘Weak Base Learner’.

Recall the Winnow algorithm for committee learning where a number of weak committee members each predicts the outcomes. Winnow algorithm incrementally reduces the weights for those wrong members while increasing the weights for correct members. Suppose that each committee member is from a weak learning model. Would similar weighting strategy help combine them to arrive at a strong learner? Is there a principled way of conducting this type of analysis? These questions motivate algorithms such as boosting.

More formally, a ‘weak learner’ is a classifier that performs barely better than flopping a coin while a probably approximately correct (PAC) learner, on the other hand, has the Bayes risk asymptotically with probability tending to 1. Boosting is motivated by the question of whether and, if so, how a number of weaker learners can be combined into a PAC learner. The first boosting algorithm AdaBoost, due to Freund and Schapire (1995; 1996; 1997) and a series of papers, was originally proposed as ensemble methods for classification, which rely on the principle of generating multiple predictions and majority voting (averaging) among the individual classifiers. Various versions of the AdaBoost algorithm have proven to be very competitive in terms of prediction accuracy in a variety of applications.

Among others, Breiman (1998; 1999) observed that the AdaBoost algorithm can be viewed as a gradient descent algorithm in function space, inspired by numerical optimization and statistical estimation. On this basis, he proposed the Arcing (adaptive resampling and combining) algorithm.

The path-breaking work of Friedman, Hastie, and Tibshirani (2000) laid out important foundations which linked AdaBoost and other boosting algorithms to the framework of statistical estimation and additive basis expansion. In their terminology, boosting is represented as “forward stagewise additive modeling”: the word “additive” doesn’t imply a model fit which is additive in the covariates, but refers to the fact that boosting is an additive (in fact, a linear) combination of simple basis functional estimators. Efron, et al. (2004) pointed out the relation between boosting and ℓ_1-penalized estimation. The insights of Friedman, Hastie, and Tibshirani (2000) opened new perspectives to use boosting methods in many other contexts than classification. Extensions of boosting has been made to regression, density estimation, for survival analysis or for multivariate analysis. See, e.g., the stochastic gradient boosting (SGB) method of Friedman (2001). In quite a few of these proposals, boosting is not only a black-box prediction tool but also an estimation method for models with a specific structure such as linearity or additivity. Boosting can then be seen as an interesting regularization scheme for estimating a model. Buhlmann and Hothorn (2007) provides a nice overview of the developments and implements of boosting.

In the following discussion, our focus is on the binary classification problems. The available data consist of $\{(\mathbf{x}_i, y_i) : i = 1, \ldots, n\}$, where $y_i \in \{\pm 1\}$ instead of 1/0. With this notation, a misclassification event $y \neq \hat{y}$ is equivalent to $y\hat{y} < 0$ for a predicted response $\hat{y}$.

2 AdaBoost

Adaptive Boosting (AdaBoost) generates a sequentially weighted set of *weak base classifiers* that are combined to form an overall strong classifier. A 'weak' learner is one whose performance is slightly better than random guessing. For example, a 'stump', a two-terminal node tree (see Figure 1), is often used.

2.1 The AdaBoost Algorithm

The specific procedure of AdaBoost is presented below in Algorithm 1. The weighting strategy plays a critical role in AdaBoost. Note that there are two sets of weights: one set $\{w_i : i = 1, \ldots, n\}$ for the individual observations and the other $\{\beta_m : m = 1, \ldots, M\}$ for each base decision tree model $\mathcal{T}_m$. They are assigned and updated with different principles: for individual weights w_i, larger weights w_i are given to observations that is misclassified while smaller w_i values are given to those correctly classifier; for model weights β_m, it can be seen that larger β_m values are given to models with good predictive performances while smaller β_m values are given to models with poor predictive performances.

Algorithm 1 AdaBoost (Adaptive Boosting) Algorithm

```
Give training data $\mathcal{L} = \{(\mathbf{x}_i, y_i) : i = 1, \ldots n\}$, want to predict a new observation at $\mathbf{x}_0$.
**Initialize** $h(\mathbf{x}_0) = 0$ and weights $w_i = 1/n$ for $i = 1, \ldots, n$; and set $J, M \in \mathbb{N}$;
**for** $m = 1, 2, ..., M$ **do**
    Train a decision tree $\mathcal{T}_m(\cdot)$ of size $J$ with weight $w_i$;
    Compute weighted error rate $\epsilon_m = \sum_{i=1}^n w_i I\{y_i \neq G_m(\mathbf{x}_i)\}$;
    Compute half log-odds for correct classification $\beta_m = \frac{1}{2}\log\frac{1-\epsilon_m}{\epsilon_m}$;
    Update $w_i := w_i \exp\{-\beta_m \cdot y_i \mathcal{T}_m(\mathbf{x}_i)\}$ for $i = 1, \ldots, n$.
    Normalize $w_i := w_i / \sum w_i$ so that $\sum w_i = 1$.
    Update $h(\mathbf{x}_0) := h(\mathbf{x}_0) + \beta_m \mathcal{T}_m(\mathbf{x}_0)$;
**end for**
Output $\hat{y}_0 = \mathrm{sgn}\{h(\mathbf{x}_0)\}$.
```

The final prediction that AdaBoost makes at $\mathbf{x}_0$ can be expressed as

$$\hat{y}_0 \;=\; \text{sgn}\,\{h(\mathbf{x}_0)\} \;=\; \text{sgn}\left\{\sum_{m=1}^{M} \beta_m \mathcal{T}_m(\mathbf{x}_0)\right\}. \tag{1}$$

A schema of boosting is provided in Figure 2.[1] In each step, AdaBoost attempts to find a classifier (i.e., a decision tree of size J) according to the current distribution of weights on the observations. To understand the weights w_i, first note that we require that $\mathcal{T}_m$ is weak learner with error rate $\epsilon_m < 0.5$. In fact, this is always guaranteed by majority voting. It follows that $\beta_m = (1/2)\log\{(1-\epsilon_m)/\epsilon_m\} > 0$ and hence $\exp(\beta_m) \geq 1$ and $\exp(-\beta_m) \leq 1$. If an observation is incorrectly classified (i.e., $y_i \mathcal{T}_m(\mathbf{x}_i) = -1 < 0$) using the current distribution of weights, then the observation will receive more weight in the next iteration, up-weighted by a factor $\exp(\beta_m) = (1-\epsilon_m)/\epsilon_m > 1$. On the other hand, correctly classified observations (i.e., $y_i G_m(\mathbf{x}_i) = 1 > 0$) under the current distribution of weights will receive less weight in the next iteration, down-weighted by a factor $\exp(-\beta_m) = \epsilon_m/(1-\epsilon_m) < 1$.

In the final overall model (1), classifiers $\mathcal{T}_m$ that predict more accurately (i.e., smaller ϵ_m) on the training data receive more weight β_m is decreasing in ϵ_m, whereas classifiers that predict poorly receive less weight, since as $\beta_m = (1/2)\log\{(1-\epsilon_m)/\epsilon_m$ is decreasing in ϵ_m.

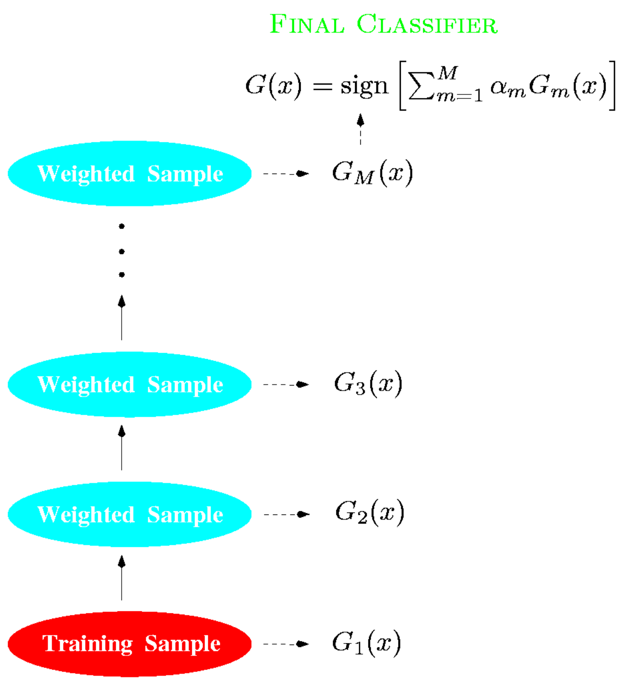

Figure 2: Schema of Boosting.

Boosting is an important development in classification and predictive modeling. It consistently

[1] The figure is taken from Hastie, Tibshirani, and Friedman (2008), where different notations are used. In particular, $G_m(\cdot) = \mathcal{T}(\cdot)$ and $\alpha_m = \beta_m$. The new notations that I used may serve better later in explaining why AdaBoost is a forward stagewise additive model with the exponential loss.

produces significantly lower error rates than a single decision tree and many other classification methods. Very interestingly, the test error from boosting seems to consistently decrease and then level off as more base classifiers are added, rather than ultimately increases. It has been debated until about the year of 2000 whether the AdaBoost algorithm is immune to overfitting when running more iterations, i.e., stopping wouldn't be necessary. It is clear nowadays that AdaBoost and also other boosting algorithms are overfitting eventually, and early stopping is necessary. Nevertheless, the AdaBoost algorithm is quite resistant to overfitting, especially when misclassification rate is used as the performance criterion. It has been found that boosting's variance increases with exponentially small increments while its squared bias decreases exponentially fast as the number of iterations grow. This also explains why boosting's overfitting kicks in very slowly.

It is also worth noting that there are two natural ways of using the weights at each step of boosting: one is to conduct a weighted analysis and the other is to re-sampling data with weighted probability. Earlier in the AdaBoost Algorithm, the weighted (tree) model fitting was alternatively executed with weighted resampling. But the performance of the current version seems comparably better.

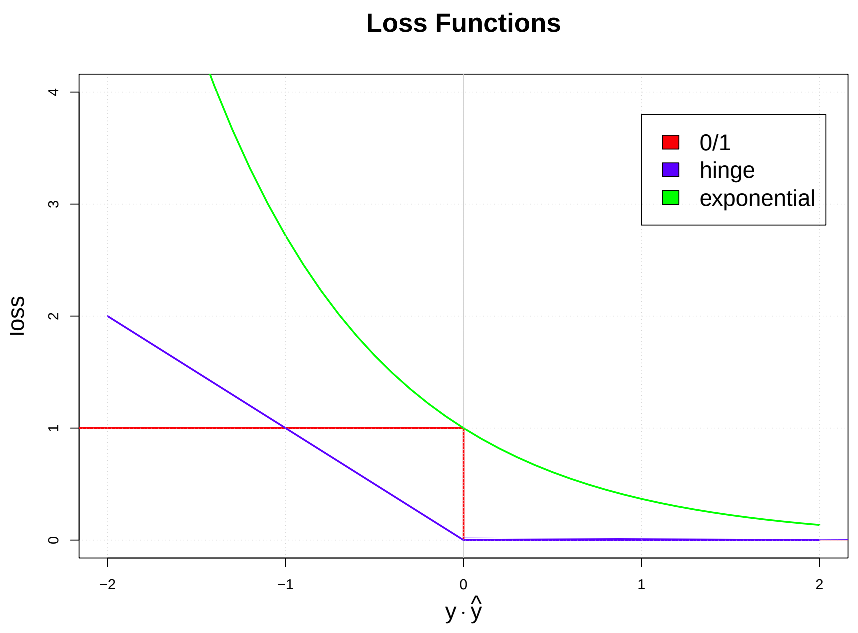

Figure 3: Three loss functions: the 0/1 loss, the hinge loss, and the exponential loss as a function of $y\hat{y}$. Missclassification occurs when $y\hat{y} < 0$.

2.2 Statistical Perspective of AdaBoost

Friedman, Hastie, and Tibshirani (2000) provides a statistical explanation of AdaBoost. In view of (1), AdaBoost essentially fits a forward stagewise additive model with the exponential loss. Let's first take a look at the exponential loss function, defined as $l(y; \hat{y}) = \exp(-y\hat{y})$. A plot of the smooth exponential loss as a function of $y\hat{y}$ is given in Figure 3, together with the 0/1 and hinge losses.

Same as before, we can also write the loss function as $l(y; h(\mathbf{x}))$ for a discriminant function $h(\mathbf{x})$. The theoretical minimizer of the exponential risk is characterized as the half log odds for having $y = +1$ by Lemma 1. Equivalently, $\pi(\mathbf{x}) = e^{h(\mathbf{x})} / \left\{ e^{h(\mathbf{x})} + e^{-h(\mathbf{x})} \right\}$. This explains why it makes sense to take the sign of $\{h(\mathbf{x})\}$ for binary prediction.

Lemma 1. *Denote* $\pi(\mathbf{x}) = \Pr(y = 1|\mathbf{x})$. *We have*

$$h^\star(\mathbf{x}) \;=\; \underset{h}{\operatorname{argmin}}\, E_{\mathbf{x},y} \exp\{-yh(\mathbf{x})\} \;=\; \frac{1}{2} \log \frac{\pi(\mathbf{x})}{1-\pi(\mathbf{x})}. \tag{2}$$

Proof. It suffices to consider

$$E_{y|\mathbf{x}} \exp\{-yh(\mathbf{x})\} \;=\; \pi(\mathbf{x}) \exp\{-h(\mathbf{x})\} + \{1 - \pi(\mathbf{x})\} \exp\{h(\mathbf{x})\} \;\triangleq\; Q(h),$$

which has the derivative with respect to $h(\cdot)$ as follows

$$\frac{dQ(h)}{dh} = -\pi(\mathbf{x}) \exp\{-h(\mathbf{x})\} + \{1 - \pi(\mathbf{x})\} \exp\{h(\mathbf{x})\}.$$

Setting $dQ(h)/dh = 0$ and solving for $h(\mathbf{x})$ would complete the proof. $\square$

The other ingredient for boosting is forward stagewise additive models. An additive model has the following form for the regression or discriminant function

$$h_M(\mathbf{x}) \;=\; \sum_{m=1}^{M} \beta_m b(\mathbf{x}, \boldsymbol{\gamma}_m) \;=\; h_{M-1}(\mathbf{x}) + \beta_M b_m(\mathbf{x}, \boldsymbol{\gamma}_M), \tag{3}$$

where M is the total number of additive terms and $b_m(\mathbf{x}, \boldsymbol{\gamma}_m)$ is a basis function characterized by parameter $\boldsymbol{\gamma}_m$. In AdaBoost, each basis function consists of a decision tree $\mathcal{T}_m$ of size J. Let $\mathbf{T}_m$ denote the dummy variable matrix induced by tree $\mathcal{T}_m$ and $\boldsymbol{\gamma}_m$ are essentially the predicted values at the terminal nodes of $\mathcal{T}_m$. It follows that

$$b_m(\mathbf{x}, \boldsymbol{\gamma}_m) \;=\; \mathcal{T}_m(\mathbf{x}) \;=\; \mathbf{T}_m \boldsymbol{\gamma}_m.$$

With the forward stagewise procedure, $h_M(\mathbf{x})$ in the additive model (3) can be built sequentially by fixing the current model $h_{M-1}(\mathbf{x})$ and updating the term $\beta_M b(\mathbf{x}, \boldsymbol{\gamma}_M)$ only.

Algorithm 2 Forward Stagewise Additive Model

```
  **Initialize** $h(\mathbf{x}_0) = 0$;
  **for** $m = 1, 2, ..., M$ **do**
    Compute
        $\{\beta_m, b_m(\cdot), \gamma_m\} = \underset{\beta, b_m(\cdot), \boldsymbol{\gamma}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} l\{y_i;\, h_{m-1}(\mathbf{x}_i) + \beta\, b(\mathbf{x}_i, \boldsymbol{\gamma})\};$    (4)
    Update $h(\mathbf{x}) := h(\mathbf{x}) + \beta_m b_m(\mathbf{x}, \boldsymbol{\gamma}_m)$.
  **end for**
```

We are ready to prove the main conclusion as stated in Proposition 1.

Proposition 1. *AdaBoost is a forward stagewise additive model with the exponential loss.*

Proof. Recall that AdaBoost uses decision trees as the basis functions, i.e., $b_m(\mathbf{x}, \boldsymbol{\gamma}) = \mathcal{T}_m(\mathbf{x})$. Under the exponential loss, consider the main step (4) in Algorithm 2:

$$\begin{aligned}(\beta_m, \mathcal{T}_m) &= \underset{\beta, \mathcal{T}}{\operatorname{argmin}} \ \frac{1}{n}\sum_{i=1}^{n} \exp\left[-y_i\left\{h_{m-1}(\mathbf{x}_i) + \beta\mathcal{T}(\mathbf{x}_i)\right\}\right] \\ &= \underset{\beta, \mathcal{T}}{\operatorname{argmin}} \ \frac{1}{n}\sum_{i=1}^{n} w_i^{(m)} \cdot \exp\left\{-\beta y_i \mathcal{T}(\mathbf{x}_i)\right\}, \qquad (5)\end{aligned}$$

where the term

$$w_i^{(m)} := \exp\left\{-y_i h_{m-1}(\mathbf{x}_i)\right\} \qquad (6)$$

depends neither on β nor $\mathcal{T}$ and hence can be treated as weights.

Problem (5) can be solved in two steps. In the first step, we fix $\beta > 0$ and solve for $\mathcal{T}$. With a fixed $\beta > 0$, (5) equivalently becomes

$$\begin{aligned} & \min_{\mathcal{T}} \ e^{-\beta} \sum_{i:\ y_i = \mathcal{T}(\mathbf{x}_i)} w_i^{(m)} + e^{\beta} \sum_{i:\ y_i \neq \mathcal{T}(\mathbf{x}_i)} w_i^{(m)} \\ \iff & \min_{\mathcal{T}} \ (e^{\beta} - e^{-\beta}) \sum_{i:\ y_i \neq \mathcal{T}(\mathbf{x}_i)} w_i^{(m)} + e^{-\beta}\sum_{i=1}^{n} w_i^{(m)} \\ \iff & \min_{\mathcal{T}} \ \sum_{i:\ y_i \neq \mathcal{T}(\mathbf{x}_i)} w_i^{(m)} \quad \text{after removing terms irrelevant to } \mathcal{T}. \\ \iff & \min_{\mathcal{T}} \ \sum_{i=1}^{n} w_i^{(m)} I\{y_i \neq \mathcal{T}(\mathbf{x}_i)\}. \end{aligned}$$

In other words, AdaBoost seeks the decision tree $\mathcal{T}$ that minimizes the weighted error rate. The solution is denoted as $\mathcal{T}_m$. Let ϵ_m be the minimized error rate,

$$\epsilon_m := \frac{\sum_{i=1}^{n} w_i^{(m)} I\{y_i \neq \mathcal{T}(\mathbf{x}_i)\}}{\sum_{i=1}^{n} w_i^{(m)}}.$$

Now we come to the second step: solving for β in (5) fixing $\mathcal{T}_m$. Simply taking the derivative of the objective function with respect to β and setting it to 0 yields

$$\begin{aligned} & \frac{1}{n}\sum_{i=1}^{n} w_i^{(m)} \cdot \exp\left\{-\beta y_i \mathcal{T}(\mathbf{x}_i)\right\}\left\{-y_i \mathcal{T}(\mathbf{x}_i)\right\} \overset{set}{=} 0 \\ \Longrightarrow \ & \sum_{y_i = \mathcal{T}(\mathbf{x}_i)} w_i^{(m)} e^{-\beta} = \sum_{y_i \neq \mathcal{T}(\mathbf{x}_i)} w_i^{(m)} e^{-\beta} \\ \Longrightarrow \ & (1 - \epsilon_m) = \epsilon_m e^{2\beta} \\ \Longrightarrow \ & \beta_m = \frac{1}{2}\ln\frac{1-\epsilon_m}{\epsilon_m}, \end{aligned}$$

which is exactly the same as used in AdaBoost Algorithm 1.

Finally, check the updated weight $w_i^{(m+1)}$ for the i-th observation. By Equation (6),

$$w_i^{(m+1)} := \exp\{-y_i h_m(\mathbf{x}_i)\} = \exp\left[-y_i \left\{h_{m-1}(\mathbf{x}_i) + \beta_m \mathcal{T}_m(\mathbf{x}_i)\right\}\right] = w_i^{(m)} \exp\left\{-y_i \beta_m \mathcal{T}_m(\mathbf{x}_i)\right\},$$

which also matches exactly with the weight w_i in Algorithm 1. $\square$

There is a slight discrepancy between the statistical explanation and AdaBoost though. From the proof of Proposition 1, we should seek a tree $\mathcal{T}_m$ at each stage that minimizes the weighted misclassification error rate $\sum_{i=1}^{n} w_i^{(m)} I\{y_i \neq \mathcal{T}(\mathbf{x}_i)\}$. In this spirit, a large tree seems appealing. However, in fact, AdaBoost benefits from simple tree models. The tree size parameter J is an important tuning parameter. When $J = 2$ or tree depth is 1, the AdaBoost model is purely additive in modeling the effects of X_j's'; when $J = 4$ with tree depth 2, AdaBoost would pick up first-order interactions among X_j's; and so on. At the mth stage, one could have apply a different J_m tree size, which is not advisable though since it would be hard to determine these tuning parameters. Empirical experiences (Hastie, Tibshirani, and Friedman, 2008) recommend fixing $J \leq 6$.

3 Arcing

The results from Breiman (1998), showing that boosting can be interpreted as a functional gradient descent algorithm in function space, uncovering older roots of boosting. He proposed the Arcing (adaptive resampling and combining) (Arc-×4; Breiman, 1998) as a simplified version of the AdaBoost. Arcing gives similar performance to AdaBoost. Unlike bagging, pruning the individual trees and selecting the optimally-sized tree improves performance (Bauer and Kohavi, 1999) in boosting.

At the m-th step, a model (decision tree) is fitted using weights for each case. For the i-th case the Arc-×4 weights are

$$w_i := \frac{1 + c_i^4}{\sum_{i=1}^{n}(1 + c_i^4)},$$

where $0 \leq c_i \leq m$ is the number of times that that ith case is misclassified in the preceding steps. A simple illustrative example is given in Table 1. For example, to obtain entries w_1 for $m = 2$, we have $2/8 \times 6 = 1.5$, $1/8 \times 6 = 0.75$, and so on.

Again, there are two ways of using the weights w_i here, i.e., either by using a weighted analysis or by resampling the data such that the probability that the ith case is selected is w_i. For convenience, the weights can be normalized to frequencies by multiplying by the sample size n, i.e., $w_i := n \cdot w_i$ (as shown in Table 1). Bauer and Kohavi (1999) found that resampling performed better than reweighting for arc-×4 but did not change the performance of AdaBoost. AdaBoost uses a different (more complicated) formula for w_i. Both formulas put greater weight on cases that are frequently misclassified.

The process is repeated M times and the M models are combined by voting or averaging the posterior probabilities. AdaBoost used weighted voting where models with fewer misclassifications, particularly of the hard-to-classify cases, are given more weight. Breiman (1998) used $M = 50$ and Bauer and Kohavi (1999) used $M = 25$. Breiman called Boosting an "equalizer" in the sense that when m increases, quantities in the c (misclassified) column (a weighted version) in Table 1 would become essentially similar. Arcing improves performance to a greater degree than bagging, but the

Table 1: A Simple Arcing Example Done by Hand

	$m = 1$			$m = 2$		$m = 3$		$m = 4$
id	w_i	c_i	$1 + c_i^4$	w_i	c_i	w_i	c_i	w_i
1	1	1	2	1.5	1	0.5	2	0.97
2	1	0	1	.75	0	.25	0	.06
3	1	1	2	1.5	2	4.25	3	4.69
4	1	0	1	.75	1	.5	1	.11
5	1	0	1	.75	0	.25	0	.06
6	1	0	1	.75	0	.25	1	.11
Total	$n = 6$		8	$n = 6$		$n = 6$		$n = 6$

improvement is not consistent (Breiman, 1998; Bauer and Kohavi, 1999). An implementation of Arcing is available in SAS Enterprise Miner.

4 Gradient Boosting

Following the optimization perspective of Breiman (1998) and the statistical view of AdaBoost, Friedman, Hastie, and Tibshirani (2000) and Friedman (2001) developed a more general, statistical framework termed 'gradient boosting', which yields a direct interpretation of boosting as a method for function estimation with the "stagewise, additive modeling" approach.

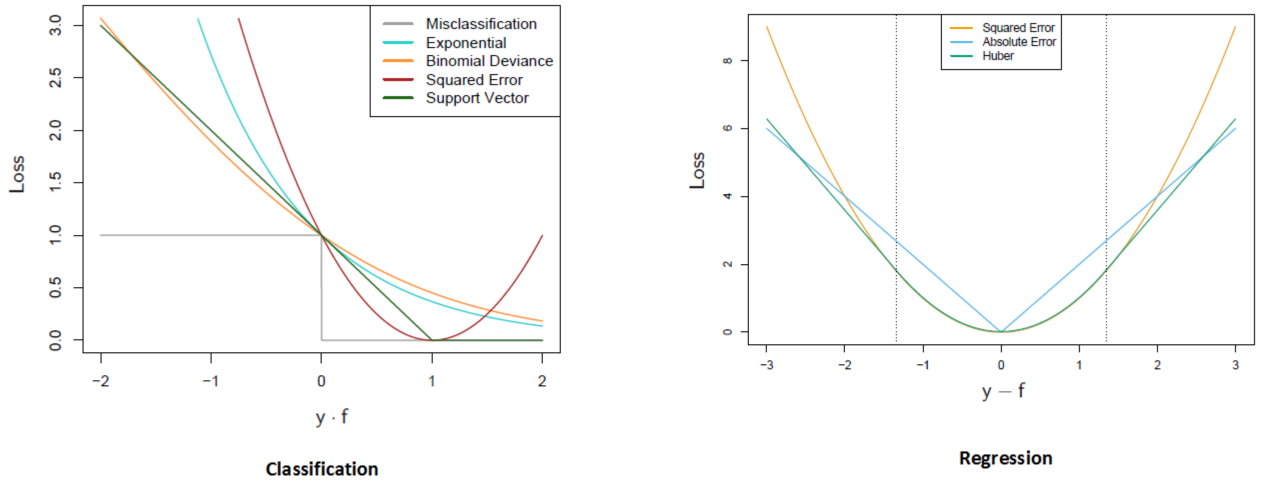

Figure 4: Losses Functions for Classification and Regression.

Gradient Boosting can be understood in the traditional way of minimizing loss function with some specified form of models. Different loss functions can be integrated in this general framework.

The main idea is outlined briefly below. Consider the regression or discriminant function of additive model

$$h_M(\mathbf{x}) \;=\; \sum_{m=1}^{M} b_m(\mathbf{x}, \gamma_m) \;=\; h_{M-1}(\mathbf{x}) + b_M(\mathbf{x}, \gamma_M). \tag{7}$$

Compared to the additive model (3), the coefficient β_m has been absorbed into the basis function $b_m(\mathbf{x}, \gamma_m)$. Nevertheless, a tuning parameter called the *learning rate* such that $\beta_m \equiv \beta$ for $m = 1, \ldots, M$ will be applied for the regularization purpose later on. The basis function $b_m(\mathbf{x}, \gamma_m)$ represents a "weak base learner" such as tree predictions. Boosting estimates $h_M(\mathbf{x})$ via update with *forward stagewise* steps.

Let $l\{y; h(\mathbf{x})\}$ be some loss function that measures the performance of prediction or fitted value $h(\mathbf{x})$ with respect to y. We estimate $h(\mathbf{x})$ by optimizing the empirical risk function

$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} l\{y_i; h(\mathbf{x}_i)\} = \frac{1}{n}\sum_{i=1}^{n} l_i.$$

One standard method of optimization is gradient descent. Specifically, the gradient

$$\mathbf{g}_m := \left.\frac{\partial \hat{R}(h)}{\partial h}\right|_{h=h_{m-1}} = (g_{mi}) \in \mathbb{R}^n$$

is n-dimensional since each of its components is evaluated at each observation $h(\mathbf{x}_i)$ for $i = 1, \ldots, n$ such that $g_{mi} = dl_i/dh(\mathbf{x}_i)$ avaluated at $h(\cdot) = h_{m-1}(\cdot)$. The gradient $\mathbf{g}_m$ turns out to be some kind of residuals, termed as generalized or pseudo residuals. For example, if the square loss $l(y, h) = (y - h(\mathbf{x}))^2/2$ is used, then $-\partial l/\partial h = y - h(\mathbf{x})$.

Algorithm 3 Gradient Boosting Algorithm

```
Initialize $h_0(\mathbf{x}) = 0$ and set $J, M \in \mathbb{N}$ and learning rate $0 < \beta < 1$.;
for $m = 1, 2, ..., M$ do
  for $i = 1, 2, ..., n$ do
    Compute and evaluate
        $g_{mi} = \left.\frac{\partial\, l\{y_i, h(\mathbf{x}_i)\}}{\partial\, h(\mathbf{x}_i)}\right|_{h=h_{m-1}}$
  end for
  Train a regression tree structure $\mathcal{T}_m$ of size $J$ with target variables $-g_{mi}$.
  Let $\mathbf{T}_m$ denote the dummy variable matrix induced by $\mathcal{T}_m$.
  Estimate the associated coefficients $\boldsymbol{\gamma}_m$ in $\mathcal{T}_m$ by solving $\min_{\boldsymbol{\gamma}} \hat{R}(h_{m-1} + \mathbf{T}_m\gamma)$.
  Update $h_m(\mathbf{x}) := h_{m-1}(\mathbf{x}) + \beta\,\mathbf{T}_m\boldsymbol{\gamma}_m$;
end for
Output $\hat{h}(\mathbf{x}) = h_M(\mathbf{x})$.
```

The key idea of 'gradient boosting' is to approximate the (negative) gradient $d\hat{R}/dh$ with a weak base learner $b(\mathbf{x}, \boldsymbol{\gamma})$ and proceed in a forward stagewise manner. The procedure is outlined in Algorithm 3. Gradient boosting inherently relies on a gradient descent search for optimizing the underlying loss function to determine both the weights and the learner at each iteration. The regression tree $\mathcal{T}_m$ in Algorithm 3 approximates the negative gradient with least squares. Following this general idea, many other boosting variants (e.g., real boost, logit boost, gentle boost) were developed by using different types of loss functions (see Figure 4) for different types of responses. Among these developments, *Stochastic Gradient Boosting* (SGB; Friedman, 2002) employs an additional random subsampling (with replacement) strategy at each iteration to construct $\mathcal{T}_m$. This helps not only with computational speed but also with decorrelation of the base learners.

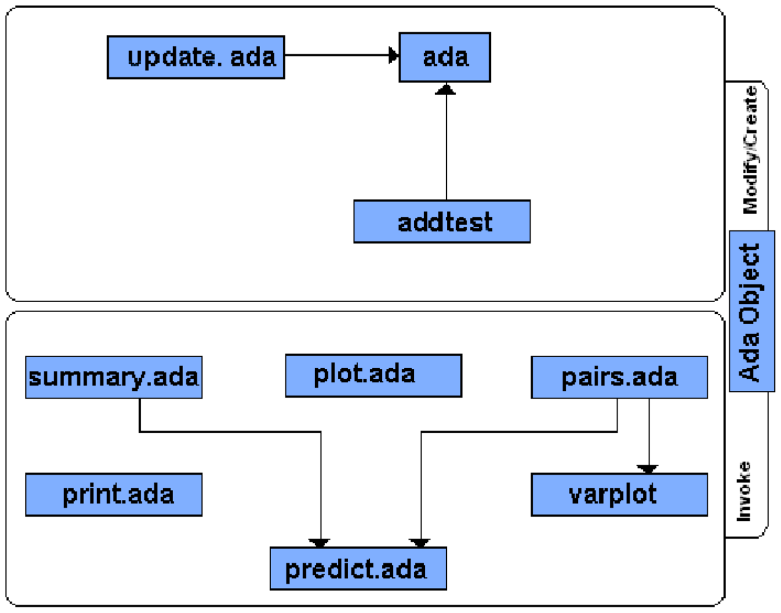

Figure 5: Function Flow in R Package `ada`. The top section consists of the functions used to create the `ada` object, while listed in the bottom section are the functions invoked using an initialized `ada` object.

5 R Implementation

There are several R packages that implement boosting, including

1. `ada` — The `ada` package implements the original AdaBoost algorithm, along with the Gentle and Real AdaBoost variants, using both exponential and logistic loss functions for classification problems. In addition, it allows the user to implement regularized versions of these methods by using the learning rate as a tuning parameter, which leads to improved computational performance. The base classifiers employed are classification/regression trees and therefore the underlying engine is again the `rpart` package. A copy of the function flow of `ada` is given in Figure 5.

2. `gbm` — The `gbm` package offers two versions of boosting for classification (gentle boost under logistic and exponential loss). In addition, it includes squared error, absolute error, Poisson and Cox type loss functions.

3. `mboost` — The `mboost` package has to a large extent similar functionality to the `gbm` package and in addition implements the general gradient boosting framework using regression based learners.

4. `boost` - Implementation of boosting specially for gene expression data.

References

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Machine Learning*, **36**: 105–139.

Breiman, L. (1998). Arcing classifiers (with discussion). *The Annals of Statistics*, **26**: 801-?849.

Breiman, L. (1999). Prediction games & arcing algorithms. *Neural Computation*, **11**: 1493?-1517.

Bühlmann, P. and Hothorn, T. (2007). Boosting Algorithms: Regularization, Prediction and Model Fitting. *Statistical Science*, **22**(4): 477–505.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2008). *Elements of Statistical Learning*, 2nd Edition. Chapman and Hall.

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression (with discussion). *The Annals of Statistics*, **32**: 407–451.

Freund, Y. and Schapire, R. (1995). A decision-theoretic generalization of online learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory, Lecture Notes in Computer Science*, Springer.

Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA.

Freund, Y. and Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, **55**: 119–139.

Friedman, J. (2001). Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, **29**: 1189–1232.

Friedman, J. H. (2002) Stochastic gradient boosting. *Computational Statistics & Data Analysis*, **38**: 367–378.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion). *The Annals of Statistics*, **28**: 337–407.