6 Boosting

6.1 Introduction

One of the reasons why random forests is so effective for complex response functions is that it capitalizes on very flexible fitting procedures. As a result, it can respond to highly local features of the data. Such flexibility is desirable because it can substantially reduce the bias in fitted values compared to the fitted values from parametric regression, unless based on prior information, the parametric regression happens to hit upon an appropriate functional form.

The flexibility in random forests comes in part from individual trees that can find nonlinear relationships and interactions. Another source of the flexibility is large trees that are not precluded from having very small sample sizes in their terminal nodes. Yet another source of the flexibility is the sampling of predictors. Predictors that work well, but only for a very few observations, have the opportunity to participate.

But as now stated many times, that flexibility comes at a price: the risk of overfitting. Random forests consciously addresses overfitting by using OOB observations to construct the fitted values and measures of fit, and by averaging over trees. Experience to date suggests that this two-part strategy—very flexible fitting functions and averaging over OOB observations—can be highly effective.

But the two-part strategy, broadly conceived, can be implemented in other ways. An alternative method to accommodate highly local features of the data is to give the observations responsible for the local variation more weight in the fitting process. If in the binary case, for example, a fitting function misclassifies those observations, that function can be applied again, but with extra weight given to the observations misclassified. Then, after a large number of fitting attempts, each with difficult-to-classify observations given relatively more weight, overfitting can be reduced if the fitted values from the different fitting attempts are combined in a sensible fashion. Ideas such as these lead to very powerful statistical learning procedures that can compete with random forests. These procedures are called "boosting."

R.A. Berk, *Statistical Learning from a Regression Perspective*,
DOI: 10.1007/978-0-387-77501-2_6, © Springer Science+Business Media, LLC 2008

Boosting gets its name from its ability to take a “weak learning algorithm,” which performs just a bit better than random guessing, and “boosting” it into an arbitrarily “strong” learning algorithm (Schapire, 1999: 1). It “combines the outputs from many ‘weak’ classifiers to produce a powerful ‘committee’ ” (Hastie et al., 2001: 299). So, boosting has some of the same look and feel as random forests.

But, boosting formally differs from random forests in at least four important ways. First, in traditional boosting, there are no chance elements built in. At each iteration, boosting works with the full training sample and all of the predictors. Some recent developments in boosting exploit random samples from the training data, but these developments are enhancements that are not fundamental to the usual boosting algorithms. Second, with each iteration the observations that are misclassified, or otherwise poorly fitted, are given more relative weight. No such weighting is used in random forests. Third, the ultimate fitted values are a combination over a large set of earlier fitting attempts. But the combination is not a simple average as in random forests. Finally, the fitted values and measures of fit quality are usually constructed from the “within-sample” data. There are no out-of-bag observations, although some recent developments make that an option.

To appreciate how these pieces can fit together, we turn to Adaboost, which is perhaps the most widely known boosting procedure (Freund and Schapire, 1996). For reasons we soon examine, the “ada” in Adaboost stands for “adaptive” (Schapire, 1999: 2). Adaboost illustrates well boosting’s key features and despite a host of more recent boosting procedures is still among the best classifiers available (Mease and Wyner, 2008).

6.2 Adaboost

Adaboost is the poster child for boosting and provides a useful introduction to the method. It was designed originally for classification problems, which once again are discussed first.

Consider a binary response coded as 1 or –1. Adaboost then has the following general structure. The pseudocode that follows is basically a reproduction of what Hastie et al. (2001) show on their page 301.

```
1. Initialize the observation weights $w_i = 1/N, i = 1, 2, \ldots, N$.
2. For $m = 1$ to $M$:
    a) Fit a classifier $G_m(x)$ to the training data using the weights $w_i$.
    b) Compute: $err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{n} w_i}$.
    c) Compute $\alpha_m = \log[(1 - err_m)/err_m]$.
    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 $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.
```

There are N cases and M iterations. $G_m(x)$ is a classifier for pass m over the data. Any number of procedures might be used to build a classifier, but highly truncated trees (called "stumps") are common. The operator I is an indicator variable equal to 1 if the logical relationship is true, and 0 otherwise. The binary response is coded 1 and –1 so that the sign defines the outcome. The classification error for pass m over the data is denoted by err_m.

The value of err_m is transformed into a logit, which then defines α_m. The new weights, one for each case, are then computed as w_i. All cases incorrectly classified are "up-weighted" relative to the previous pass over the data by e^{α_m}. Consequently, Adaboost will pay relatively more attention in the next iteration to the cases that were misclassified. In some expositions of Adaboost (Freund and Schapire, 1999), α_m is defined as $\frac{1}{2}\log(1 - err_m/err_m)$. Then, incorrectly classified cases are up-weighted by e^{α_m} and correctly classified cases are down-weighted by $e^{-\alpha_m}$. In the end, classification is determined by a vote over the M classifiers G_m, with each vote weighted by α_m.

To summarize, Adaboost combines a large number of fitting attempts of the data. Each fitting attempt is undertaken by a classifier using weighted observations. The observation weights are a function of how poorly an observation was fitted in the previous iteration. The fitted values from each iteration are then combined as a weighted average. There is one weight for each fitting attempt, applied to all of the fitted values, which is a function of the overall classification error of that fitting attempt. The observation weights and the iteration weights both are a function of the classification error, however, their forms and purposes are quite different.

There are now several variants on the basic Adaboost algorithm (Friedman et al., 2000). For example, one can think of Adaboost as "discrete" Adaboost because the fitting function produces a binary response. "Real" Adaboost exploits a fitting function that generates class membership probabilities instead. For these probabilities, the log-odds of class membership can be computed, which in turn, are used instead of an assigned class when weights are updated. Because the output from real Adaboost is less lumpy than from Adaboost, a claim is made that the algorithm may perform a bit better. "Gentle Adaboost" is a more robust version of Adaboost whose loss function gives less weight to extreme values. Limited experience to date with realistic datasets suggests that all three procedures classify about equally well. But there are exceptions, and it is possible to construct datasets in which one or the other will perform substantially better. More will be said about such performance assessments later.

6.2.1 A Toy Numerical Example of Adaboost

To help fix these ideas, it is useful to go through a numerical illustration with very simple data. There are five observations with response variable values for $i = 1, 2, 3, 4, 5$ of $1, 1, 1, -1, -1$, respectively.

1. Initialize the observations with each weight $w_i = 1/5$.

2. For the first iteration using the equal weights, suppose the fitted values for observations $i = 1, 2, 3, 4, 5$ are $1, 1, 1, 1, 1$. The first three are correct and the last two are incorrect. The error for this first iteration is:
$$err_1 = \frac{(.20 \times 0) + (.20 \times 0) + (.20 \times 0) + (.20 \times 1) + (.20 \times 1)}{1} = .40.$$
3. The weight to be given to this iteration is
$$\alpha_1 = \log \frac{(1 - .40)}{.40} = \log(.60/.40) = \log(1.5) = .41.$$
4. The new weights are:
$$w_1 = .20 \times e^{(.41 \times 0)} = .20$$
$$w_2 = .20 \times e^{(.41 \times 0)} = .20$$
$$w_3 = .20 \times e^{(.41 \times 0)} = .20$$
$$w_4 = .20 \times e^{(.41 \times 1)} = .30$$
$$w_5 = .20 \times e^{(.41 \times 1)} = .30$$
5. Now we begin the second iteration. We fit the classifier again and for $i = 1, 2, 3, 4, 5$ get $1, 1, 1, 1, -1$. The first four are correct and the last one is incorrect. The error for the second iteration is
$$err_2 = \frac{[(.20 \times 0) + (.20 \times 0) + (.20 \times 0) + (.30 \times 0) + (.30 \times 1)]}{1.2} = .25$$
6. The weight to be given to this iteration is
$$\alpha_2 = \log \frac{(1 - 25)}{.25} = \log(.75/.25) = 1.1.$$
7. We would normally keep iterating, beginning with the calculation of a third set of weights. But suppose we are done. The classes assigned are:

$$\hat{y}_1 = \text{sign}[(1 \times .41) + (1 \times 1.1)] > 0 \Rightarrow 1$$
$$\hat{y}_2 = \text{sign}[(1 \times .41) + (1 \times 1.1)] > 0 \Rightarrow 1$$
$$\hat{y}_3 = \text{sign}[(1 \times .41) + (1 \times 1.1)] > 0 \Rightarrow 1$$
$$\hat{y}_4 = \text{sign}[(1 \times .41) + (1 \times 1.1)] > 0 \Rightarrow 1$$
$$\hat{y}_5 = \text{sign}[(1 \times .41) + (-1 \times 1.1)] < 0 \Rightarrow -1.$$

One can see in this toy example how in the second iteration, the misclassified observations are given relatively more weight. One can also see that the class assigned (i.e., +1 or –1) is just a weighted average of the classes assigned at each iteration. The second iteration had fewer wrong (one out of five rather than two out of five) and so was given more weight in the ultimate averaging. These principles would apply even for very large datasets and thousands of iterations.

6.2.2 A Statistical Perspective on Adaboost

Adaboost has of late been studied by statisticians, (e.g., Ridgeway, 1999; Friedman et al., 2000; Wyner, 2003; Bühlmann and Yu, 2004; Friedman et al., 2004; Zhang and Yu, 2005; Mease at al., 2007; Mease and Wyner, 2008) and in response, by its original inventors (Shapire, 2002). From this recent work several interesting features of Adaboost have been clarified.

Adaboost fits a stagewise additive model using basis functions in much the same spirit as CART and random forests. In CART, the basis functions are indicator variables that determine the optimal splits. Once a split is defined, it is fixed. Later splits have no impact on earlier splits. Each terminal node is characterized by a set of indicator variables that define the basis functions for that node. The weighted sum of the basis functions is the classifier for all of the data.

In random forests, the basis functions are the individual tree classifiers, each a function of X. Earlier trees are unaffected by later trees. Classes are assigned to observations by determining the class most commonly assigned over trees. Votes are summed over trees, with each tree weighted the same.

In Adaboost, as with random forests, the basis functions are the individual "weak" classifiers, each also a function of X. Often these weak classifiers are trees. Earlier classifications are unaffected by later classifications. Classes are assigned by a weighted sum over classifiers, with weights determined by the values of α_m.

Given the broad similarities between Adaboost and random forests, it is not surprising that many of the same concepts can be applied to both. Just as in random forests, for example, there is in Adaboost a margin, which plays much the same role as the margin in random forests. Thus, a larger margin implies less generalization error. There can also be in Adaboost population generalization error, although unlike random forests, there is no convergence to that value as the number of iterations increases without limit. The lack of convergence raises some important issues (addressed shortly) about how Adaboost should be used in practice.

There also are a number of formal links between Adaboost and a variety of statistical procedures that provide a very useful bridge between the two and a statistical framework within which to place boosting (Breiman, 1999; Friedman et al., 2000). With this framework in place, several important conclusions can be derived that have significant implications for work with real data.

Recall that in conventional parametric regression with a quantitative response variable, the goal is to fit conditional means of the response variable. If the intent is description, the regression hyperplane for the training data ideally goes right through the conditional means. If the enterprise is estimation, the regression hyperplane provides unbiased estimates of the conditional means in the population, or the conditional means implied by the underlying stochastic process. As noted in Chapter 1, the same motives can drive any of the statistical learning procedures we have discussed when there is a quantitative response. When the response variable is categorical, a similar motivating framework can be applied, but the units of the response are different. For example, it is common to fit or estimate the conditional log odds (i.e., conditional logits) of the response categories. What is Adaboost trying to fit?

Hastie et al. (2001: 306–308) show that the Adaboost iterations are implicitly targeting

$$f(X) = \frac{1}{2}\log\frac{\hat{P}(Y=1|X)}{\hat{P}(Y=-1|X)}. \tag{6.1}$$

This is just one-half the usual log-odds (logit) function for $P(Y = 1|X)$. The 1/2 implies using the sign to determine the class. In other words, the "solution" Adaboost is seeking is the population conditional probabilities, or if within-sample results are all that matter, the conditional proportions. This is familiar territory. However, the means to the end shown in Equation 6.1 is minimizing the loss function $e^{-yf(x)}$. Adaboost is attempting to minimize exponential loss with the observed class and the predicted class as its arguments. This focus on exponential loss raises at least two important issues.

First, relying on the mathematical relationship between the exponential loss function and conditional probabilities can miss a key point in practice. Mease and Wyner (2008) show that although at each stage the true conditional probability is indeed the minimizer, over stages there can be gross overfitting of the estimated probabilities. Mease et al. (2007) had earlier demonstrated that because classification depends only on the sign of the classifier, the computed probabilities of class membership are pushed toward 0.0 or 1.0 as the number of iterations increases. In other words, even when there is no evidence of overfitting for class membership, there can still be massive overfitting of the conditional probabilities (Buja et al., 2008). Indeed, the massive overfitting is desirable because it implies large margins for the fitted classes. Possible solutions to this form of overfitting are discussed later, but one must be very cautious about making too much of the estimated conditional probabilities.

Second, the exclusive attachment to the exponential loss function naturally raises the question of whether there are other loss functions that might perform better. Hastie and his colleagues (2001: 306–309) show that minimizing negative binomial log likelihood (i.e., the deviance) is also (as in Adaboost) in service of finding the true conditional probabilities, or the within-sample conditional proportions. Might this loss function, implemented as "Logitboost," be preferred?

On the matter of overfitting conditional probabilities, the answer is no. The same overfitting problems surface (Mease et al., 2007). With respect to estimating class membership, the answer is maybe. Hastie et al. (2001: 308–312) show that the Logitboost loss function is somewhat more robust to outliers than the Adaboost loss function. They argue that, therefore, Logitboost may be preferred if a significant number of the observed classes on the response variable are likely to be systematically wrong or noisy. Biased or noisy measurement could produce large disparities between the observed and fitted classes that would tend to dominate the fit. However, Mease and Wyner (2007) show through simulation counterexamples that this advice can often be wrong. There is no doubt that exponential loss is more vulnerable to outliers in principle, but the implications of this for practice are not yet clear. Perhaps the best advice when analyzing a given dataset is to try both procedures with the training data, and then with test data see which classifies more accurately.

Despite these and other controversies, there are some real gains to be had placing boosting within a statistical framework of loss function minimization. Statisticians have done a lot of thinking about loss functions. We turn to one particularly useful approach shortly.

6.3 Why Does Adaboost Work So Well?

There is no formal stopping rule for Adaboost and as a result, Adaboost can overfit (Jiang, 2004). The number of passes over the data is a tuning parameter that in practice depends on trial and error, often indexed by a measure of fit. One such measure is the cross-validation statistic, but there are several others that each penalize model complexity a bit differently. Often the number of classification errors will decline up to a certain number of passes over the data and then begin to increase. The point of inflection can sometimes be treated as a useful stopping point. But, there is nothing in boosting implying convergence.

Indeed, for a given sample size, "boosting forever" is not consistent (Mannor et al., 2002). But, for a given stopping point, Zhang and Yu (2005) show that under fairly general conditions, boosting will estimate the population generalization error as the number of observations increases without limit. The population characteristic being estimated is essentially the same as Breiman's for random forests, but the number of observations rather than the number of iterations increases without limit. Importantly, the same caveat holds: the proof of consistency says nothing about the quality of the population classifier responsible for generalization error.

There are also some interesting twists on the usual problem of what can be learned from asymptotics about the results from some data on hand. For example, in a given sample, there may be by random selection no observations where there are several critical turning points in the $f(X)$. It is likely, therefore, that at least those turning points will be fitted in a misleading manner, which a proof of consistency cannot usefully address. That is, even if an estimator can be shown to be consistent under certain conditions, a given sample may miss key features of the $f(X)$.

Still, there is broad consensus that Adaboost performs remarkably well. Part of the reason may be that as does random forests, Adaboost provides an opportunity for many different predictors to usefully contribute. Rather than working with random subsets of predictors, Adaboost reweights the data so that predictors that might have contributed little to the fit in earlier stages may do so for later stages. In the same spirit, a basis function for a given predictor that may work poorly at an early stage may work better at a later stage. Just as in random forests, the result is a very flexible fitting procedure that can help reduce bias in the fitted values. Then in the weighted averaging process, the variance can be brought under better control.

6.3.1 Least Angle Regression (LARS)

One can obtain a useful window on this process through Least Angle Regression (Efron et al., 2004). Although the initial focus for LARS was on model selection, there are some insights to be had on boosting. LARS proceeds in a stagewise fashion in the spirit of forward stepwise regression. But rather than making an all-or-nothing decision about how a prospective regressor should participate in the model, each variable selected has its role somewhat diluted.

Recall conventional forward stepwise regression. For a given response variable,

1. Find the predictor that has the largest absolute correlation with the response.
2. Compute the one-predictor regression equation and the residuals.
3. Find among the remaining predictors the one that has the largest absolute correlation with the residuals.
4. Add that predictor to the model, and compute the two-predictor regression equation and the residuals.
5. Keep adding predictors in this fashion until there is an insufficient improvement in the model's performance.

A usual feature of this approach is that the basis function that produces the largest reduction in the error sum of squares is the one added to the model. Such algorithms are sometimes characterized as "greedy" because by a specified criterion they make an optimal decision at each step that does not necessarily lead to a global optimum. Greedy algorithms have the advantage, however, of being practical and often give very good results. The alternative of searching over all possible models for a global optimum is typically far too taxing and in some cases, effectively impossible.

The full impact of the included predictors is at each step transmitted through the fitted values to the residuals. One result is residuals that are uncorrelated with the regressors currently in the model. Another result is that other predictors correlated with the regressors already selected may have their chances of being included seriously compromised. Their potential fitting capabilities may be to some extent pre-empted by predictors already in the model.

Consider now stagewise regression. Stagewise regression has some of the same look and feel as forward stepwise regression, but there are important differences. As before, there is a response variable and for ease of exposition, it is common to assume predictors that have been standardized to have a mean of 0.0 and a standard deviation of 1.0. Then,

1. Find the predictor that has the largest absolute correlation with the response.
2. Compute a "regression" coefficient proportional to that correlation coefficient.
3. Compute the fitted values and residuals.
4. Find among the remaining predictors the one that has the largest absolute correlation with the new residuals.
5. Compute a "regression" coefficient proportional to that correlation coefficient.
6. Update the fitted values and compute new residuals.
7. Keep adding predictors in this fashion until there is an insufficient improvement in the model's performance.

Underneath Steps 3 and 6 is an updating procedure for the fitted values of the following form,

$$\hat{y} \rightarrow \hat{y} + \varepsilon \cdot \mathrm{sign}(\hat{c}_j) \cdot x_j, \tag{6.2}$$

where $\hat{y}$ is the fitted values of the response, x_j is the predictor selected at a given stage, c_j is proportional to correlation with the residualized response, and ε is a small constant (Efron et al., 2004: 410). Then, residuals are computed as one would expect by subtracting the updated fitted values from the values of the observed response. These residuals serve as the response variable for the next stage.

At each stage, therefore, a new fitting equation with a single new predictor is computed from which new residuals follow. The impact of each new predictor is discounted substantially by the multiplicative factor ε. The result is residuals whose correlation with each newly added predictor is reduced but not eliminated. Prospective predictors, correlated with the included predictors are then more likely to remain in play and have a greater chance of being included in later stages. Also, predictors used in earlier stages can be used again in later stages. In this sense, the "greediness" of the algorithm is reduced.

How is the value of ε determined? To date, there is no formal way to determine the value of ε. The standard recommendation is that ε should be very small (e.g., .01) and that the fitting process should be allowed to run for several thousand stages. There is some experience suggesting that in this manner, a very flexible fitting function will result. In short, the fitted values should be allowed to very gradually arrive at a satisfactory result.

LARS can be seen in part as a computational shortcut for stagewise regression. Rather than incrementing the fitted values in tiny steps over many stages, LARS only requires up to as many stages as there are regressors. It arrives more quickly at the desired regression function by using different weights in the updating process. At any given stage, the residuals from the previous stage serve as the response varible. But when a new predictor is added to the model, all of the earlier included predictors, as well as the new one, participate in the fitting process. Moreover, they are each forced to have equal impact on the fitted values regardless of what might have happened had, say, forward stepwise regression been applied to the data.

Like for stagewise regression, the LARS residuals are not forced to be uncorrelated with the included predictors, and regressors can have more than one opportunity to contribute to the fitted values. Also, predictor variables not yet in the model have a better chance of being included in later stages. In the language we have been using, weak and highly specialized predictors can participate in the fitting process. The result is a very flexible fitting function.

But, what has all this to do with boosting? In conventional regression, regressors are either included in the model or not. In some circumstances this is too ham-fisted. Predictors compete for the opportunity to be included, and there are clear winners and clear losers. LARS illustrates how that the all-or-nothing strategy can be improved. Often it will be better to blend the impact of a wide variety of predictors rather than choose among them. Random forests and boosting share much the same perspective. And this can be very effective when highly flexible fitting functions are needed and when there is not a clear distinction between the regressors that belong in the model and those that do not. Another benefit is a kind of shrinkage that can increase the stability of the fitted values. As new predictors are added to the model, their weights are discounted and their potential impact on the fitted values is spread across other predictors.

Finally, the links among LARS, the lasso, and boosting mean that one can see boosting in part as a shrinkage procedure (Bühlmann and Yu, 2006; Bühlmann, 2006). If one looks backwards from the final boosting pass through the data, there is shrinkage at work. The shrinkage is more dramatic as one gets closer to the initial iteration.

In summary, there is no definitive explanation of why Adaboost works as well as it does. But it is likely that two factors are important: the weighting of residuals over many passes through the data so that weak and strong predictors can play a role, and the averaging across sets of fitted values that help to reduce overfitting and to increase stability. These assets carry over to a wider range of boosting procedures to which we can turn now.

6.4 Stochastic Gradient Boosting

Boosting can be approached from an unusually wide variety of perspectives (Shapire, 2002). Many different classifiers can be boosted using many different algorithms and loss functions. It also seems that boosting is related not just to a number of traditions in statistics but to game theory, linear programming, other core areas in applied mathematics, and computer science. Finally, boosting is "hot." The half-dozen journals or so that publish work on statistical learning have in almost every issue a useful paper on boosting or related approaches.

To have so much new and interesting work is surely a joy for the researchers. But practitioners are faced with the serious problem of needing stable tools already programmed that reflect the best of these recent developments. Yet, it is not even clear at this point which procedures work best for which kinds of data analyses or even any consensus on how "best" is to be defined. And the availability of software depends as much on happenstance as a well-considered response to what the user community needs.

In so confused and volatile an environment, moving from discussions of the various procedures in principle to tools ready for serious practice means making a number of educated guesses about what will be most productive. To that end, the material that follows emphasizes boosting additive trees. Just as in random forests, CART serves as the base classifier. We also limit ourselves to several rather conventional loss functions. Finally, we stay within R and what is one of the best implementations of boosting widely available (Ridgeway 2005), based on stochastic gradient boosting (Friedman, 2001, 2002). Experience to date suggests that stochastic gradient boosting using trees provides a very flexible boosting framework without, in general, sacrificing fitting performance. Moreover, stochastic gradient boosting can be used with either categorical response variables or quantitative response variables, depending on the loss function used. Finally, stochastic gradient boosting is closely linked to a number of common statistical traditions which, given the exposition style of this book, will seem familiar.

The basic logic behind stochastic gradient boosting is very clever. What follows is a first approximation of gradient boosting with a few key details overlooked for now. As has become our didactic practice, we start with a binary response variable.

Suppose that the response variable is binary and coded as 1 or 0. From a regression tree, not a classification tree, fitted values $\hat{y}_i$ can be obtained. For each observation, there is also the observed value of the response y_i. After applying a monotonic transformation to y_i (details later), the transformed values of $\hat{y}_i$ are subtracted from the y_i to obtain what are, in effect, residuals. In the next iteration, a regression tree is fit to these residuals. The new set of fitted values is then added to the old fitted values (details later) to obtain a new set of fitted values. After a sufficient number of such iterations, the last set of fitted values can be used to assign classes. Commonly, observations with $\hat{y}_i > 0.5$ are assigned a "1," and observations with $\hat{y}_i \leq 0.5$ are assigned a "0."

Larger positive or negative residuals imply that for those observations, the fitted values are less successful. When the regression tree attempts to maximize the quality of the fit overall, it will respond more to the observations with larger positive or negative residuals. In effect, therefore, these residuals serve as weights and hence, provide a key connection to boosting as originally proposed.

Now consider gradient boosting more formally. The discussion that follows on boosting trees draws heavily on Ridgeway (1999) and on Hastie et al. (2001: Sections 4.9–4.11.

A given tree can be represented as

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j), \tag{6.3}$$

with, as before, the tree parameters $\Theta = \{R_j, \gamma_j\}$, where j is an index of the terminal node, $j, \ldots, J$, R_j a predictor-space region defined by the jth terminal node, and γ_j is the value assigned in each observation in the jth terminal node. The goal is to construct values for the unknown parameters Θ so that the loss function is minimized. At this point, no particular loss is specified, and we seek

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j). \tag{6.4}$$

How this can be done for a given tree was discussed when CART was examined. The problem now is more difficult. We seek to minimize the loss over a set of trees. We once again proceed in a stagewise fashion so that at iteration m we need to find

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i;\Theta_m)), \tag{6.5}$$

where $f_{m-1}(x_i)$ are the results as of the previous tree. Given the results from the previous tree, the intent is to reduce the loss as much as possible using the fitted values from the next tree. This can be accomplished through an astute determination of $\hat{\Theta}_m = [R_{jm}, \gamma_{jm}]$ for $j = 1, 2, \ldots, J_m$. Thus, Equation 6.5 is a way to update the fitted values in an optimal manner.

Equation 6.5 can be reformulated as a numerical optimization task. In this framework, g_{im} is the gradient for the ith observation on iteration m, defined as the partial derivative of the loss with respect to the fitting function. Thus,

$$g_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}. \tag{6.6}$$

Equation 6.6 represents for each observation the potential reduction in the loss as the fitting function $f(x_i)$ is altered. The larger the absolute value of g_{im}, the greater is the change in the loss as $f(x_i)$ changes. So, an effective fitting function would respond most to the larger absolute values of g_{im}.

The g_{im} will generally vary across observations. A way must be found to exploit the g_{im} so that over all of the observations, the loss is reduced the most it can be. One approach is to use a numerical method called "steepest descent," in which a "step length" ρ_m is found so that

$$\rho_m = \arg\min_{\rho} L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m). \tag{6.7}$$

In other words, a scalar ρ_m is determined for iteration m so that when it multiplies the vector of gradients, the loss function from the previous iteration is reduced the most it can be.

The link between the method of steepest descent and gradient boosting is g_{im}. Consider the disparities between tree-generated fitted values and the actual values of the response. Those disparities are a critical input to the loss function. The size of the loss depends on all of the N disparities, but larger disparities make greater contributions to the loss than smaller disparities. Thus, a fitting function will reduce the loss more substantially if it does an especially good job at reducing the larger disparities between its fitted values and the actual values. There is a greater payoff in concentrating on the larger disparities. Thus, disparities resulting from the fitting process play much the same role as the gradients in the method of steepest descent.

And now the payoff. Friedman (2002) show that if one uses certain transformations of the disparities as the gradients (details soon), there is a least squares solution to finding the best parameter values for the fitting function. That is,

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i; \Theta))^2. \tag{6.8}$$

What this means in practice is that if one fits successive regression trees by least squares, each time using as the "response variable" a certain transformation of the disparities produced by the previous regression tree, one can obtain a useful approximation of the required parameters. For a binary outcome, the classifier that results, based on a large number of combined regression trees, is much the same as Adaboost. Moreover, by recasting the boosting process in gradient terms, many useful variants follow.

We turn, then, to the steps involved in gradient boosting as implemented in gbm(), the software R we soon apply. The algorithm is also called stochastic gradient boosting because of the random sampling in Step 2b below.

Consider a training dataset with N observations and p variables, including the response y and the predictors x.

1. Initialize $f_0(x)$ so that the constant κ minimizes the loss function: $f_0(x) = \arg\min_\kappa \sum_{i=1}^{N} L(y_i, \kappa)$.
2. For m in $1, \ldots, M$, do Steps a through e.
   a) For $i = 1, 2, \ldots, N$ compute the negative gradient as the working response

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

b) Randomly select without replacement $W \times p$ cases from the data set, where W is less than the total number of observations. Note that this is a simple random sample, not a bootstrap sample. How large W should be is discussed shortly.
c) Using the randomly selected observations, fit a regression tree with J_m terminal nodes to the gradients r_{im}, giving regions R_{jm} for each terminal node $j = 1, 2, \ldots, J_m$.
d) For $j = 1, 2, \ldots, J_m$, compute the optimal terminal node prediction as

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

where region R_{jm} is denotes the set of x-values that define the terminal node j for iteration m.
e) Still using the sampled data, update $f_m(x)$ as

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

where ν is a "shrinkage" parameter that determines the learning rate. The importance of ν is discussed shortly.

3. Output $\hat{f}(x) = f_M(x)$.

Ridgeway (1999) has shown that by using this algorithmic structure, all of the procedures within the generalized linear model, plus several extensions of it, can properly be boosted by the stochastic gradient method. Stochastic gradient boosting relies on an empirical approximation of the true gradient (Hastie et al., 2001: Section 10.10). The trick is determining the right r_i for each special case. The "residuals" need to be defined. Among the definitions of r_{im} are the following.

1. Gaussian: $y_i - f(x_i) \ldots$ the usual regression residual.
2. Bernoulli: $y_i - \frac{1}{1+e^{-f(x_i)}} \ldots$ the difference between the binary outcome coded 1 or 0 and the fitted ("predicted") proportion for the conventional logit link function.
3. Poisson: $y_i - e^{f(x_i)} \ldots$ the difference between the observed count and the fitted count for the conventional log link function.
4. Laplace: $\text{sign}[y_i - f(x_i)] \ldots$ the sign of the difference between the values of the response variable and the fitted medians.
5. Adaboost: $-(2y_i - 1)e^{-(2y_i-1)f(x_i)} \ldots$ not within the generalized linear model so not as easily seen as a kind of "residual."

There are a number of other gradient boosting possibilities. Hastie and his colleagues (2001: 321) provide the gradient for a Huber robust regression. Ridgeway (2005) offers boosted proportional hazard regression in gbm().

Kriegler (2007) has added to gbm() a Laplace loss function that through an analogy to quantile regression, allows for asymmetric loss. More is said about Kriegler's work later.

Stochastic gradient boosting also can be linked to various kinds of penalized regression of the general form discussed in earlier chapters. One insight, implied earlier, is that the sequences of results that are produced with each pass over the data can be seen as a regularization process akin to shrinkage (Bühlmann and Yu, 2004; Friedman et al., 2004). There is less shrinkage with each successive pass over the data.

In short, with gradient boosting, each tree is constructed much as a conventional regression tree. The difference is how the "target" for the fitting is defined. By using disparities defined in particular ways, a wide range of fitting procedures can be boosted.

6.4.1 Tuning Parameters

The stochastic gradient boosting algorithm just described has two important innovations beyond the original version of gradient boosting. First, a page is taken from bagging with the use of random sampling in Step 2b to help control overfitting. The sampling is done without replacement, but as noted earlier, there can be an effective equivalence between sampling with and without replacement, at least for conventional bagging (Buja and Stuetzle, 2006).

The sample size, whether with or without replacement can be a tuning parameter. The issues are rather like those that arise when the number of folds in N-fold cross-validation is considered. And as with N-fold cross-validation, there seems to be no formal and general answer. Practice seems to favor a conventional sample size of N when sampling with replacement and a conventional sample size of $N/2$ when sampling without replacement. But it can make sense for any given data analysis to try sample sizes that also are about 25% smaller and larger and choosing the best sample size based on out-of-sample performance.

Second, it can be very useful to reduce the rate at which the updating occurs by setting ν to a value substantially less than 1.0 (Step 2e). A value of .001 often seems to work reasonably well, but values larger and smaller by up to a factor of 10 are usually worth trying as well. Again, the value of the tuning parameter is usually determined best by out-of-sample performance.

By slowing down the rate at which the algorithm "learns," a larger number of basis functions can be computed. The flexibility of the fitting process is increased, and the small steps lead to shrinkage at each pass through the data. A cost is a larger number of passes through the data. Fortunately, one can usually slow the learning process down substantially without a prohibitive increase in computing.

Third, a tuning parameter that also affects the flexibility of the fitting function is the "depth" of the interaction variables desired: no interactions, two-way, three-way, and so on. In other words, by allowing for what are, in effect, sets of product variables, one can increase the "dictionary" of basis functions evaluated. This may seem unnecessary because CART already has the capacity, at least in principle, for building interaction effects basis functions. But, depending on how the partitioning proceeds and on the ways in which predictor variables are related to one another, CART may fail to find some needed interactions or represent them improperly. For example, it may miss entirely a given two-way interaction or represent it as a three-way interaction. By explicitly building in interaction variables, one increases the chances that for many passes through the data CART will get it right.

The price, once again, can be computational. For example, one could include main effects plus all two-way and three-way interaction effects. But even for a small number of predictors, all two-way interactions alone will dramatically increase the number of terms evaluated at each CART partitioning of the data. In practice, therefore, it is rare to go beyond all two-way interactions. And unless the response function is thought to be rather complex, including only main effects may well suffice.

Fourth, yet another tuning parameter that affects fitting function flexibility is the minimum number of observations in each tree's terminal node. Smaller node sizes imply larger trees and a more flexible fitting function. Minimum terminal node sizes of between 5 and 15 seem to work reasonably well in many settings, but it can be worth experimenting with somewhat larger terminal node sizes if computational constraints are significant and if the number of observations used to construct each tree is large. The risk is that with larger terminal node sizes and the smaller trees that can result, some important nonlinearities may be missed.

Finally, the number of passes over the data needs to be determined. Because there is no convergence and no clear stopping rule, the usual practice is to run a large number of iterations and inspect a graph of the fitting error (e.g., residual deviance) plotted against the number of iterations. Usually, the error will decline rapidly at first and then level off. It can even start to increase when the number of iterations is very large. If there is an inflection point at which the fitting error starts to increase, the number of iterations can be stopped just short of that number. If there is no inflection point, the number of iterations can be determined by when reductions in the error effectively cease.

Determining the number of iterations is rarely a serious problem in practice. One proceeds in steps. Several hundred iterations are run, and the performance of the fitting procedure examined. There are often useful tools, such as cross-validation statistics, to help evaluate performance. If the results are unsatisfactory, more iterations are run. This process stops when the performance of the fitting procedures no longer seems to be improving. Stochastic gradient boosting results can be quite robust to the number of iterations used, once it is apparent that there are no important gains to be made. But there are exceptions. In particular, when the response is binary and interest centers on the fitted probabilities, the fitted probabilities can be quite sensitive to the number of iterations. An example is provided later.

There are some important relationships between the tuning parameters. The usual goal of a data analysis is to construct a set of fitted values with low bias and low variance. Larger trees, higher-order interaction variables, and smaller steps can contribute to reducing the bias. Smaller steps can also help reduce the variance by not allowing a few sets of widely varying fitted values to destabilize the procedure, and by indirectly increasing the need for more passes through the data. Random sampling, which will increase the independence between the sets of fitted values, also can help increase stability.

But exactly how the possible values for each of the tuning parameters should be tuned as a group is not apparent. Is one better off, for instance, to proceed with larger trees and small learning steps? More generally, can certain values for one tuning parameter compensate for certain values for the another? At this point, it is difficult to find clear guidance (Buja et al., 2008).

To summarize, there are several tuning parameters associated with stochastic gradient boosting. Fortunately, much of the available software comes with sensible defaults, and it is often a good idea to stick with these, at least at first. Then some trial-and-error tuning can also be useful. The only tuning parameter likely to need immediate attention is the number of trees to grow. The program gbm(), for example, offers useful information on how many trees are needed, but the user is free to do what seems appropriate. Perhaps the most important message is that gradient boosting can be quite forgiving in general with respect to its tuning parameters.

6.4.2 Output

The key output from stochastic gradient boosting is much the same as the key output from bagging: predicted classifications, predicted probabilities, error rates, and confusion tables. However, unlike bagging and random forests, there are not the usual out-of-bag observations. Therefore, the confusion tables commonly depend on resubstituted data; the data used to build the model are also used to evaluate its performance. As a result, it can be important to have both a training dataset and a test dataset. Confusion tables should be constructed from the test data set. If a simple random sampling option is available, a kind of out-of-bag data is available for evaluation. Recall these are not what is excluded from random samples drawn with replacement, but a fraction of the total training dataset not chosen when a subset of the observations is selected for each tree. Exactly how these data are used will depend on the software.

Just as for bagging and random forests, the use of multiple trees means that it is impractical to examine tree diagrams to learn how individual predictors perform. The solutions currently available are much like those implemented for random forests. There are partial dependence plots that are effectively the same as those used in random forests. However, these plots must be treated cautiously when the outcome variable is binary. Recall that in an effort to classify well, boosting can push the fitted probabilities away from .50 toward 0.0 and 1.0, and in the case of stochastic gradient boosting, the fitted probabilities can be very sensitive to values of the tuning parameters. Consequently, the fitted probabilities can be misleading. For partial dependence plots with binary predictors, the vertical axis is a function of these fitted probabilities, usually in a logit metric. If the probabilities are suspect, so are the logits.

There are also importance measures for each predictor. The exact form these importance measures can take depends on the software used. But one common option is the reduction of the loss function normalized to 100. The software stores how the loss decreases when each predictor is chosen for particular splits over trees. The average decrease over trees is the raw contribution each predictor makes to the fit. Then these contributions are summed, and each contribution is reported as a proportion of the total. In gbm(), there is on a somewhat experimental basis a random shuffling approach to importance based on predictive skills, but to date it does not use the out-of-bag observations. So it does not represent true forecasting accuracy. Recall that for random forests, importance is defined by contributions to forecasting skill.

6.5 Some Problems and Some Possible Solutions

Because there are so many different kinds of boosting, it is difficult to arrive at any overall assessments of strengths and weaknesses. Moreover, the menu of boosting options continues to grow partly in response to concerns about the performance of older boosting methods. Nevertheless, a few provisional observations may be useful for practitioners.

6.5.1 Some Potential Problems

Boosting is a very powerful tool whose reach will no doubt expand in the near future. For many data analysis problems, it performs well and can be a legitimate competitor to random forests when either approach could be properly applied. But boosting also has some serious drawbacks.

As with random forests, the existing proofs of consistency are not fully satisfying. Suppose in the population there is some function of X, $h(X)$, constructed that links inputs of outputs. Under certain reasonable conditions, including a training dataset that is a random sample from that population, boosting will provide a consistent estimate of $h(X)$. But unless $h(X)$ is the same as the true mechanism $f(X)$ linking inputs to outputs, the function estimated from the training data will not be a consistent estimate of $f(X)$. That is, in large samples boosting can get the wrong function approximately right.

Boosting also can overfit the data. Unlike random forests, there is no mechanism in boosting for capitalizing on random samples of the data and then averaging the results over these samples. Some implementations of boosting have the option of cross-validation measures of fit or other measures that can provide useful guidance on when to stop the boosting process. But even a very good cross-validation stopping rule does not necessarily imply that all is well. The problems with binary outcomes and the fitted probabilities noted earlier are an instructive example.

A related matter is that costs are addressed solely within the functional form of the loss. In Adaboost, for example, classification errors are weighted exponentially but symmetrically. There is no distinction between false positives and false negatives and hence, no way to take their different costs directly into account. In stochastic gradient boosting, classification problems are transformed into regression problems when the residuals are defined. Thus, fitting errors as used in the algorithm are no longer categorical, and there are no longer false negatives and false positives. And the loss functions are once again symmetric. A positive residual of a given size is treated the same as a negative residual of the same size.

Similar issues carry over when the response is quantitative. Although the concepts of false negatives and false positives no longer apply, one might still wish for an asymmetric loss function. If the intent, for instance, is to characterize how the number of homeless people in a census tract is related to features of that tract, overestimates of the number of homeless might have very different consequences from underestimates of the number of homeless. Homeless advocates would perhaps see underestimates as more costly than overestimates. Local public officials might take the opposite view. But neither would like see the costs of overestimates and underestimates treated the same (Berk et al., 2008)

In short, the inability to take asymmetric costs into account means that symmetric costs are being assumed. In practice, this can be untenable. A closely related consequence is that there is no principled way to address the problems that can follow when a response variable is highly skewed. For classification exercises, then, it can be very difficult for boosting to perform better than assigning classes solely from the modal response variable category.

Finally, tuning parameters on occasion can make an important difference in the results. Then, one has no choice but to experiment with different sets of tuning parameter values. Unfortunately, this can be at best a trial-and-error process with too often no definitive resolution.

6.5.2 Some Potential Solutions

Many of boosting's vulnerabilities, just as for any statistical procedure, are exposed by inadequate data. Stated a bit differently, it will be rare indeed, for boosting to be able to solve problems stemming from weaknesses in the information on which it operates. Little more need be said about the importance of large samples, a rich set of predictors, and accurate measurement. Better data are always better, and data analysis difficulties that may seem to result from the boosting procedure applied, can be remedied if by data of higher quality.

The difficulties that can arise by assuming symmetric costs and/or working with highly skewed response variables can be addressed within the same broad framework. The key is to allow for asymmetric costs. For stochastic gradient boosting and quantitative response variables, Kriegler (2007) suggests attaching weights to the loss functions that can capture asymmetric costs. These weights, in turn, are carried forward when the empirical gradients are constructed so that the CART fitting process at each pass through the data takes them into account. To date, this idea has been employed for the Laplace, Gaussian, and Poisson distributions. Applications to real datasets look promising. One can use the weights to make the costs of forecasting errors responsive to policy and where appropriate, use such costs to adjust for skewed distributions. Asymmetric weighting for the Laplace distribution has been implemented in gbm(). An illustration is provided later.

When the response variable is binary, Mease and his colleagues (2007) argue for weighting the classification errors directly and asymmetrically within an Adaboost (not stochastic gradient boosting) framework. Imagine that one can estimate accurately the probability that a given case is in a given class. It is common to assign that case to a particular class if the associated probability is greater than .50. As noted in earlier chapters, the .50 threshold implies that the costs of false positives and false negatives are the same, and by raising the threshold above or below .50, asymmetric costs can be taken into account. For example, if the threshold were placed at .75, it would imply that the costs of falsely placing a case in the specified class are three times higher than the costs of incorrectly failing to place a case in that class. If one thinks of these thresholds as quantiles, there is a direct connection between the use of such quantiles and the use of costs in classification exercises. Using the quantiles to classify has been called quantile classification (Mease et al., 2007).

In practice, quantile classification is undertaken by oversampling or undersampling in much the same way it is done in random forests. The algorithm is called Jous-Boost and is available in R. The direct links between asymmetric costs, quantile classification, and disproportional stratified sampling allow one to implement costs sensitive boosting within an Adaboost framework. Moreover, one can then obtain appropriate estimates of the probability function. The details can be found in a paper by Mease and his colleagues (2007).

About all that can be said about problems with determining the values of tuning parameters is that it is important to be systematic in one's search of the parameter space. Increasingly, there is software to aid in this process. The procedure tune() in the *e1071* library is one example. It can also be important to appreciate at a deeper level that one usually tunes in response to some function of fit quality. For many applications this is appropriate. But there is no necessary connection between fit quality and scientific or policy responsiveness. As noted several times in earlier chapters, a better-fitting model may be less instructive than a worse-fitting model. Tuning for fit quality, therefore, is no assurance of sensible results. Finally, one must be cautious about boosting output that is highly sensitive to values of the tuning parameters. For example, one might reasonably decide that the fitted probabilities from a binomial model should not be used or interpreted. And, it would not be a good idea under these circumstances to use fitted probabilities to construct propensity scores (McCaffrey et al., 2004).

6.6 Some Examples

6.6.1 A Garden Variety Data Analysis

Among the most common kinds of analyses in the social sciences are regressions of wages on various biographical variables. We turn to some survey data from the Panel of Income Dynamics to do just such an analysis. We boost using a Gaussian loss function in gbm() to provide a relatively straightforward illustration.

Figure 6.1 shows a boosting performance plot. On the horizontal axis is the number of iterations. On the vertical axis is the change in the normal log-likelihood computed, in this case, from out-of-bag observations. These are the observations not used when the fractional simple random samples were drawn for each tree. They can provide a more conservative assessment of how well the iterations are doing than the resubstituted data. Because the point is to determine how well the interations are doing with the data actually being processed, it is not clear that a more conservative estimate is called for. No measure of fit is being computed that will be generalized to out-of-sample data. In this instance, and as expected, the big improvements come early with no substantial gains after about iteration 4000.

The purpose of Figure 6.1 is to see how the fitting proceeds as the number of iterations increases and to choose a cutoff point. If there is evidence that the performance has not bottomed out, additional iterations can be undertaken. If the performance curve has become effectively flat, there is important information about the useful number of additional iterations needed. Iterations beyond the cutoff point can be discarded.

Commonly, there is statistic a computed from OOB data or through cross-validation that evaluates whether the improvement in performance in a given iteration is worth the increase in complexity. Recall that each additional iteration can be viewed as adding another basis function, which makes the fitting procedure more complex. In this case, the cutoff was determined to be iteration 7746.

Figure 6.2 is an importance plot. Importance is measured by the reduction in the log-likelihood attributable to each predictor, then normalized so that the contributions to the fit add to 100. Recall that for CART the contribution

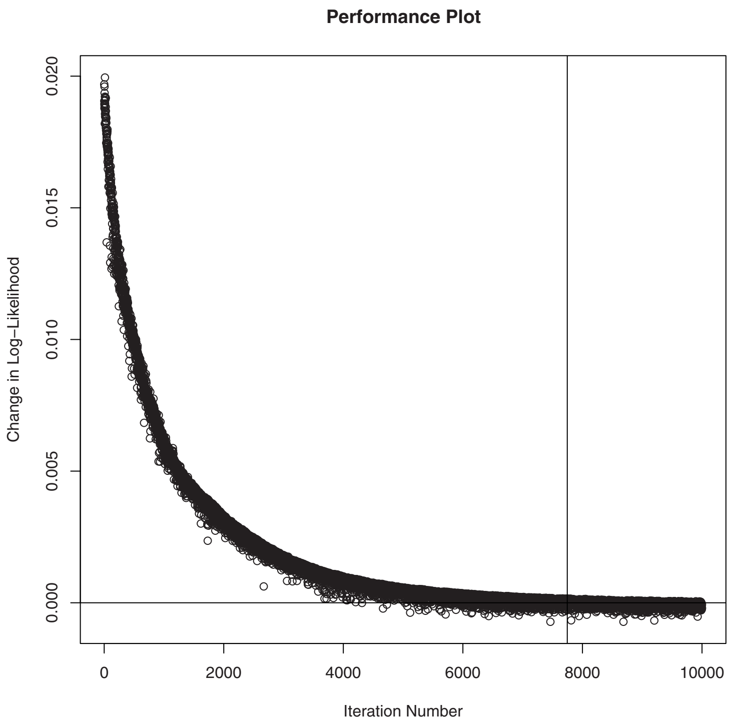

Fig. 6.1. Performance of Gaussian boosting for wages.

of each predictor to the fit of a given tree can be easily calculated. When a predictor is chosen as the splitting variable, the reduction in heterogeneity is determined. The sum of such reductions over the entire tree is that predictor's importance. Random forests averages each predictor's importance over trees. Stochastic gradient boosting, as implemented in gbm(), does the same. In Figure 6.2, a little more than 50% of the fit can be attributed to age. Education accounts for about 35% of the fit. Sex accounts for about 12% of the fit. Language spoken makes almost no contribution.

Finally, Figure 6.3 is the partial dependence plot for age. The vertical axis is in dollars per hour. The horizontal axis is in years of age. One can see that after about age 20, increases in age are associated with increases in wages, but that after about age 40, the relationship flattens out and after age 60, may even decline a bit. Note how one would have been misled had a linear relationship been assumed, and a quadratic form would have only done a little better. Even a cubic polynomial, had that been anticipated, would have missed some of the more interesting features of the relationship, such as the flat part up to about age 20.

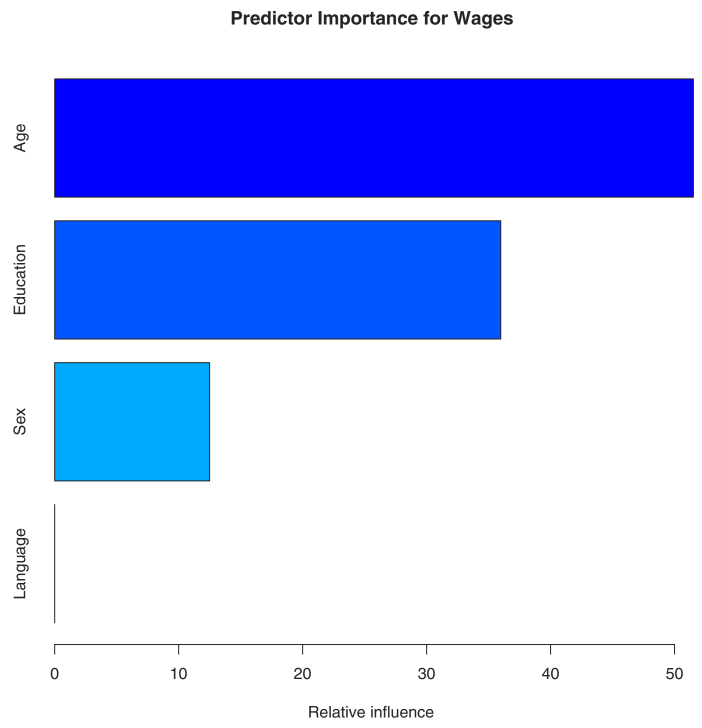

Fig. 6.2. Predictor importance for Gaussian boosting for wages.

In general, it is useful to construct partial dependence plots for all quantitative predictors as long as there are a sufficient number of different predictor values and a substantial number of observations for each. Recall that there is often no point in trying to overlay a smoother when the predictor values are few. But, a lot depends on the complexity of the partial response function. If the function is simple, a partial dependence plot based on few unique predictor values can be helpful. One must also consider whether there are a sufficient number of observations for each unique predictor value. Regions where the data are sparse risk unstable results that can make the response function look more complex than it really is.

It is not appropriate to construct dependence plots for the categorical variables such as sex and language. The values on the horizontal axis would be meaningless. Bar charts are a more useful option. For each category, the conditional mean is plotted. Recall that this is the strategy employed by random forests.

There are no confusion tables for quantitative response variables. But in principle, one has access to all of the usual regression diagnostics. For this

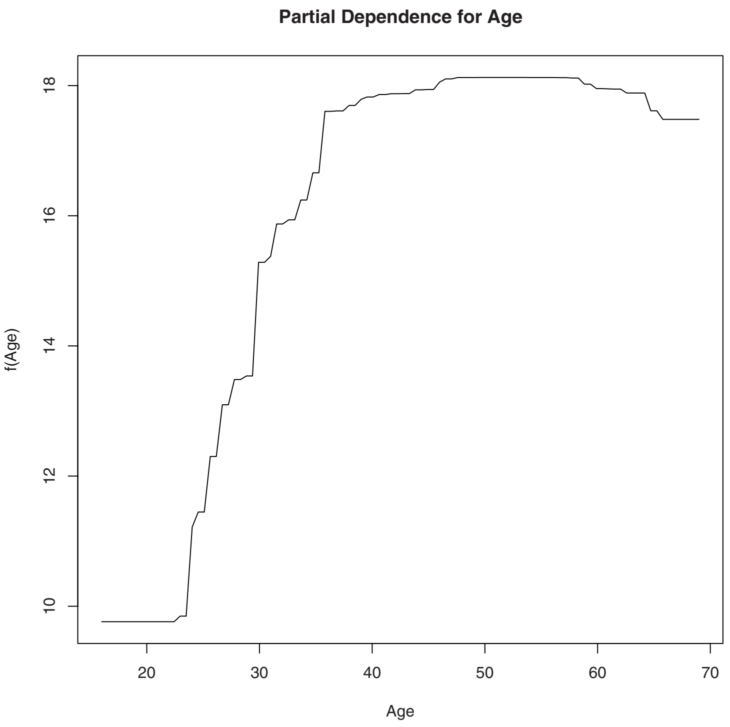

Fig. 6.3. Partial dependence on age for Gaussian boosting for wages.

boosted Gaussian regression, about 35% of the variance is accounted for by functions of the predictors. A conventional linear regression accounted for about 29% of the variance. Boosting clearly improves the fit in this example, although it also uses a larger effective number of parameters. Most of the improvement comes from the nonlinear relationship between age and income. Boosting can help most when one or more relationships between the response variable and predictors is complex.

Figure 6.4 shows four common plots used to evaluate the quality of a regression fit. Beginning with the plot in the upper-left corner, it is clear that the variance in wages increases with the average wage. This is confirmed by the plot at the lower left corner. In both, there also seems to be some evidence of a cluster of outliers suggesting an omitted categorical predictor. The plot on the upper-right corner indicates that the residuals are quite close to normal. The plot on the lower-right hand corner initially gives the impression that there are several influential observations, but the values they have for Cook's distance are very small.

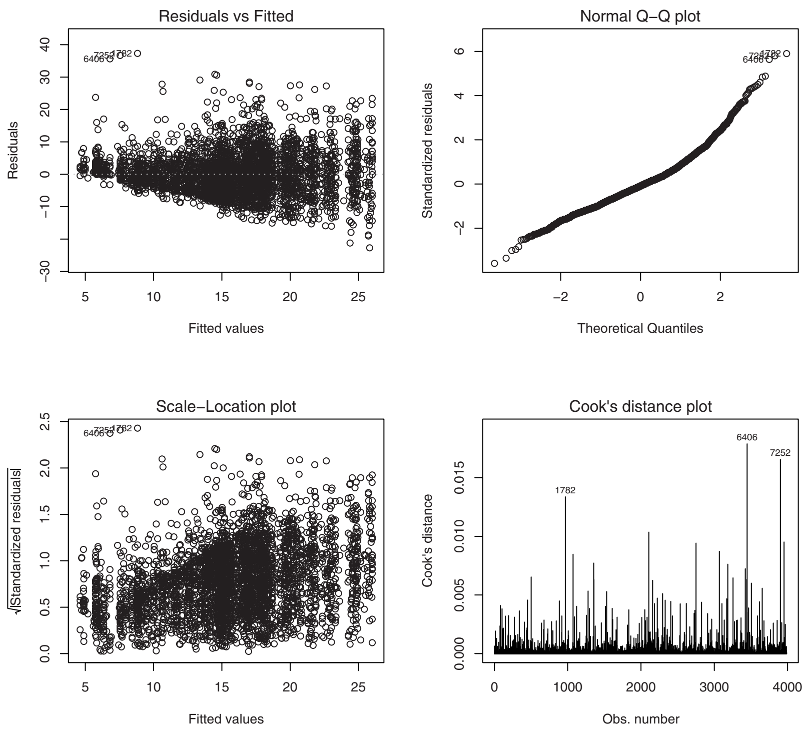

Fig. 6.4. Diagnostic plots for Gaussian boosting of wages.

We make more of such diagnostics in a later example. For now it may suffice to note that the importance of these diagnostics depends in part on the goals of the analysis. Just as in conventional regression, whether the residuals are normal, for instance, will typically not matter much unless the sample is small and traditional hypothesis tests and/or confidence intervals are desired.

6.6.2 Inmate Misconduct Again

Although boosting can be expected in general to perform at least as well as Gaussian regression models, it will sometimes shine when such a conventional regression model does not fit the data very well. The boosting process can improve the fit, sometimes dramatically. The same holds for binomial regression models. But when the response variable is highly unbalanced, there can be serious problems. To make this point, we return to the prison inmate data. We once again consider inmate misconduct using the same predictors as before.

Figure 6.5 shows a boosting performance plot. As expected, the big improvements come early with few real gains after about iteration 4000. The cutoff was determined to be iteration 6148.

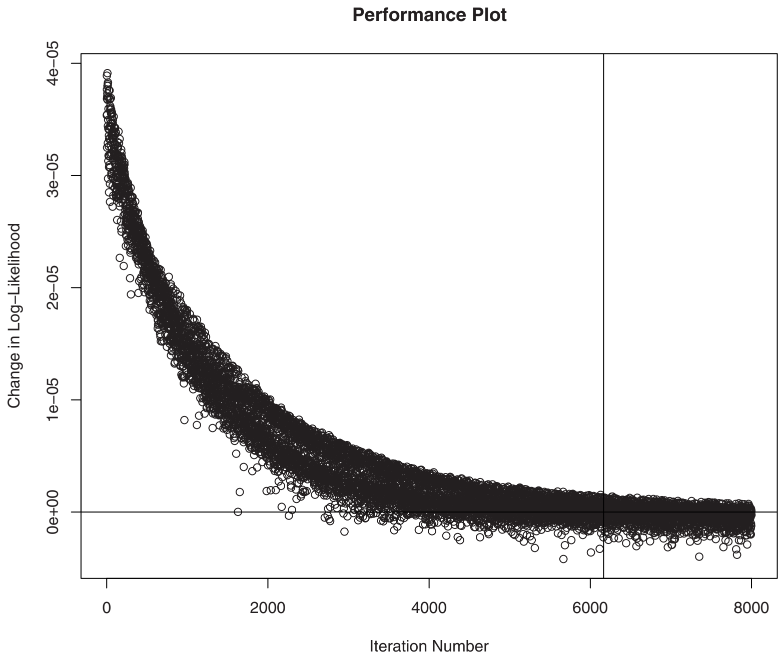

Fig. 6.5. Performance of binomial boosting for inmate misconduct.

Table 6.1 shows the confusion table using the training data, not OOB data or test data. Overall, the fit is quite good. Only about a fifth of the data are misclassified. However, all of the success comes from predicting the nomisconduct class well. And this is pretty easy to do with no predictors all because if the marginal distribution alone is used, and no misconduct is predicted, about 21% of the cases will be identified incorrectly. Moreover, boosting fails miserably when trying to identify inmates who engage in misconduct. Out of every 100 inmates who engaged in misconduct, only about 9 are correctly identified as such.

Table 6.1 is not necessarily an accurate rendering of how the classifier would perform in practice because only training data are used to construct the table. But it is likely from the reported use errors that if no misconduct was predicted, the forecast would be correct the vast majority of time. If misconduct was predicted, the forecast would be correct somewhat more than half the time.

	None Predicted	Misconduct Predicted	Model Error
No Misconduct	3745	62	0.01
Misconduct	900	99	0.91
Use Error	.19	.39	Overall Error = .21

Table 6.1. Confusion table for binomial boosting of inmate misconduct.

But the errors in use, as well as the model errors, depend on the costs of classification errors. And one can see where at least part of the problem lies. The cost ratio of false negatives to false positives is approximately .02. About one false positive is equal to about 50 false negatives. Recall that this is completely upside down from the point of view of corrections officials. And there is currently no way to intervene in the stochastic gradient boosting process and alter these relative costs.

In short, this is about the best that boosting is likely to do and indeed, probably overly optimistic because the confusion table is constructed from the training data. On these same data with the same predictors, random forests performs a bit better, but neither really shines with the default ratio of false negatives to false positives. Costs must be better taken into account. Then, using the cost ratio favored by prison administrators, random forests does dramatically better predicting the true positives.

Figure 6.6 shows predictor relative importance through their contributions to the fit. Sentence length dominates, followed by the two age variable, and gang activity is close behind. This is roughly consistent with our earlier random forest results but difficult to compare directly because Figure 6.6 is derived from contributions to the fit, not forecasting accuracy.

As one illustration of an estimated response function, Figure 6.7 shows the partial dependence plot for sentence length. The vertical axis is in logits as previously defined for partial dependence plots. Recall, $f_k(X) = \log[\mathrm{p}_k(X)] - \frac{1}{K}\sum_{k=1}^{K}\log[\mathrm{p}_k(X)]$, where $\mathrm{p}_k(X)$ is the proportion of observations in category k. However, because of the sensitive nature of the conditional probabilities, it is not clear how seriously one can take the logit values shown. For what it may be worth, inmate misconduct increases rather linearly with sentence length up to a sentence of around six years and then levels off.

We now repeat the analysis using very serious misconduct as the response. Recall that such behavior is very rare. A bit less than 3% of the inmates are reported for incidents of very serious misconduct. Figure 6.8 shows how the boosting algorithm performs. Many fewer iterations are required this time. Also, it is clear from the wider vertical spread of the points that the boosting results are much less stable than before. A likely explanation is that the margins associated with each iteration are substantially smaller and if so, it suggests that the boosted model is not faring well.

Indeed, boosting binomial regression fares quite badly in this case. The confusion table reproduces the marginal distribution of the response because

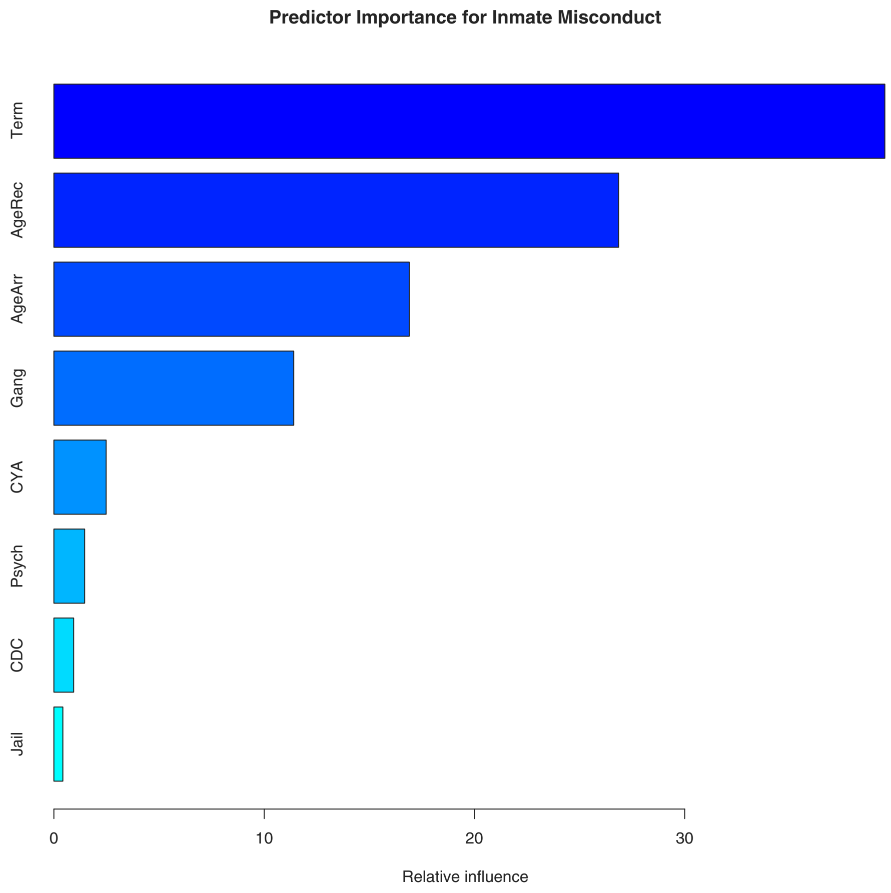

Fig. 6.6. Predictor importance for binomial boosting for inmate misconduct.

not a single inmate is identified as have engaged in very serious misconduct although 138 actually had. Figure 6.9 underscores how bad the performance is. The figure is a histogram for the subset of cases in which there actually was an incident of very serious misconduct. On the horizontal axis are the fitted probabilities. The largest of these values is less than .20, whereas a value of more than .50 is needed for an inmate to be classified as a serious risk. About the best that can be said about the analysis is that because no cases of very serious misconduct are correctly identified, the potential problems with fitted probabilities pushed toward 0 and 1 do not materialize.

Figure 6.10 shows the relative importance of each predictor for the quality of the fit. The pattern is largely the same, but gang activity has moved up to second place, and the gap in importance between sentence length and the

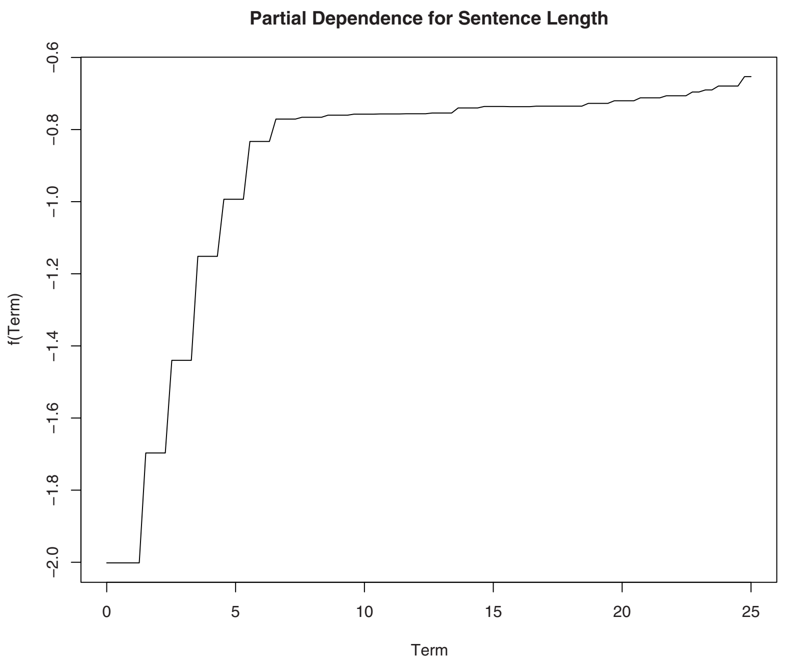

Fig. 6.7. Partial dependence on sentence length in years for binomial boosting for inmate misconduct

other predictors has increased. Thus, sentence length and gang activity are more important for the fit of very serious misconduct compared to the full range of inmate misconduct.

Finally, Figure 6.11 shows the partial dependence plot for sentence length. The response function is now roughly linear over all sentence lengths and does not flatten out for very long sentences. This may help to explain why sentence length has gained in its relative importance. For the reasons discussed earlier, however, it is may not be wise to make much of the response function shown.

In summary, in this application and for the default symmetric costs, boosting does a bit worse than random forests for incidents of general misconduct and much worse than random forests for very serious incidents of misconduct. Boosting, just as conventional regression can stumble badly with highly unbalanced response variables. Moreover, for a binary outcome and stochastic gradient boosting, there is currently no direct way to build asymmetric costs into the fitting process. Consequently, the views of corrections administrators cannot be taken into account. The best one can do is change the classification threshold so that fitted probabilities other than .50 determine the assigned class. This assumes that one has confidence in those probabilities.

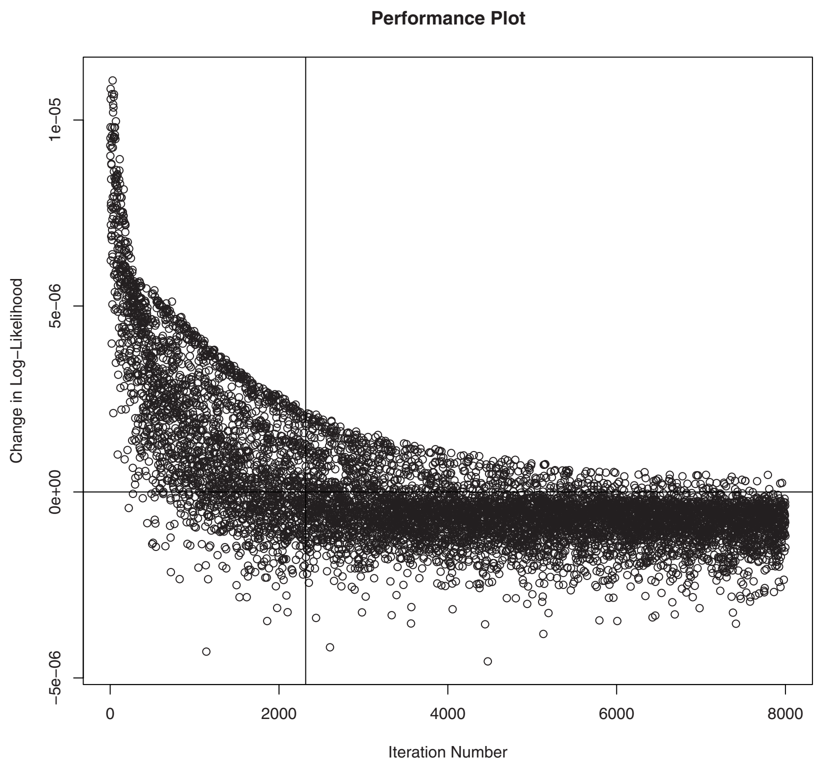

Fig. 6.8. Performance of binomial boosting for very serious inmate misconduct

6.6.3 Homicides and the Impact of Executions

Boosting cannot be expected to improve on conventional Gaussian regression if that regression already fits the data very well. There is nothing to boost. To see how this plays out when the conventional regression already performs with near perfection, and to raise some new issues, we return to the homicides data.

Once again, there are for all 50 states over a 21-year period, the number of homicides per year. As predictors we use the number of executions lagged by one year and then state and year as factors. The key question is whether once one controls for the average number of homicides in a state over the 21 years and the average number of homicides by year over each of the states, the number of executions is related to the number of homicides. For purposes of this illustration, we assume that the number of homicides is conditionally Poisson. Using the generalized additive model we are able to account for well over 95% of the deviance.

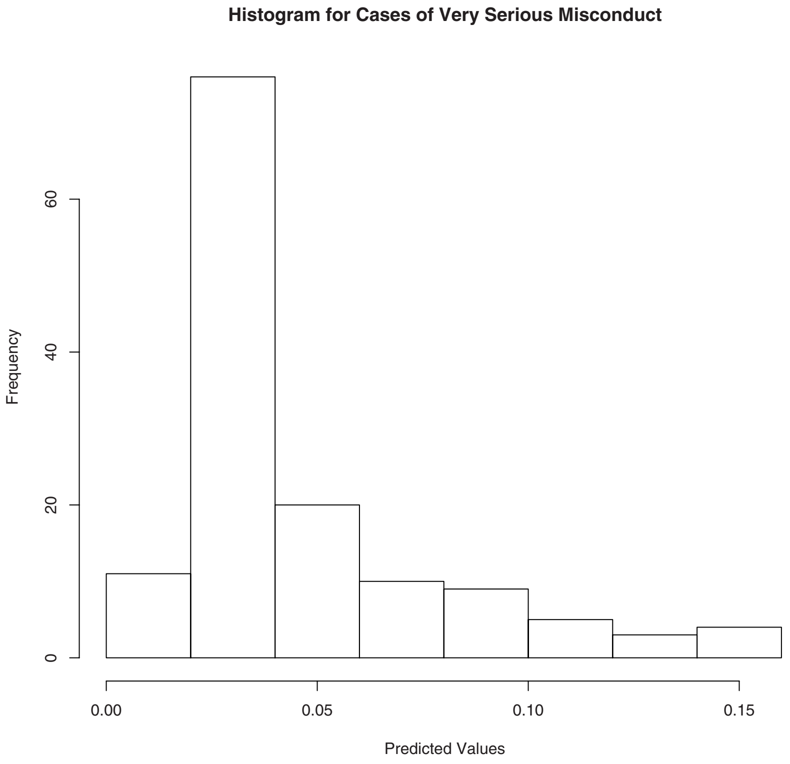

Fig. 6.9. Predicted probability of very serious misconduct.

Figure 6.12 shows how gradient boosting performs in this application. As usual, the changes in the log-likelihood are large early and after about 2000 iterations, the gains are small. Still, the optimal number of iterations is a little less than 10,000.

Figure 6.13 shows that virtually all of the fitting story belongs to the state categorical variable. Its relative contribution is 99.6 out of 100. In contrast, the relative contribution of the number of executions is .008 out of 100.

In Figure 6.14 is plotted the partial dependence of the number of homicides on the number of executions. For less than five executions in a given state in a given year, the relationship is flat. For five or more executions in a given state in a given year, the relationship is negative. But as pointed out earlier, only about 1% of the states have five or more executions in a given year. The apparent evidence for a deterrent effect can only be found where there are almost no data. There is no evidence for deterrence for most states in most years, and too little data to tell when the number of executions is five or more.

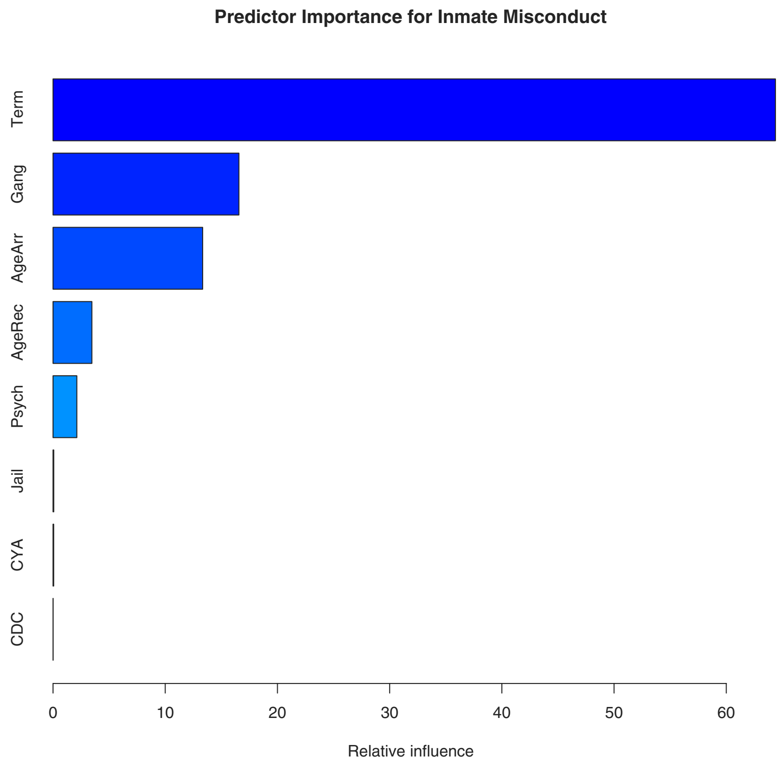

Fig. 6.10. Predictor importance for binomial boosting for very serious inmate misconduct.

A confusion table makes no sense for count data. But we have access to most of the usual regression diagnostics. To begin, the predictors as a group account for about 98% of the deviance. So, the fit is excellent. In addition, Figure 6.15 shows in clockwise order beginning at the upper left 1) the actual number of homicides plotted against the predicted number of homicides, 2) a normal–normal plot of the residuals, 3) a plot of the transformed residuals against the predicted number of homicides, and 4) a plot of Cook's distance by observation number.

From these plots we learn that there are three clumps of fitted values. The two clumps on the right side of the first plot suggest that two smaller subsets of years and/or states may differ from the rest. There is lots of daylight between the clumps. From the first and third plot, we learn that roughly consistent with the Poisson model, the conditional variance of the residuals increases with the conditional mean. However, there is also ample evidence of overdispersion. The second graph indicates that, as one would expect, the

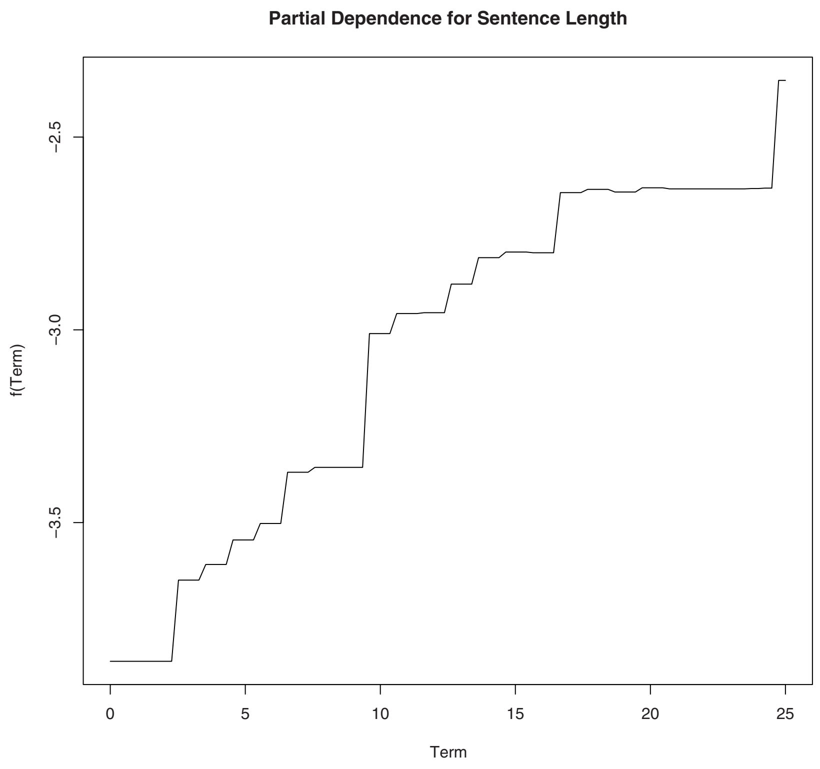

Fig. 6.11. Partial dependence on sentence length in years for binomial boosting for very serious inmate misconduct.

residuals are far from normal, especially at the tails. In fact, the residuals are strongly skewed to the right. This is to be expected given an outcome assumed to be conditionally Poisson. However, the skewing may be linked to a few influential variables. The fourth graph reveals that there are several large influential observations that may well be affecting the fit in significant ways. All of these diagnostics suggest problems with the model, even though most of the variation is accounted for by the predictors.

Figure 6.16 reproduces the partial dependence plot but from data reanalyzed with observations having five executions or more removed. A little more than 1% of the data are lost. Even with only five values for the predictor, the plot is instructive; the plot is a flat straight line. If the response function is even a little more complex, the plot would not have been helpful, even if the computer agreed to construct the plot. There are too few values for the predictor.

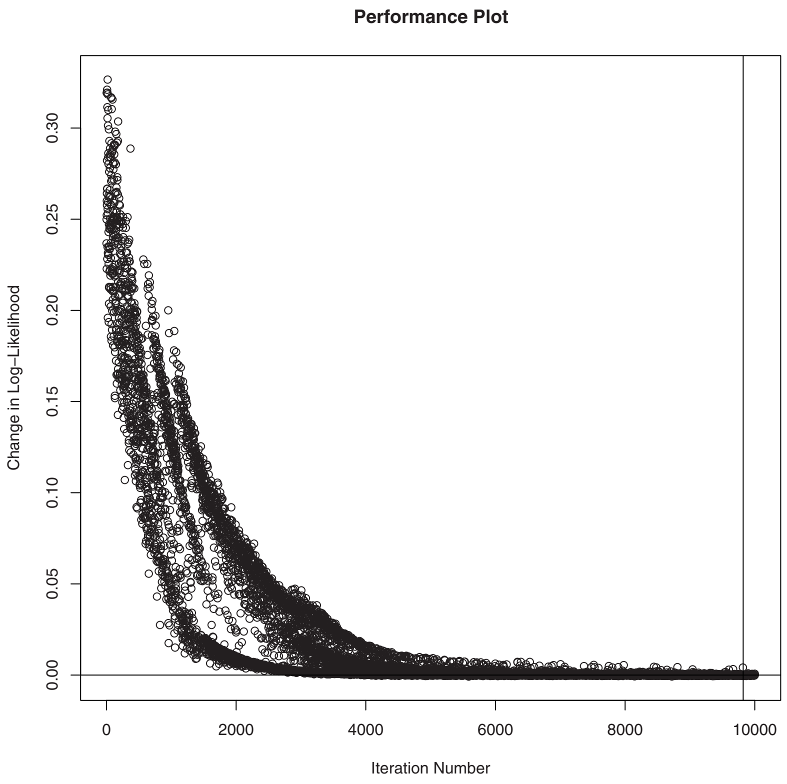

Fig. 6.12. Performance of Poisson boosting for the number of homicides

There is no evidence whatsoever of any deterrence. So, the few influential observations really did affect the boosting results. This is an important lesson. Just as in conventional regression, outliers can make a very big difference. Not surprisingly, the four diagnostic plots (not shown) now look a lot better.

One can in this case obtain virtually the same story using the generalized additive model, including the story about the influential observations. Boosted Poisson regression fits the data slightly better, but not enough to matter; both models fit the data nearly perfectly. And the subject matter conclusions are the same; the gains from boosting compared to parametric regression are slight. The intent in boosting is to take weak predictors and make them strong. There is not much point in boosting predictors that are already very strong.

6.6.4 Imputing the Number of Homeless

Consider again the problem of imputing the number of homeless individuals in Los Angeles County census tracts (Berk et al., 2008). Recall that when

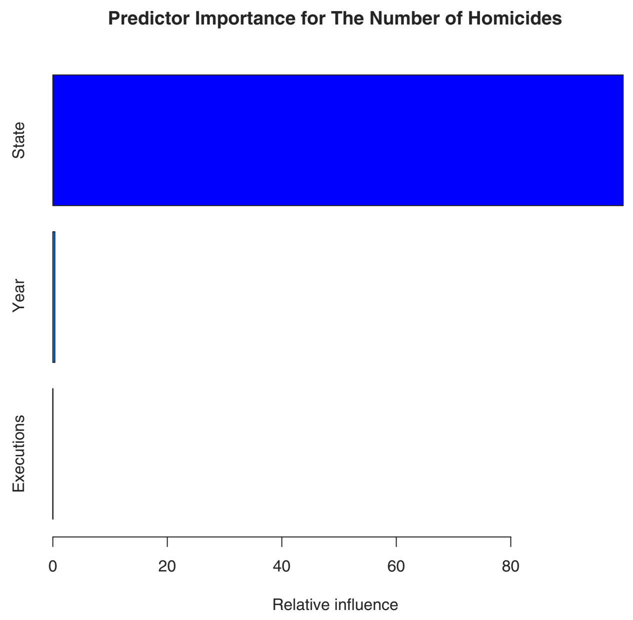

Fig. 6.13. Predictor importance for Poisson boosting for the number of homicides.

random forests was discussed, quantile random forests was applied in order to respond to a few especially high counts. At that time, it was noted that quantile random forests takes the composition of the forest as given, and only adjusts for the summary statistics extracted. A key problem with that approach was that variable importance measures and partial dependence plots were unaltered because they rely on the random forest.

Kriegler (2007) has developed a procedure to weight the loss function in stochastic gradient boosting so that asymmetric costs can be taken into account, not just at the end when summary statistics are constructed, but as the boosting procedure proceeds. Consequently, measures of predictor importance and partial dependence plots are altered accordingly. To date, cost-weighting has been applied to linear (Laplace) loss, Gaussian loss, and Poisson loss, and linear loss has been implemented in gbm().

Figure 6.17 shows for the homeless data, observed street counts plotted again predicted street counts for the weighted linear loss function. Smoothers are overlaid. For the 1/11th quantile, corresponding to weighting overestimates as ten times more costly than underestimates, the fitted values change little, and counts larger than about ten are captured poorly. For the 1/2 quantile, corresponding to equal costs, the overall fit is quite good, but observed counts larger than 50 are not captured well. For the 10/11th quantile, cor-

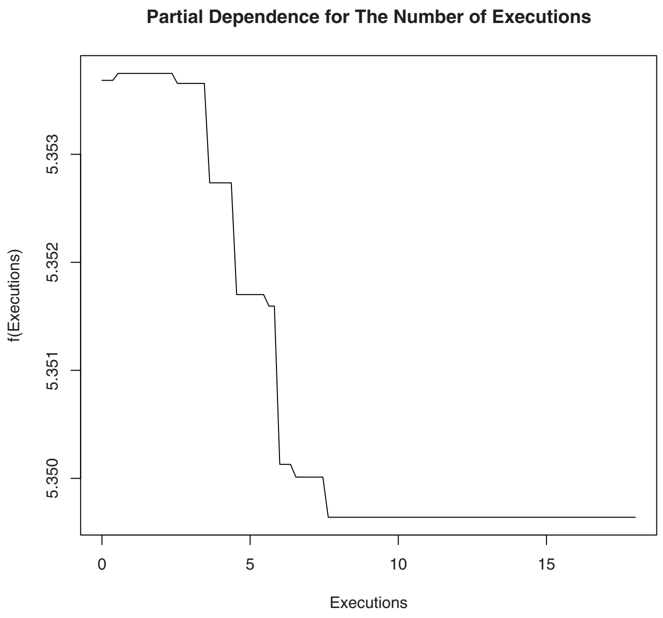

Fig. 6.14. Partial dependence on Executions for Poisson Boosting for the Number of Homicides

responding to costing underestimates as ten times more than overestimates, the overall fit is disappointing, but the larger observed counts are much more effectively fitted. However, still larger relative costs for underestimates would be needed if the very highest observed counts were to be fitted well.

In this instance, the partial dependence plots for key predictors do not change shape materially across different cost ratios. One would tell pretty much the same story about how the predictors are related to the response for a wide range of cost ratios. However, there are some rearrangements of variable importance suggesting that several predictors vary in their forecasting skill depending on which quantile in the response distribution is the target. The details need not concern us here.

As before, there is no statistical answer to the question of which set of fitted values should be preferred. That decision depends on which relative costs are appropriate for the decisions to be made. But it is likely that for counts of the homeless, homeless advocates would see underestimates as far more costly than overestimates. It is less clear what position public officials would take. Larger counts might lead to criticisms to their policies but might also help generate increased funding.

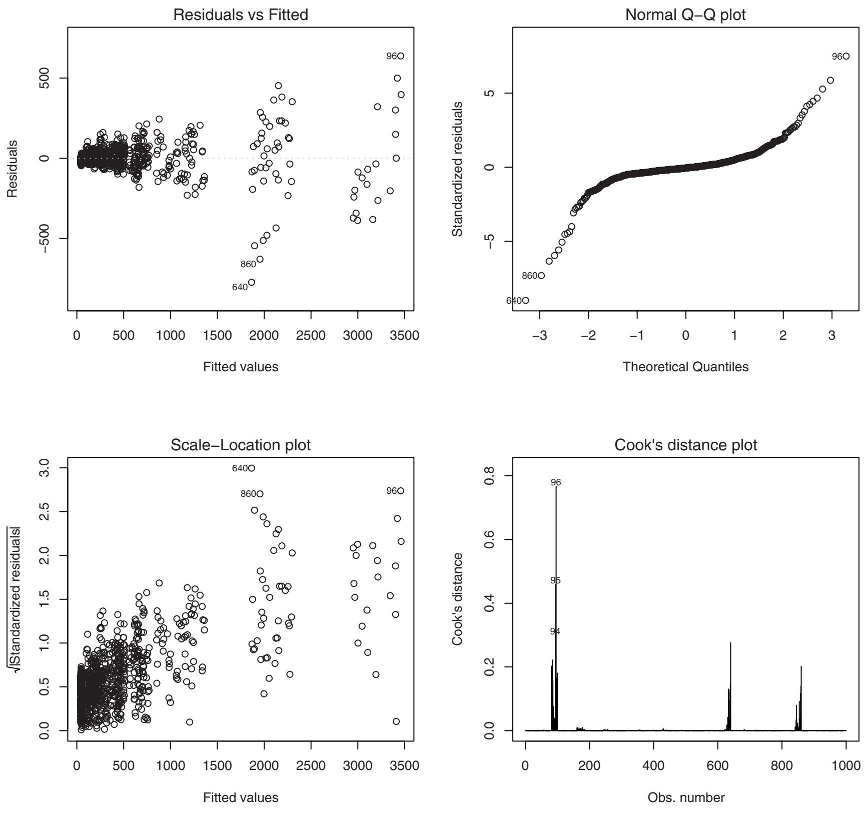

Fig. 6.15. Fit diagnostics for Poisson boosting for the number of homicides.

6.6.5 Estimating Conditional Probabilities

As a final illustration, consider a random sample of American female adults and whether they are in the labor force. The data come from the "Mroz" data in the *car* library in R. The response variable is binary. For this example, the predictors are age, the log of expected wage, household income, and whether there is a child under six in the household.

Two different fit statistics were used to determine when to stop iterating: one based on the OOB data and one based on the data used to build the model. The former indicated that 4000 iterations would be about right. The latter indicated that 10,000 iterations would be about right.

Both stopping rules led to 72% of the observations being correctly classified; classification accuracy was virtually identical. However, the fitted proportions that might be used as estimates of conditional probabilities differed

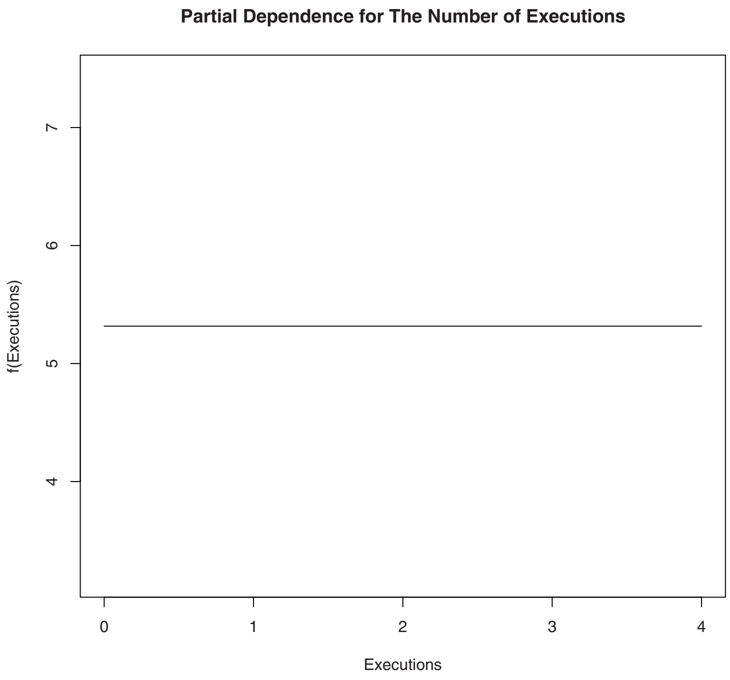

Fig. 6.16. Partial dependence on executions for Poisson boosting for the number of homicides—outliers excluded.

substantially. Figure 6.18 is a scatterplot of the two sets of fitted proportions with a 1-to-1 line overlaid. Because of the stochastic content in the algorithm, the two sets of fitted proportions cannot be exactly the same, but they should cluster tightly around the 1-to-1 lines. The plot shows that fitted proportions are significantly more spread out when there are 10,000 iterations rather than 4000 iterations. On the average the two sets of values differ most at the tails, especially the lower tail.

There is some craft lore arguing that determining the number of iterations using the OOB data can lead to underfitting, which implies that the fitted proportions based on 10,000 iterations should be preferred. Even if this is true, the sensitivity of the fitted proportions to at least one tuning parameter when classification skill is nearly the same, is a bit unsettling. And although craft lore can be very helpful, it comes with no guarantees. If there is a keen interest in the fitted proportions, perhaps as estimates of conditional probabilities, it may be important to consider using Jous-Boost, described briefly earlier (Mease et al., 2007). .

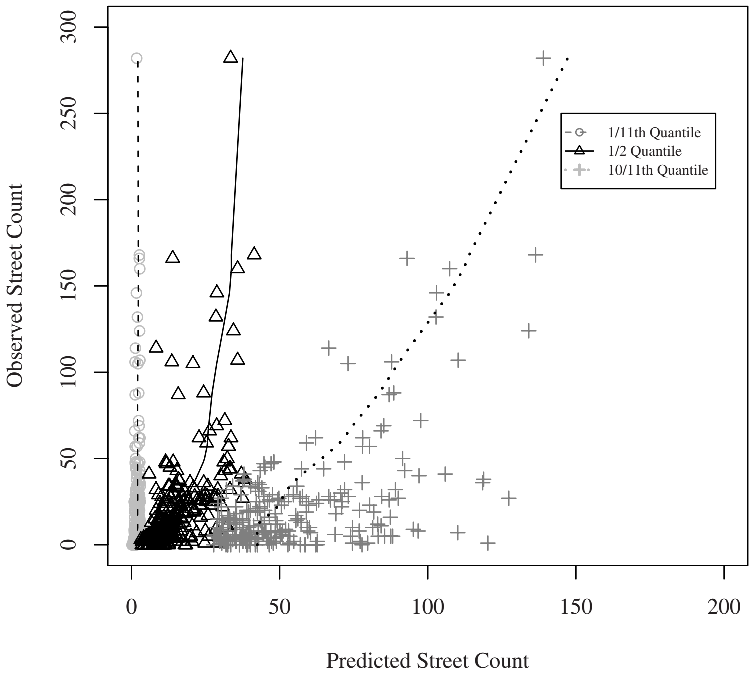

Fig. 6.17. Observed and fitted values for different quantiles.

6.7 Software Considerations

As noted earlier, boosting is in a great state of flux, and nowhere is this more evident than in the software available. The boosting analyses reported in this chapter were done with the procedure gbm() in R. It performs very well using gradient boosting, allowing for a wide variety of loss functions. It also has a number of useful tuning parameters and several helpful forms of graphical output. The name "gbm" stands for Generalized Boosted Models. The package is written by Greg Ridgeway, who also is the maintainer (gregr@rand.org).

There are several other boosting procedures in R. The procedure mboost(), for example, does gradient boosting for the generalized linear model and the generalized additive model. GAMBoost() boosts the generalized additive model using likelihood based approaches. The procedures boost() and ada() implement Adaboost, Logitboost, and other classification procedures. To date, an important advantage that gbm() has over the alternatives in R

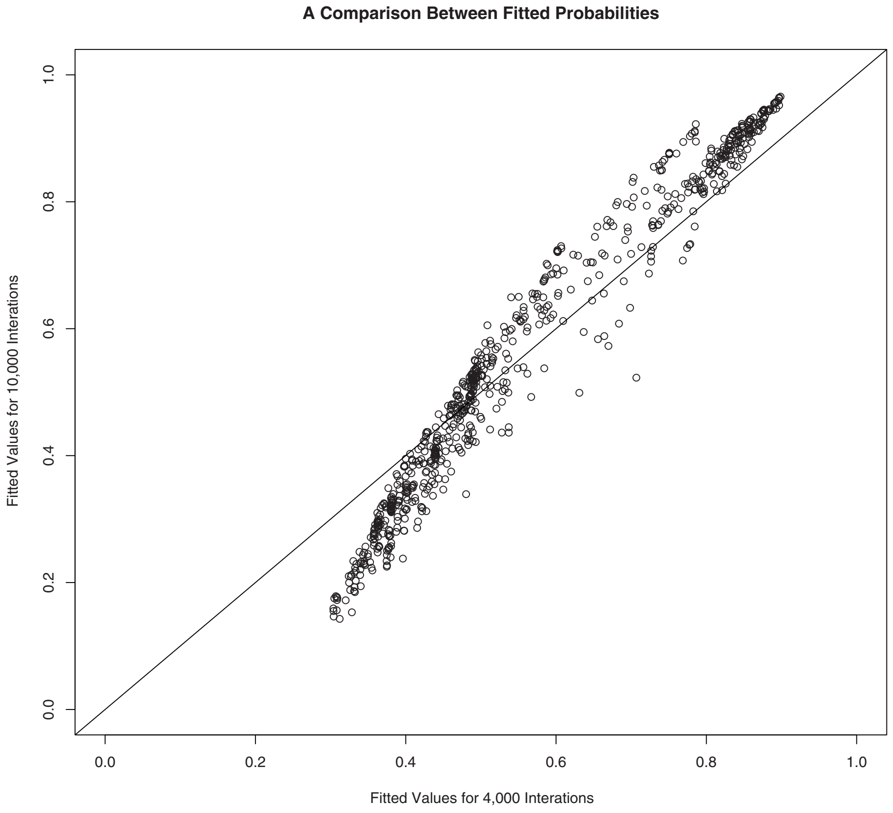

Fig. 6.18. Fitted proportions for two different stopping rules.

is a better range of outputs. The documentation that goes with gbm() is also especially good.

The only visible private sector provider currently is, once again, Salford Systems. Salford Systems offers a program called Multiple Additive Trees (MART), which is essentially gradient boosting. As with all Salford System products, the user interface is very friendly and the documentation excellent. But the price is substantial and the advertising hype a bit offputting.

6.8 Summary and Conclusions

Boosting is a very slick approach to statistical learning. The underlying concepts are interesting and their use to date creative. Boosting has also stimulated very productive interactions among researchers in statistics, applied mathematics, and computer science. Perhaps most important, boosting has been shown to be very effective for certain kinds of data analysis.

However, there are important limitations to keep in mind. First, boosting is designed to improve the performance of weak predictors. Trying to boost predictors that are already strong is not likely to be productive. A set of strong predictors can lead to an effective fit within a conventional regression model. Then, the residuals are essentially noise. Then, there is no more information to extract.

Unfortunately, there is no convincing way from the data alone to know if the residuals really lack any systematic information. But if the list of variables represents all the predictors known to be important, if these predictors are well measured, and if the partial response plots are consistent with widely accepted and detailed theory, the chances are good that boosting will not help much.

Second, if the goal is to fit conditional probabilities, boosting can be a risky way to go. One useful alternative was discussed, but it has yet to be extensively field-tested with real data. It follows that calculations using the fitted probabilities can be highly suspect.

Third, boosting is not alchemy. Boosting can improve the performance of many weak fitting procedures, but the improvements may fall far short of the performance needed. Boosting cannot overcome variables that are measured poorly or important predictors that have been overlooked. The moral is that (even) boosting cannot overcome a seriously flawed measurement and badly executed data collection. The same applies to all of the statistical learning procedures discussed in this book.

Finally, when compared to other statistical learning procedures, especially random forests, boosting will often allow for a wider range of applications, and for the same kinds of applications, perform competitively. In addition, its clear links to common and well-understood statistical procedures can help make boosting understandable. However, boosting's usual reliance on symmetrical loss functions is a major difficulty, especially for research results that will be used to inform practical decisions and broader public policy.

Exercises

Problem Set 1

Generate the following data. The systematic component of the response variable is quadratic. If 1000 observations are too large for your computer to easily handle, work with a sample of 500 observations.

```
x1=rnorm(1000)
x12=x1^2
```

```
ysys=1+(-5*x12)
y=ysys+(5*rnorm(1000))
dta=data.frame(y,x1,x12)
```

1. Plot the the systematic part of y against the predictor $x1$. This represents the $f(x)$ you are trying to recover. Plot y against $x1$. This represents the data to be analyzed. Why do they look different?

2. Apply gbm() to the data. There are a lot of tuning parameters and parameters that need to be set for later output so, here is some code to get you started.

   ```
   out<-gbm(y~x1,distribution="gaussian",n.trees=10000,
        data=dta1,cv.folds=5)
   gbm.perf(out,method="cv")
   ```

   Construct the partial dependence plot using

   ```
   plot(out,n.trees=???)
   ```

   where the ??? is the number of trees, which is the same as the number of iterations. Make five plots, one each of the following number of iterations: 100, 500, 1000, 5000, and the number recommended by the cross-validiation method in the second step above. Study the sequence of plots and compare them to the plot of the true $f(X)$. What happens to the plots as the number of iterations approaches the recommended number? Why does this happen?

Problem Set 2

From the *car* library load the data "Leinhardt." Analyze the data using gbm(). The response variable is infant mortality.

1. Plot the performance of gbm(). Interpret the two lines that are plotted and explain what their divergence implies.

2. What is the recommended number of iterations?

3. Construct a graph of the importance of the predictors. Which variables seem to affect the fit substantially and which do not?

4. Construct the marginal partial dependence plot for each predictor Interpret each plot.

5. Construct all of the two-variable plots (see examples in help(gbm)). Interpret each plot.

6. Construct the three-variable plot (see examples in help(gbm)) Interpret the plot.

7. Consider the quality of the fit. How large is the improvement compared to when no predictors are used?

8. Write a paragraph or so, on what the analysis of these data has revealed about correlates of infant mortality at a national level.

9. Now repeat the analysis using random forests. How do the results compare to the results from stochastic gradient boosting? Would you have arrived at substantially different conclusions depending on whether you used random forests or stochastic gradient boosting?

Problem Set 3

From the *MASS* library, analyze the dataset called Pima.tr. The outcome is binary: diabetes or not (coded as "Yes" and "No"). Assume that the costs of failing to identify someone who has diabetes are three times higher than the costs of falsely identifying someone who has diabetes. The predictors are all of the other variables in the dataset.

The goal is to analyze these data using several different procedures and then make comparisons across the results. The statistical procedures to compare are logistic regression, the generalized additive model, random forests, and stochastic gradient boosting. You will need to make a number of decisions so that the methods are as comparable as possible (e.g., what loss function to use for stochastic gradient boosting). But also feel free to try several different versions of each procedure (e.g., "Adaboost" v. "bernoulli" for stochastic gradient boosting).

1. Construct confusion tables for each model. Be alert for whether the fitted values are for "resubstituted" data. Do some procedures fit the data better than others? Why or why not?

2. Cross-tabulate the fitted values for each model against the fitted values for each other model. How do the sets of fitted values compare?

3. Compare the "importance" assigned to each predictor. This is tricky. For example, how can sensible comparisons be made between the output of a logistic regression and the output of random forests?

4. Compare partial response functions. This too is tricky. For example, what can you do with logistic regression?

5. If you had to make a choice to use one of these procedures, which would you select? Why?