Why does boosting work from a statistical view

Jialin Yi
Applied Mathematics and Computational Science
University of Pennsylvania
Philadelphia, PA 19139
`jialinyi@sas.upenn.edu`

Abstract

We review boosting methods from a statistical perspective. Boosting can be integrated into statistical decision theory with appropriate loss function, e.g. exponential loss for Adaboost. We demonstrate that Adaboost is the forward stepwise additive nonparametric estimator for the Bayesian classifier. Based on this perspective, a more robust boosting algorithm called "LogitBoost" is proposed, which optimizes the cross entropy risk. Finally, we present the consistency of boosting method and how a optimal stopping should be chosen.

1 Introduction

In this project, we discuss the statistical foundation of boosting which is one of the most important ideas in machine learning and statistical learning area during 90's. The boosting methods are motivated to ensemble a set of weak classifiers to form a strong classifier in PAC learning framework. A breakthrough in boosting methods is the invention of "Adaboost" algorithm that works on finite training data and has controllable computational complexity because of its nature as a iterative algorithm [4]. The huge success of Adaboost in so many areas is beyond what PAC framework could explain. After ten years of efforts from statistics community and machine learning community, the convergence and consistency of Adaboost have been established. Based on these understanding, many more powerful boosting algorithms have been developed and the applications of boosting methods have been enlarged to regression problem [5].

For statisticians, it becomes common to interpret boosting as minimizing the empirical risk for a asymmetric loss function that penalizes more heavily on the incorrect classifications. And different functional gradient descent algorithms specify different boosting methods. Table 1 shows the loss function for common boosting algorithms.

Boosting and decision theory		
Algorithms	Loss function	Gradient
Adaboost	$\exp(-Yf(X))$	$-Y\exp(-Yf(X))$
Gradient Boosting	differential function $L(Y, f(X))$	$\partial L/\partial f$
MarginBoost	$\psi(-Yf(X))$, ψ is convex	$-Y\psi'(-Yf(X))$
LogitBoost	$\log(1+e^{-Yf(X)})$	$-Ye^{-Yf(X)}/(1+e^{-Yf(X)})$

Table 1: Loss functions and gradients of common boosting algorithms

Section 2 gives the details of two most common boosting methods for classification and regression respectively—Adaboost and gradient boosting. And we give an example on how gradient boosting could improve the performance of ordinary least square on nonlinear data to see the power of boosting. The dramatic power of boosting lies in its expansion form which is an additive model in statistics.

Section 3 introduces statistical decision theory and its notions. Supervised learning problem could be phased as the parametric/nonparametric estimation with specific loss function and penalty. For classification, we show that the best theoretical classifier with respect to exponential loss is the Bayesian classifier. And in square loss, the optimal prediction for square loss is the conditional expectation. Many boosting methods, including Adaboost and gradient boosting, are actually the *forward stepwise fitting* solution to minimize the empirical risk function. We give a proof for Adaboost in Section 4.

Even though minimizing the empirical risk is consistent by VC theory, the consistent of forward stepwise fitting is not promised. Also, in practice we need to determine the optimal number of iterations so that we could study the computational complexity of boosting method. These issues are discussed in Section 5.

2 Boosting and Additive model

The initial motivation of boosting is to strengthen the weak learning algorithm under Probabilistic Approximating Correct (PAC) learning framework [9]. That is to transform a weak classifier which is slightly better than random guessing into the algorithm that could achieve arbitrarily small accuracy with arbitrarily high confidence level if given (infinite) access to the oracle.

The first boosting procedure was given by Schapire (1990) and this procedure also proved the equivalence of the weak PAC learnability and strong PAC learnability for classification [17]. The idea was to apply the weak classifier to filtered data repeatedly and force the learning algorithm to learn heavily on those incorrectly classified sample data.

For two-class problem, after learning the initial weak classifier $G_0(x)$ on the first N training data,

- $G_1(x)$ is learned from N new data points, half of which are misclassified by $G_0(x)$
- $G_2(x)$ is learned from another N new training data on which $G_1(x)$ and $G_0(x)$ disagree.
- the boosted classifier $G_3(x) = MajorityVote(G_0(x), G_1(x), G_2(x))$

However, the above procedure is not practical for the supervise learning in industry. First, it requires potentially infinite training data to produce the filtered data for $G_1(x)$ and $G_2(x)$. But in industry, we often have a fixed number of training data. Besides, the algorithm is not computationally efficient.

Nevertheless, the idea in the initial boosting procedure is invaluable and gives birth to a practical boosting algorithm that makes a difference in the real world — Adaboost.

2.1 Adaboost

Adaboost is the first practical boosting algorithm, developed by Freund and Schapire (1996), and achieves huge success in a wide range of applications [4]. The Adaboost algorithm we describe here is for the binary classification problem due to Freund and Schapire (1997).

We will use the statisticians' convention that X_i is the sample or the training data and X for random variable. Suppose $X \in \Pi$ and $Y \in \Omega$ are predictor space and response space. In binary classification, the response variable $Y \in \{-1, 1\}$ and $X \in \mathbb{R}^d$ is the predictor vector. A classifier is a function $G : \mathbb{R}^d \to \{-1, 1\}$. And a weak classifier satisfies $\mathbf{E}[\mathbb{1}(G(X) \neq Y)] < 1/2$ where the expectation is taken over the joint distribution of (X, Y). The error rate of a classifier $G_m(x)$ is

$$err_m = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(G_m(X_i) \neq Y_i)$$

for $m = 1, 2, \cdots, M$. For Adaboost, we only consider the classifiers

$$G(x) = sgn(F(x)) \tag{1}$$

where $F(x)$ is a real function standing for the classifer boundary.

In Adaboost, the weak learning algorithm trains M classifiers using the filtered sample data in each iteration and then the weighted sum of all M classifiers is the strong classfier we want. The ultimate classifier boundary takes form

$$F(x) = \sum_{m=1}^{M} \alpha_m F_m(x)$$

where $F_m(x)$ is the corresponding classifier boundary curve of $G_m(x)$. The classifier weight α_m is computed as in Algorithm 1.

Algorithm 1 Adaboost for binary classification

```
Input: Sample $\{(X_i, Y_i)\}_{i=1}^N$, iteration rounds $M$, weak learning algorithm $WeakLearn$
Initialize the sample weights $w_i = 1/N, i = 1, 2, \cdots, N$
for $m = 1$ to $M$ do
    fitting the classifier boundary curve $F_m(x)$ using $WeakLearn$ on training data with weights $\{w_i\}_{i=1}^N$
    Compute
        $$err_m = \frac{\sum_{i=1}^N w_i I(Y_i \neq F_m(X_i))}{\sum_{i=1}^N w_i}$$
    Compute $\alpha_m = \log[(1 - err_m)/err_m]$
    Update the weights $w_i \leftarrow w_i \exp[\alpha_m I(Y_i \neq F_m(X_i))]$ for $i = 1, 2, \cdots, N$.
end for
Output $G(x) = sgn(F(x))$ where $F(x) = \sum_{m=1}^M \alpha_m F_m(x)$.
```

Algorithm 1 gives details of Adaboost algorithm. The current classifier $G_m(x)$ and classifier boundary curve are trained using sample data with weights $\{w_i, i = 1, 2, \cdots, N\}$. The classifier weight α_m of $F_m(x)$ is a decreasing logarithmic transformation of the error rate of classifier $G_m(x)$. And then the weights for sample points that are incorrectly classified by $G_m(x)$ are scaled by a factor of $\exp(\alpha_m)$ for the next iteration.

2.2 Gradient Boosting

Although Adaboost improves learning algorithms dramatically, the application is restricted in classification. However, the idea of boosting and ensemble could be applied to much broader area.

Leo Breiman (1997) first observed that boosting could be viewed as a solution of an optimization problem given the appropriate loss function [1]. Explicit regression boosting algorithm was first developed by Friedman (2001) [7] simultaneously with the more general functional gradient boosting perspective of Llew Mason, Jonathan Baxter, Peter Bartlett and Marcus Frean [14] which will be discussed extensively in Section 3.

In regression, we will define the general loss function as $L : \mathbb{R}^d \times \Omega \to [0, +\infty)$ instead of the 0-1 loss $\mathbb{1}(Y_i \neq f(X_i))$ for classification.

Algorithm 2 gives the regression boosting for the differential loss function, called ”gradient boosting” by Friedman. At each iteration, we compute the *pseudo-residuals* which is the gradient of the loss function with respect to the prediction function. Then we choose the basis prediction closest to the gradient. Note that this approach is similar to the gradient descent which is very greedy. However this approximation is good enough for practical use. The consistency of this method will be discussed in Section 6.

2.3 An example: gradient boosted least square

Consider a data generating process

$$Y = X_1^{3/2} + 2\sqrt{X_2} + \mu + \epsilon$$

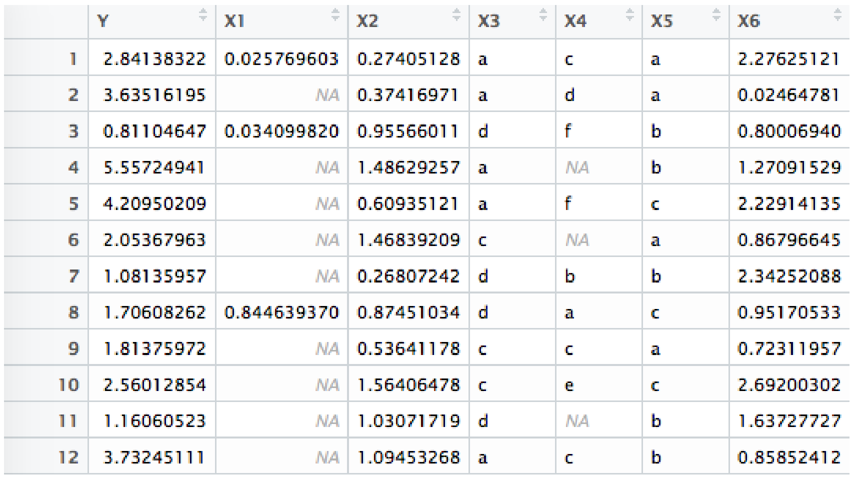
where X_1 and X_2 are uniformed distributed on $[0, 1]$ and $[0, 2]$, μ is drawn from $(-1, 0, 1, 2)$ with replacement which cannot be seen from the training data, and ϵ is Gaussian white noise such that the *signal-to-noise ratio* is 10.

Algorithm 2 Gradient Boosting for differential loss function

```
Input: Sample $\{(X_i, Y_i)\}_{i=1}^{N}$, iteration rounds $M$, collection $\mathcal{F}$ of basis functions
Initialize prediction function with a constant value
    $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(Y_i, \gamma)$
for $m = 1$ to $M$ do
    compute pseudo-residuals for $i = 1, 2, \cdots, N$
        $r_i^m = -\frac{\partial L(Y_i, f_{m-1}(X_i))}{\partial f_{m-1}(X_i)}$
    fitting a prediction function $g_m(x) \in \mathcal{F}$ to the pseudo-residuals, i.e. using the training data $\{(X_i, r_i^m) : i = 1, 2, \cdots, N\}$.
    solve the one-dimensional optimization problem
        $\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{N} L(Y_i, f_{m-1}(X_i) + \gamma g_m(X_i))$
    Updating the functinon $f_m(x) = f_{m-1}(x) + \gamma_m g_m(x)$
end for
Output $f_M(x)$
```

Besides X_1 and X_2, in training data there are category variables X_3, X_4 and X_5 that expose some information about Y. Also, there is some unrelated variable X_6 following uniform distribution on [0,3]. And there is randomly assigned missing data in the training samples.

This example is due to [15] and Figure 1 shows the data frame.



	Y	X1	X2	X3	X4	X5	X6
1	2.84138322	0.025769603	0.27405128	a	c	a	2.27625121
2	3.63516195	NA	0.37416971	a	d	a	0.02464781
3	0.81104647	0.034099820	0.95566011	d	f	b	0.80006940
4	5.55724941	NA	1.48629257	a	NA	b	1.27091529
5	4.20950209	NA	0.60935121	a	f	c	2.22914135
6	2.05367963	NA	1.46839209	c	NA	a	0.86796645
7	1.08135957	NA	0.26807242	d	b	b	2.34252088
8	1.70608262	0.844639370	0.87451034	d	a	c	0.95170533
9	1.81375972	NA	0.53641178	c	c	a	0.72311957
10	2.56012854	NA	1.56406478	c	e	c	2.69200302
11	1.16060523	NA	1.03071719	d	NA	b	1.63727727
12	3.73245111	NA	1.09453268	a	c	b	0.85852412

Figure 1: Data frame for the simulated data

In this example, the basis prediction function is

$$\hat{Y} = \sum_{i=1}^{6} \beta_i X_i$$

Since the data generating process is nonlinear, least square method is not expected to give good prediction on Y. However, gradient boosting would reduce the prediction error to one third of its original value.

The model is implemented with *gbm* package in R (ver. 3.4.0) and the code is seen in Appendix. Figure 2 shows that as the number of boosting iterations increases, the training error (black curve) decreases more than 2/3 of its original value and also the test/validation error (red curve) first decreases like the training error before iteration 129 and then increases slowly. The optimal number of iterations will be discussed in Section 5.

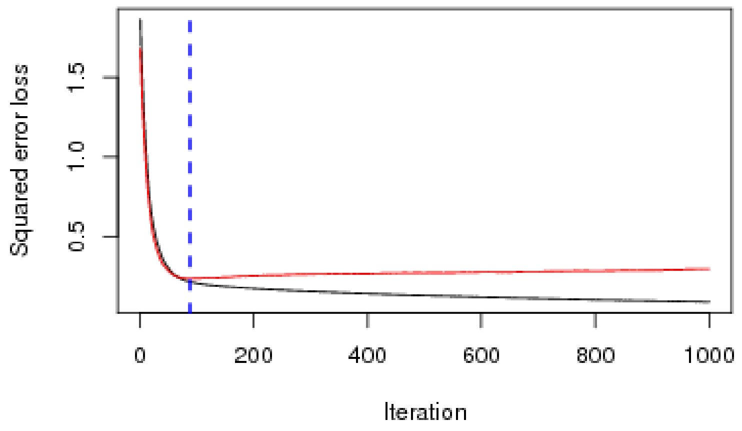


Figure 2: Performance of gradient boosted least square model. The black curve is the training error and the red curve is the square error on the validation set. The optimal boosting iteration (blue dashed line) is 129.

2.4 Additive model

The huge success of boosting is not as mysterious as it used to be when it was born. The key lies in the form of the output. Boosting can be viewed as an example for the *ensemble* method which obtains better learning performance by combining multiple learning algorithms. In Algorithm 1, Adaboost ensembles multiple classifiers generated by one learning algorithm $Weaklearn$ on data with different weights. The marginal boundary of boosted algorithm is a weighted sum of a collection of margins in the weak classifiers. In Algorithm 2, gradient boosting applies the basis prediction algorithm repeatedly to approximate the residuals in last iteration and chooses the weights of regression models so that the empirical risk is minimized in each iteration.

For statisticians, boosting is an additive model takes form

$$f(x) = \sum_{m=1}^{M} \beta_m f_m(x) \tag{2}$$

where $f_m(x) \in \mathcal{F}$, $\mathcal{F}$ is a collection of the basis classifier. β_m is the weight parameter for each prediction model. To determine β_m and f_m, we need statistical decision theory.

3 Statistical decision theory

In this section, we discuss how to solve the classification and regression from a statistical perspective. In most cases, Π is Euclidean space $\mathbb{R}^d$ or its subset so we let $\Pi = \mathbb{R}^d$ without loss of generality. $(X_i, Y_i) \sim i.i.d.(X, Y)$ for $i = 1, 2, \cdots, N$ are samples. $\mathcal{G} = \cup_{n=1}^{\infty} \mathcal{G}_n$ is the set of all linear combinations of the basis learners where $\mathcal{G}_n = \{\sum_{i=1}^{n} \beta_i f_i(x) : f_i(x) \in \mathcal{F}\}$.

To evaluate an estimate $f : \mathbb{R}^d \to \Omega$, there are two common criteria, one is the *risk* that is expected loss $r(f) = \mathbf{E}[L(Y, f(X))]$ and another is *max risk* $\max_{X \in \mathbb{R}^d, Y \in \Omega} L(Y, f(X))$. For boosting, we show that boosted estimate is consistent for expected risk.

Many machine learning problems can be phased in statistical decision framework with

- A collection $\mathcal{F}$ of basis functions for representing the target function as a linear combination or convex combination $f(x) = \sum_{i=1}^{N} \beta_i f_i(x)$ where $f_i \in \mathcal{F}$ for all i.
- A loss function $L(Y, f(X))$ appropriate for the problem, for example $(Y - f(X))^2$ for regression and $\mathbb{1}(Y \neq f(X))$ for classification.
- A penalty/regularizer $J(\beta)$ that controls the size of coefficients in the model.

The empirical risk is $\frac{1}{N} \sum_{i=1}^{N} L(Y_i, f(X_i))$, then the learning process is to solve the optimization problem

$$\begin{aligned} \underset{f}{\text{minimize}} \quad & \frac{1}{N} \sum_{i=1}^{N} L(Y_i, f(X_i)) + \lambda J(\beta) \\ \text{subject to} \quad & f(x) = \sum_{m=1}^{M} \beta_m f_m(x), \text{ where } f_m(x) \in \mathcal{F} \end{aligned} \tag{3}$$

If the loss function L constructed properly, the resulting problem is convex and hence can be solved by convex optimization algorithms:

- Support vector machine uses a hinge loss function and a basis collection generated by a positive definite kernel with L_2 penalty.
- CART uses a hinge loss for classification and square loss for regression, a collection that is linear combination of random trees with penalty for the size of the trees.
- Lasso uses a square loss and a collection of identity functions with L_1 penalty.

3.1 Bayesian classifier

In classification, $Y \in \Omega = \{-1, +1\}$. Suppose Y has nonzero support on -1 and 1, we define the *Bayesian classifier*

$$f(X) = \begin{cases} 1 & \text{if } \mathbf{P}(Y = 1|X) > \frac{1}{2} \\ -1 & \text{otherwise} \end{cases}$$

equivalently, $f(X) = sgn[\frac{1}{2} \log(\mathbf{P}(Y = 1|X)/\mathbf{P}(Y = -1|X))]$. We give the result that Bayesian classifier is the optimal classifier for *exponential* loss.

Lemma 3.1. *If Y has nonzero support on* $\{-1, 1\}$ *and* $f(X)$ *is the Bayesian classifier, then*

$$f(X) = \arg \min_{g(x) \in \{-1,1\}} \mathbf{E}\left[\exp(-Y g(X))\right] \tag{4}$$

The proof is given by Friedman (2000) and idea is to consider the condition expectation of the loss given X [5].

Proof. Note that the above optimization problem is equivalent to minimize the conditional expectation of the loss

$$\mathbf{E}\left[\exp(-Y g(X)) | X\right] = \exp(g(X))\mathbf{P}(Y = -1|X) + \exp(-g(X))\mathbf{P}(Y = 1|X)$$

Hence when $\mathbf{P}(Y = 1|X) > \mathbf{P}(Y = -1|X)$, let $g(X) = 1$ □

The exponential loss is an convex approximation to the 0-1 loss. For exponential loss, the loss function penalizes the incorrect classification with higher loss value.

For more generalized loss function $L(Y, f(X)) = \psi(Y f(X))$ where ψ is convex with $\psi(1) < \psi(-1)$, Zhang (2005) showed that Bayesian classifier was also the optimal classifier that minimized the risk $\mathbf{E}[\psi(Y f(X))]$ [18].

So to estimate the Bayesian classifier, the straight forward way is to minimize the empirical risk.

$$\hat{f} = \arg \min_{g \in \mathcal{G}} \frac{1}{N} \sum_{i=1}^{N} \exp(-Y_i g(X_i)) \tag{5}$$

and we call this estimator as *empiricial Bayesian classifier*.

3.2 Conditional expectation predictor

In regression, if $Y \in \mathbb{R}$ is L_2 random variable, then the conditional expectation is the optimal estimate for square loss.

Lemma 3.2. *Suppose $Y \in L_2$ then*

$$\mathbf{E}(Y|X) = \arg\min_{g \in \mathcal{G}} \mathbf{E}(Y - g(X))^2 \tag{6}$$

Consider the Hilbert space of all L_2 random variable with L_2 norm, then all random variables that are X-measurable forms a subspace. The norm of the residual is minimized when $g(X)$ is the projection of Y onto this subspace, which is also the conditional expectation of Y given X.

Hence for square loss, we use a functional least square method to estimate f

$$\hat{f} = \arg\min_{g \in \mathcal{G}} \frac{1}{N} \sum_{i=1}^{N} (Y_i - g(X_i))^2 \tag{7}$$

When $\mathcal{G}$ has finite VC dimension, $\hat{f}$'s in Equation (5) and Equation (7) are consistent in PAC framework [9]. However, when $\mathcal{F}$ has infinite cardinality, we cannot solve Equation (5) and Equation (7) trivally and we need the corresponding optimization algorithms.

4 Empirical risk, optimization and boosting

In Equation (5) and Equation (7), we need to optimize the empirical risk

$$\begin{aligned} \underset{f}{\text{minimize}} \quad & \frac{1}{N} \sum_{i=1}^{N} L(Y_i, f(X_i)) \\ \text{subject to} \quad & f(x) = \sum_{m=1}^{M} \beta_m f_m(x), \text{ where } f_m(x) \in \mathcal{F} \end{aligned} \tag{8}$$

For many loss function L and basis learners $\mathcal{F}$, direct method requires computation intensively numerical optimization techniques. Alternatively, we could use a greedy algorithm called *Forward stepwise fitting* [6].

4.1 Forward stepwise fitting

Algorithm 3 Forward stepwise fitting

```
Input: Sample $\{(X_i, Y_i)\}_{i=1}^N$, iteration rounds $M$, a collection of basis function $\mathcal{F}$.
Initialize $f_0(x) = 0$
for $m = 1$ to $M$ do
    fitting the stepwise weight and function
        $(\beta_m, f_m) = \arg\min_{\beta \in \Omega, f \in \mathcal{F}} \sum_{i=1}^{N} L(Y_i, f_{m-1}(X_i) + \beta f(X_i))$
end for
Output
    $f(x) = \sum_{m=1}^{M} \beta_m f_m(x)$
```

Algorithm 3 gives details of forward stepwise fitting. For each iteration, the weight and new weak learner is chosen to minimize the empirical loss of the additive model greedily.

To see this, consider square loss,

$$L(Y_i, f_{m-1}(X_i) + \beta f(X_i)) = (Y_i - f_{m-1}(X_i) - \beta f(X_i))^2$$

the term $\beta_m f_m(x)$ that best fits the residuals is added to the current expansion of the model, which is exactly what Adaboost and gradient boost do in each iteration. Here, we give a proof for Adaboost.

4.2 Deriving Adaboost

We show that Adaboost (Algorithm 1) is equivalent to forward stepwise fitting (Algorithm 3) for exponential loss

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

For Adaboost, the collection of basis function $\mathcal{F}$ is the set of all classifiers generated by weak leaning algorithm $WeakLearn$.

To add new term to the current expansion $f_{m-1}(x)$, one must solve:

$$\begin{aligned}(\beta_m, f_m) &= \arg \min_{\beta \in \Omega, f \in \mathcal{F}} \sum_{i=1}^{N} exp(-Y_i f_{m-1}(X_i) - Y_i \beta f(X_i)) \\ &= \arg \min_{\beta \in \Omega, f \in \mathcal{F}} \sum_{i=1}^{N} w_i^m exp(-Y_i \beta f(X_i))\end{aligned} \tag{9}$$

where $w_i^m = exp(-Y_i f_{m-1}(X_i))$. Then we split the summation into two groups—correct classified and misclassified

$$\begin{aligned}\sum_{i=1}^{N} w_i^m \exp(-Y_i \beta f(X_i)) &= e^{-\beta} \sum_{i=1}^{N} w_i^m \mathbb{1}(Y_i = f(X_i)) + e^{\beta} \sum_{i=1}^{N} w_i^m \mathbb{1}(Y_i \neq f(X_i)) \\ &= e^{-\beta} \sum_{i=1}^{N} w_i^m + (e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} w_i^m \mathbb{1}(Y_i \neq f(X_i))\end{aligned}$$

therefore any $err_m = \sum_{i=1}^{N} w_i^m \mathbb{1}(Y_i \neq f(X_i)) / \sum_{i=1}^{N} w_i^m$, the solution of Equation 9 has

$$\beta_m = \frac{1}{2} \log \frac{1 - err_m}{err_m}$$

hence to minimize the loss, f must be the classifier generated by $WeakLearn$ using the weights w_i^m

$$f_m(x) = \arg \min_{\mathcal{F}} \frac{\sum_{i=1}^{N} w_i I(Y_i \neq F_m(X_i))}{\sum_{i=1}^{N} w_i}$$

To update the weights, note that

$$\begin{aligned}w_i^{m+1} &= \exp(-Y_i f_m(X_i)) \\ &= w_i^m \exp(-\beta_m Y_i f_m(X_i)) \\ &= w_i^m \exp[\beta_m (2\mathbb{1}(Y_i \neq f_m(X_i)) - 1)] \\ &= w_i^m e^{-\beta_m} \exp[\alpha_m \mathbb{1}(Y_i \neq f_m(X_i)))\end{aligned}$$

where $\alpha_m = 2\beta_m$ is defined as in Algorithm 1. The factor $e^{-\beta_m}$ scales for all terms so it does not affect the weights.

Hence the output of Adaboost (Algorithm 1) is a forward stepwise fitting solution to optimize the empirical risk for the exponential loss, which approximates the Bayesian classifier.

$$\boxed{\text{Bayesian classifier}} \rightarrow \boxed{\text{Empiricial Bayesian classifier}} \rightarrow \boxed{\text{Adaboosted classifier}}$$

Besides, under this framework, Adaboost is not trying to minimize the incorrect classification rate. Instead, Adaboost struggles to minimize the empirical exponential loss function along the iterations. By this method, Adaboost gives a good nonparametric estimator for the Bayesian classifier. This explains well the relative immunity of Adaboost to overfitting in many scenarios.

Remark. *For statisticians, Adaboosted classifier is a forward stepwise fitting nonparametric estimator for the Bayesian classifier. Even though the consistency of empirical Bayesian classifier is promised by VC theory, the consistency of Adaboosted classifier does not follow automatically. Instead, it needs further analysis about the optimization procedure and the basis function collection $\mathcal{F}$ and the related results are in Section 5.*

4.3 Robustness and cross entropy

From the previous discussion, we know that Adaboost is minimizing the empirical exponential loss. The reason is that the exponential loss leads to the Bayesian classifier. This makes perfect sense for the prediction accurancy by Lemma 3.1. Also note that by changing the loss function, the classifier becomes different and this could improve the robustness of Adaboost, see Figure 3 for common loss function for classification.

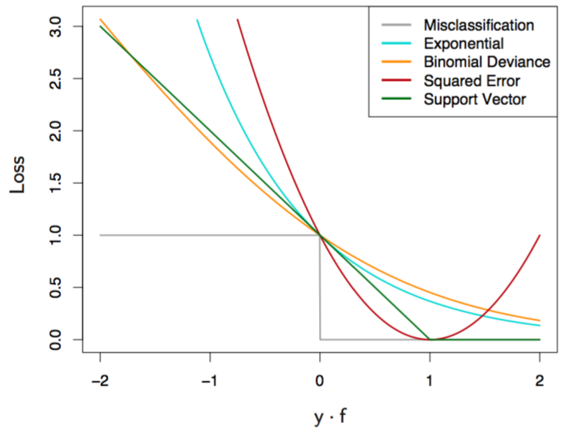


Figure 3: Loss function for binary classification. The figure comes from *Elements of Statistical Learning, 2ed* [6]. $y \in \{-1, 1\}$ and f is the predicted value. misclassification: 0-1 loss; exponential: $\exp[-yf]$; binomial deviance: $\log[1+\exp(-2yf)]$; square error: $(y-f)^2$; support vector: $(1-yf)_+$

Even though exponential loss gives accurate prediction, it is also more sensitive to the outliers and misclassfications, as in Figure 3. Hence here we give a more robust loss function that is better-known in information theory community, called *cross entropy* or *deviance*.

Let the conditional probability of $Y = 1$ when $X = x$ is $p(x)$. Define a new probability distribution

$$q(x) = \frac{Y+1}{2}$$

is the probability of $Y = 1$ when $X = x$. Then the cross entropy

$$H(q(x), p(x)) = \mathbf{E}_{q(x)}[-\log p(x)] \tag{10}$$

which describe the distance between two probability distribution. The physical intuition is the average number of bits needed to identify an event drawn from the set, if a coding scheme is used that is optimized for an "unnatural" probability distribution p, rather than the "true" distribution q.

Note that q is true distribution of Y, hence $H(q(x), p(x))$ measures how close the conditional probability is to the true distribution of Y.

$$\begin{aligned} H(q,p) &= -\frac{Y+1}{2}\log p(x) - \left(1 - \frac{Y+1}{2}\right)\log(1-p(x)) \\ &= \log\left[1+\exp(-2Yf(x))\right] \end{aligned}$$

where

$$p(x) = \mathbf{P}(Y=1|X=x) = \frac{1}{1+\exp[-2f(x)]}$$

Hence we have a new criteria for choosing the the nonparametric estimator—minimizing the empirical cross entropy

$$\begin{aligned} \underset{f}{\text{minimize}} \quad & \frac{1}{N}\sum_{i=1}^{N} \log\left[1+\exp(-2Y_i f(X_i))\right] \\ \text{subject to} \quad & f(x) = \sum_{m=1}^{M} \beta_m f_m(x), \text{ where } f_m(x) \in \mathcal{F} \end{aligned} \tag{11}$$

Also, we could solve the above optimization by forward stepwise fitting by letting $L(Y, f(X)) = \log\left[1+\exp(-2Yf(X))\right]$. Note that $\log(1+x)$ is a increasing function, then the solution of Equation 11 is also the solution of Equation 4 but cross-entropy is more robust than exponential loss as in Figure 3.

Cross entropy was developed by Rubinstein (1997) for estimating the probability of rare events in complex stochastic network [16] and was soon applied to a much more wider range of area. For a more detailed discussion of the history and applications of cross entropy, see [3]. Boosting with cross entropy loss is called ”LogitBoost”, see [5] for details.

5 Adaboost with early stopping

Ideally, the optimal boosting iteration is chosen to minimize the test error. With finite training data, we cannot get test error. Nevertheless, we could estimate the test error by *cross validation* [10].

Here we present the early stopping technique for Adaboost in Algorithm 4.

Algorithm 4 Adaboost with early stopping

```
Input: Sample $\{(X_i, Y_i)\}_{i=1}^N$, iteration rounds $M$, a collection of basis function $\mathcal{F}$.
Initialize $m = 0, F_0(x) = 0$ and $CV_0 = +\infty$
Divide the data set into $K$ folds—1st for training set and the other $K-1$ folds for validation.
repeat
    $m = m + 1$
    fitting the Adaboost classifier boundary $F_m(x)$ at iteration $m$ on training set.
    compute the validation error $err_j$ of $F_m(x)$ on $j$th fold, for $j = 2, \cdots, K$
    compute the cross validation
        $CV_m = \frac{\sum_{j=2}^K err_j}{K-1}$
until $CV_m > CV_{m-1}$
Output $G(x) = sgn(F_{m-1}(x))$
```

The underline the assumption that before the turning point, the test error decreases as the number of iterations increases. So, for each iteration, after fitting the Adaboost classifier boundary $F_m(x)$ as defined in Algorithm 1, we keep track of the cross validation error of each classifier. Once the cross validation increases, we return the last classifier.

Cross validation stopping also works for regression problem, see Figure 4 for gradient boosted least square with early stopping. Compared to Figure 2, the optimal stopping iteration becomes bigger. The results in Figure 2 could be viewed as a 2-fold cross validation, therefore K is very important in determining the optimal stopping iteration. In statistics, $K = 5$ or $K = 10$ are suitable choices [6].

6 Consistency and Regularization of boosting

Leo Breiman (2000) proved the Bayesian risk consistency of boosting when the training data is infinite [2] (which is an unrealistic assumption in the real world). Breiman's work assumes that boosting is the exactly solution optimizing empirical risk. However, the consistency of boosting is

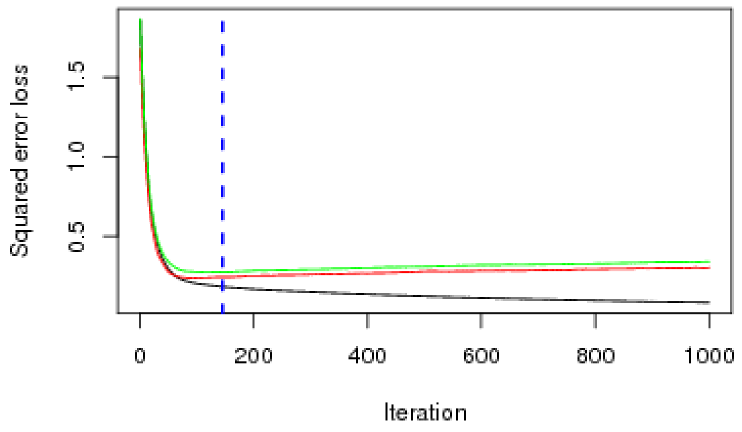


Figure 4: Performance of gradient boosted least square with 5-fold cross validation. The optimal number of iterations is 128.

influenced by the numerical optimization algorithm like forward stepwise fitting. Some work focus on the numerical optimization procedure, see [13] and [12].

Following Leo Breiman's style, which does not concern the details of the empirical risk minimization, is the work by Jiang [8] and Lugosi [11]. Jiang [8] showed that under regularity conditions about distribution of predictors and basis function set $\mathcal{F}$, there exists a sequence t_N such that

$$\lim_{N\to\infty} \mathbf{E}_{\{(X_i,Y_i)\}}\mathbf{P}[Y \neq sgn \circ F_{t_N}(X)] = \mathbf{P}[Y \neq sgn \circ F_B(X)] \tag{12}$$

where $F_B(X) = \frac{1}{2}\log[\mathbf{P}(Y = 1|X)/\mathbf{P}(Y = -1|X)]$ is the Bayesian classifier and N is the size of the training data.

Lugosi [11] studied the consistency of regularized Adaboost where the output is the solution of the following optimization problem

$$\begin{aligned} \underset{\beta_1,\cdots,\beta_M}{\text{minimize}} \quad & \frac{1}{N}\sum_{i=1}^{N}\exp[-Y_i\sum_{m=1}^{M}\beta_m f_m(x)] \\ \text{subject to} \quad & \sum_{m=1}^{M}|\beta_m| \leq \lambda \end{aligned} \tag{13}$$

For the consistency work considering the numerical optimization bias and the optimal stopping, Zhang and Yu [18] proved that for γ-Lipschitz convex function ϕ such that $\phi(-a) > \phi(a)$ for all $a > 0$, $L(Y, f(X)) = \phi(-Yf(X))$, there exist sequences k_N and β_N such that $k_N \to \infty$, $\beta_N \to \infty$ and

$$\lim_{N\to\infty} \gamma\beta_N R_N(\mathcal{F}) = 0$$

Then as long as we stop Algorithm 2 at a step $\hat{k}$ such that $\hat{k} \geq k_N$ and $||f_{\hat{k}}||_1 \leq \beta_N$

$$\lim_{N\to\infty} \mathbf{E}_{\{(X_i,Y_i)\}}\mathbf{E}L(Y, f_{\hat{k}}(X)) = \inf_{f\in\mathcal{F}} \mathbf{E}L(Y, f(X)) \tag{14}$$

References

[1] L. Breiman. Arcing the edge. Technical report, Technical Report 486, Statistics Department, University of California at Berkeley, 1997.

[2] L. Breiman. Some infinity theory for predictor ensembles. Technical report, Technical Report 579, Statistics Dept. UCB, 2000.

[3] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[4] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, volume 96, pages 148–156, 1996.

[5] J. Friedman, T. Hastie, R. Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2): 337–407, 2000.

[6] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[7] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[8] W. Jiang. Process consistency for adaboost. *Annals of Statistics*, pages 13–29, 2004.

[9] M. J. Kearns and U. V. Vazirani. *An introduction to computational learning theory*. MIT press, 1994.

[10] R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*, pages 1137–1145. Stanford, CA, 1995.

[11] G. Lugosi and N. Vayatis. On the bayes-risk consistency of regularized boosting methods. *Annals of Statistics*, pages 30–55, 2004.

[12] S. Mannor, R. Meir, and T. Zhang. The consistency of greedy algorithms for classification. In *International Conference on Computational Learning Theory*, pages 319–333. Springer, 2002.

[13] L. Mason, J. Baxter, P. L. Bartlett, M. Frean, et al. Functional gradient techniques for combining hypotheses. *Advances in Neural Information Processing Systems*, pages 221–246, 1999.

[14] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean. Boosting algorithms as gradient descent. In *Advances in Neural Information Processing Systems*, pages 512–518, 2000.

[15] G. Ridgeway. gbm: generalized boosted regression models. r package version 1.6-3.1, 2010.

[16] R. Y. Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

[17] R. E. Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[18] T. Zhang, B. Yu, et al. Boosting with early stopping: Convergence and consistency. *The Annals of Statistics*, 33(4):1538–1579, 2005.

Appendix

```
library(gbm)
N <- 1000
X1 <- runif(N)
X2 <- 2*runif(N)
X3 <- ordered(sample(letters[1:4],N,replace=TRUE),levels=letters
    [4:1])
X4 <- factor(sample(letters[1:6],N,replace=TRUE))
X5 <- factor(sample(letters[1:3],N,replace=TRUE))
X6 <- 3*runif(N)
mu <- c(-1,0,1,2)[as.numeric(X3)]
SNR <- 10 # signal-to-noise ratio
Y <- X1**1.5 + 2 * (X2**.5) + mu
sigma <- sqrt(var(Y)/SNR)
```

```
Y <- Y + rnorm(N,0,sigma)
# introduce some missing values
X1[sample(1:N,size=500)] <- NA
X4[sample(1:N,size=300)] <- NA

data <- data.frame(Y=Y,X1=X1,X2=X2,X3=X3,X4=X4,X5=X5,X6=X6)
# fit initial model
gbm1 <-
  gbm(Y~X1+X2+X3+X4+X5+X6, # formula
      data=data, # dataset
      var.monotone=c(0,0,0,0,0,0), # -1: monotone decrease,
      # +1: monotone increase,
      # 0: no monotone restrictions
      distribution="gaussian", # see the help for other choices
      n.trees=1000, # number of trees
      shrinkage=0.05, # shrinkage or learning rate,
      # 0.001 to 0.1 usually work
      interaction.depth=3, # 1: additive model, 2: two-way
    interactions, etc.
      bag.fraction = 0.5, # subsampling fraction, 0.5 is probably
    best
      train.fraction = 0.5, # fraction of data for training,
      # first train.fraction*N used for training
      n.minobsinnode = 10, # minimum total weight needed in each
    node
      cv.folds = 3, # do 3-fold cross-validation
      keep.data=TRUE, # keep a copy of the dataset with the object
      verbose=FALSE, # don't print out progress
      n.cores=1) # use only a single core (detecting #cores is
# error-prone, so avoided here)
# check performance using an out-of-bag estimator
# OOB underestimates the optimal number of iterations
best.iter <- gbm.perf(gbm1,method="OOB")
print(best.iter)
# check performance using a 50% heldout test set
best.iter <- gbm.perf(gbm1,method="test")
print(best.iter)
# check performance using 5-fold cross-validation
best.iter <- gbm.perf(gbm1,method="cv")
print(best.iter)
```

Listing 1: Demo for gradient boosted least square