# Chapter 6 Boosting

## 6.1 Introduction

As already discussed, one of the reasons why random forests is so effective for a complex $f(\mathbf{X})$ is that it capitalizes interpolation. As a result, it can respond to highly local features of the data in a robust manner. Such flexibility is desirable because it can substantially reduce the bias in fitted values. But the flexibility usually comes at a price: the risk of overfitting. Random forests consciously addresses overfitting using OOB observations to construct the fitted values and measures of fit, and by averaging over trees. The former provides ready-made test data while the latter is a form of regularization. Experience to date suggests that this two-part strategy can be highly effective.

But the two-part strategy, broadly conceived, can be implemented in other ways. Some argue that 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 averaged 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."

Boosting as originally conceived 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. 2009: 337). So, boosting has some of the same look and feel as random forests.

The original version of this chapter was revised: See the "Chapter Note" section at the end of this chapter for details. The erratum to this chapter is available at https://doi.org/10.1007/978-3-319-44048-4_10.

R.A. Berk, *Statistical Learning from a Regression Perspective*,
Springer Texts in Statistics, DOI 10.1007/978-3-319-44048-4_6

But, boosting as initially formulated differs from random forests in at least five 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 more 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 linear combination over a large set of earlier fitting attempts. But the combination is not a simple average as in random forests. The fitted values are weighted in a manner to be described shortly. Fourth, the fitted values and measures of fit quality are usually constructed from the data in-sample. There are no out-of-bag observations, although some recent developments make that an option. Finally, although small trees can be used as weak learners, boosting is not limited to an ensemble of classification and regression trees.

To appreciate how these pieces can fit together, we turn to Adaboost.M1 (Freund and Schapire 1996; 1997), which is perhaps the earliest and the most widely known boosting procedure. 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

We will treat Adaboost.M1 as the poster child for boosting in part because it provides such 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.M1 then has the following general structure. The pseudocode that follows is basically a reproduction of what Hastie et al. (2009) show on their page 339.

```
1. Initialize the observation weights $w_i = 1/N, i = 1, 2, \ldots, N$ observations.
2. For $m = 1$ to $M$ passes over the data:
    (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 $x$. It is the source of the fitted values used in the algorithm. 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.

Classification error for pass $m$ over the data is denoted by $err_m$; it is essentially the proportion of cases misclassified. In the next step, $err_m$ is in the denominator and $(1 - err_m)$ is in the numerator before the log is taken. A larger value of $\alpha_m$ means a better fit.

The new weights, one for each case, are then computed, The value of $w_i$ is unchanged if the $i$th case is correctly classified. If the $i$th case is incorrectly classified, it is "up-weighted" by $e^{\alpha_m}$. Adaboost.M1 will pay relatively more attention in the next iteration to the cases that were misclassified. In some expositions of Adaboost (Freund and Schapire 1999), $\alpha_m$ is defined as $\frac{1}{2}\log(1-err_m/err_m)$. Then, incorrectly classified cases are up-weighted by $e^{\alpha_m}$ and correctly classified cases are down-weighted by $e^{-\alpha_m}$.

In the final step, classification is determined by a sum of fitted values over the $M$ classifiers $G_m$, with each set of fitted values weighted by $\alpha_m$. This is in much the same spirit as the last step in the random forest algorithm, but for adaboost, the contributions from classifiers that fit the data are better and are given more weight, and the class assigned depends on the sign of the sum.

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 sum. 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, but their forms and purposes are quite different.

### *6.2.1 A Toy Numerical Example of Adaboost.M1*

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 from some classifier 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. Therefore, 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 then

$$\alpha_1 = \log\frac{(1 - .40)}{.40} = \log(.60/.40) = \log(1.5) = .41.$$

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 three and the fifth are correct. The fourth is incorrect. The error for the second iteration is

$$err_2 = \frac{[(.20 \times 0) + (.20 \times 0) + (.20 \times 0) + (.30 \times 1) + (.30 \times 0)]}{1.2} = .25$$

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 sum 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. The key point, however, is that operationally, there is nothing very mysterious going on.

### *6.2.2 Why Does Boosting Work so Well for Classification?*

Despite the operational simplicity of boosting, there is no consensus on why it works so well. At present, there seem to be three complementary perspectives. All three are truly interesting, and each has some useful implications for practice.

#### 6.2.2.1 Boosting as a Margin Maximizer

The first perspective comes from computer science and the boosting pioneers. The basic idea is that boosting is a margin maximizer (Schapire et al. 1998; Schapire and Freund 2012). From AdaBoost, the margin for any given case $i$ is defined as

$$mg^* = \sum_{y=G_m} \alpha_m - \sum_{y \neq G_m} \alpha_m, \tag{6.1}$$

where the sums are over the number of passes through the data for correct classifications or incorrect classifications respectively. This expression is different from Breiman's margin ($mr$) but in the same spirit. In words, for any given case $i$, the margin is the difference between the sum of the iteration weights when the classification is correct and the sum of the iteration weights when the classification is incorrect.[1]

Looking back at the toy example, the classifier is two for two for the first three observations, one for two for the last observation, and zero for two for the fourth observation. In practice, there would be hundreds of passes (or more) over the data, but one can nevertheless appreciate that the classifications are most convincing for the first three observations and least convincing for the fourth observation. The fifth observation is in between. Equation 6.1 is just an extension of this idea. The sum of the correct or incorrect classifications becomes the sum of the weights when the classification is correct or the sum of the weights when the classification is incorrect, with the weights equal to $\alpha_m$. For the first three cases in the toy example, the margin $(.41 + 1.1) - 0 = 1.51$. The margin for the fourth case is $0 - (.41 + 1.1) = -1.51$. The margin for the fifth case is $1.1 - .41 = .69$. The evidence for the first three cases is the highest. The evidence for the fourth case is the lowest, and the evidence for the fifth case is in between.

And now the punch line. "Boosting is particularly good at finding classifiers with large margins in that it concentrates on those examples whose margins are small (or negative) and forces the base learning algorithm to generate good classifications for those examples" (Schapire et al. 1998: 1656).[2] Indeed, as the number of passes through the data increase, the margins over observations generally increase, although

[1] A more general definition is provided by Schapire and his colleagues (2008: 1697). A wonderfully rich and more recent discussion about the central role of margins in boosting can be found in Schapire and Freund (2012). The book is a remarkable mix from a computer science perspective of the formal mathematics and very accessible discussions of what the mathematics means.

[2] In computer science parlance, an "example" is an observation or case.

whether they are maximized depends on the base classifier. For instance, the margins are generally not maximized for stump classification trees but are maximized for large classification trees. This should sound familiar. A closely related point was made for random forests.

Improvements in the margin over passes through the data reformulates the overfitting problem. It is possible in principle to fit the training data perfectly. One would ordinarily halt the boosting well before that point because of concerns about overfitting. But with a perfect fit of the data, generalization error in test data can be surprisingly good because the weighed averaging works in a manner much like the averaging in bagging or random forests. (Look again at the final step in the Adaboost.M1 algorithm.) Moreover, boosting *past* a perfect fit of the data can further reduce generalization error because the margins are getting larger. In short, concerns about overfitting for boosting seem to have been overstated.

There is one exception in which overfitting has actually been understated. When a classifier is also used to compute the probabilities associated with each class, boosting to minimize generalization error pushes the conditional proportions/probabilities for each observation toward 0.0 or 1.0 (Mease et al. 2007; Buja et al. 2008). This follows from the margin maximizing property of boosting. We will have more to say about this shortly.

In summary, one reason why boosting works so well as a classifier is that it proceeds as a margin maximizer. An important implication for using classifiers such as Adaboost.M1 is that overfitting can in practice not be a serious problem. One can even boost past the point at which the fit in the training data is perfect. Another important implication for practice is that if classification trees are used as the classifier, large trees are desirable. And if large trees are desirable, so are large samples.

#### 6.2.2.2 Boosting as a Statistical Optimizer

The second perspective sees Adaboost as a stagewise additive model using basis functions in much the same spirit as CART and random forests. Consider again the final step in the algorithm for Adaboost.M1

$$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]. \tag{6.2}$$

Each pass through the data involves the application of a classifier $G_m(x)$, the culmination of which is a stage. The results of $M$ stages are combined in an additive fashion with the $M$ values of $\alpha_m$ as the computed weights. This means that each $G_m(x)$ relies on a linear basis expansion of $X$, much as discussed in Chap. 1.

If boosting can be formulated as a stagewise additive model, an important question is what loss function is being used. From Friedman and his colleagues (2000), Adaboost.M1 iterations are implicitly targeting

$$f(X) = \frac{1}{2}\log\frac{\mathrm{P}(Y = 1|X)}{\mathrm{P}(Y = -1|X)}. \tag{6.3}$$

This is just one-half of the usual log-odds (logit) function for $\mathrm{P}(Y = 1|X)$. The $1/2$ results from using the sign to determine the class. This is the "population minimizer" for an exponential loss function $e^{-yf(x)}$. More formally,

$$\arg\min_{f(x)} \mathrm{E}_{Y|x}(e^{-Yf(x)}) = \frac{1}{2}(\log)\frac{\mathrm{P(Y = 1|x)}}{\mathrm{P(Y = -1|x)}} \tag{6.4}$$

Adaboost.M1 is attempting to minimize exponential loss with the observed class and the predicted class as its arguments. The focus on exponential loss raises at least two important issues for practice.

First, emphasizing the mathematical relationship between the exponential loss function and conditional probabilities can paper over a key point in practice. Although at each stage, the true conditional probability is indeed the minimizer, over stages there can be gross overfitting of the estimated probabilities. In other words, the margin maximizing feature of boosting can trump the loss function optimizing feature of boosting.

Second, a focus on the exponential loss function naturally raises the question of whether there are other loss functions that might perform better. Hastie and his colleagues (2009: 345–346) show that minimizing the 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 (i.e., a level II or level I analysis respectively). 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. (2009: 346–349) 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 of the response variable are likely to be systematically wrong or noisy. There are other boosting options as well (Friedman et al. 2000). But, it is not clear how the various competitors fare in practice and for our purposes at the moment, that is beside the point. The loss function optimization explanation, whatever the proposed loss function, is at best a partial explanation for the success of boosting.

#### 6.2.2.3 Boosting as an Interpolator

The third perspective has already been introduced. One key to the success of random forests is that it is an interpolator that is then locally robust. Another key is the averaging over a large number of trees. Although the details certainly differ, these attributes also apply to boosting (Wyner et al. 2015). As the margins are increased, the fitted values better approximate an interpolation of the data. Then, the weighted average of fitted values provides much the same stability as the vote over trees provides for random forests; "…the additional iterations in boosting way beyond the point at which perfect classification in the training data (i.e., interpolation) has occurred has the effect of smoothing out the effects of noise rather than leading to more and more overfitting" (Wyner et al. 2105: 24).

All three perspectives help explain why boosting performs so well as a classifier. From a statistical perspective, boosting is a stagewise optimizer targeting the same kinds of conditional probabilities that are the target for logistic regression. One of several different loss functions can be used depending on the details of the data. This framework places boosting squarely within statistical traditions. But boosting is far more than a round-about way to do logistic regression. The margin maximization perspective helps to explain why and dramatically reduces concerns about overfitting, at least for classification. When the classifiers are trees, large trees perform better, which suggests that in general, complex base classifiers are to be preferred. Finally, the interpolation perspective links boosting to random forests and shows that boosting classifiers have many of the same beneficial properties. In so doing, there is a deeper understanding about why maximizing margins can be so helpful, although it is not nearly the whole story. In the end, interpolation may be the key.

The implications for classification in practice are fourfold

1. complex base learners (e.g., large classification trees) help;
2. boosting beyond a perfect fit in the training data can help;
3. a large number of observations can help; and
4. a rich set of predictors can help (subject to the usual caveats such as very high multicollinearity).

## 6.3 Stochastic Gradient Boosting

At present, there are many different kinds of boosting that all but boosting mavens will find overwhelming. Moreover, there is very little guidance about which form of boosting should be used in which circumstances. For practitioners, therefore, stochastic gradient boosting is a major advance (Friedman 2001; 2002). It is not quite a one-size-fits-all boosting procedure, but within a single statistical framework provides a rich menu of options. As such, it follows directly from the statistical perspective on boosting.

Here is a rough summary of the procedure for classification. Suppose that the response variable in the training data is binary and coded as 1 or 0. The procedure is initialized with some constant such as the overall proportion of 1s. This constant serves as the fitted values from which residuals are obtained by subtraction in the usual way. The residuals are then appended to the training data as a new variable. Next, a random sample of the data is drawn without replacement. One might, for example, sample half the data. A regression tree, not a classification tree, is applied to the sample with the residuals as the response. Another set of fitted values is obtained.

From these, a new set of residuals is obtained and appended. Another random sample is taken from the training data and the fitting process is repeated. The entire cycle is repeated many times: (1) fitted values, (2) residuals, (3) sampling, (4) a regression tree. In the end, the fitted values from each pass through the data are combined in a linear fashion. For classification, these can be interpreted as proportions or probabilities depending on whether the analysis is at level I or level II respectively. 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.

The weighting so central to boosting occurs implicitly through the residuals from each pass. Larger positive or negative residuals imply that for those observations, the fitted values are less successful. As the regression tree attempts to maximize the quality of the fit overall, it responds more to the observations with larger positive or negative residuals.

Consider now somewhat more formally the sources of the term "gradient" in stochastic gradient boosting. Either numerical or categorical response variables are allowed along with a variety of loss functions. As with random forests, trees are a key component of the algorithm. The discussion that follows on boosting trees draws heavily on Ridgeway (1999) and on Hastie et al. (2009: Sects. 10.9–4.10).

A *given* tree can be represented as

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j), \tag{6.5}$$

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 $j$th terminal node, and $\gamma_j$ is the value assigned to each observation in the $j$th terminal node. The goal is to construct values for the unknown parameters $\Theta$ so that the loss function is minimized. At this point, no particular loss $L$ is specified, and we seek

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j). \tag{6.6}$$

As noted earlier, minimizing the loss function for a single tree is challenging. For stochastic gradient boosting, the challenge is even greater because we seek to minimize the loss over a set of trees. As a rough-and-ready approximation, we once again proceed in a stagewise fashion so that at iteration $m$, we need to find

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)), \tag{6.7}$$

where $f_{m-1}(x_i)$ are the results 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, Eq. 6.7 expresses the aspiration of updating the fitted values in an optimal manner.

Equation 6.7 can be reformulated as a numerical optimization task. In this framework, $g_{im}$ is the gradient for the $i$th observation on iteration $m$, defined as the partial derivative of the loss with respect to the fitting function. Thus,

$$g_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}. \tag{6.8}$$

Equation 6.8 represents for each observation $i$ 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 more to the larger absolute values of $g_{im}$ than small ones.

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" $\rho_m$ is found so that

$$\rho_m = \arg\min_{\rho} L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m). \tag{6.9}$$

In other words, a scalar $\rho_m$ 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.

Figure 6.1 illustrates the basics of the process when there are two predictors, $X_1$ and $X_2$. There is a single location, represented by the red filled circle, where the smallest loss can be found. The algorithm starts at some arbitrary point and proceeds in steps determined by the direction and step length for which the loss is reduced the most, subject to a constraint on the length of the step. Often the step lengths are reduced in flatter regions of the function so that the minimum is not overshot. Typically there will be many more predictors, but the ideas represented in Fig. 6.1 generalize.

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. 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) shows that if one uses certain transformations of the disparities as the gradients (details soon), there is a least squares solution to finding effective parameter values for the fitting function. That is,

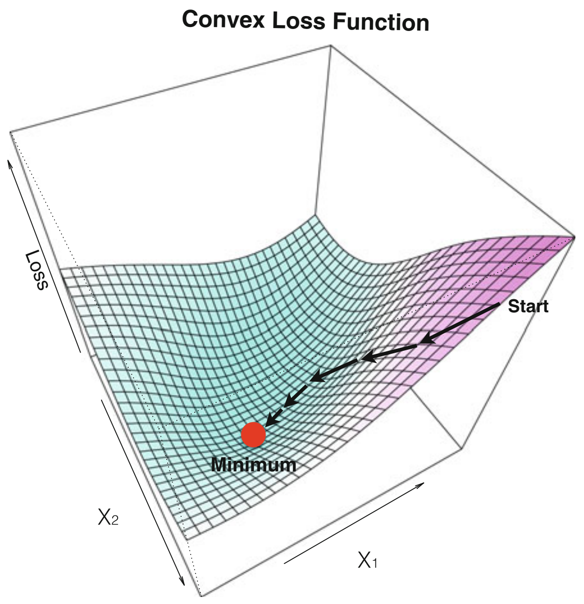

**Fig. 6.1** An illustration of steepest descent looking down into a convex loss function (The predictors are $X_1$ and $X_2$. Loss is represented on the vertical axis.)

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i; \Theta))^2. \tag{6.10}$$

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, can be much like Adaboost or Adaboost itself. Moreover, by recasting the boosting process in gradient terms, many useful variants follow including the boosting of fitting procedures for quantitative response variables.

We turn, then, to stochastic gradient boosting, implemented in R as *gbm()*, that is a generalization of Friedman's original gradient boosting. Among the differences are the use of sampling in the spirit of bagging and a form of shrinkage.[3]

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

[3]For very large datasets, there is a scalable version of tree boosting called *XGBoost()* (Chen and Guestrin 2016) that can provide remarkable speed improvements but has yet to include the range of useful loss functions found in *gbm()*. More will be said about *XGBoost()* when "deep learning" is briefly considered in Chap. 8.

1. Initialize $f_0(x)$ so that the constant $\kappa$ minimizes the loss function: $f_0(x) = \arg\min_{\kappa} \sum_{i=1}^{N} L(y_i, \kappa)$.[4]
2. For $m$ in $1, \ldots, M$, do steps a–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 $W$ cases from the data set, where $W$ is less than the total number of observations. This is a simple random sample, not a bootstrap sample, which seems to improve performance. How large $W$ should be is discussed shortly.

   (c) Using the randomly selected observations, each with their own $r_{im}$, fit a regression tree with $J_m$ terminal nodes to the $r_{im}$, 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 $R_{jm}$ denotes the set of $x$-values that define the terminal node $j$ for iteration $m$.

   (e) Drop all of the cases down the tree grown from the sample and, 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 $\nu$ is a "shrinkage" parameter that determines the learning rate. The importance of $\nu$ is discussed shortly.

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

Ridgeway (1999) has shown that 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: Sect. 10.10). The trick is determining the right $r_i$ for each special case; the "residuals" need to be defined. Among the current definitions of $r_{im}$ are the following, each associated with a particular kind of regression mean function: linear regression, logistic regression, robust regression, Poisson regression, quantile regression, and others.

[4]Other initializations such as least squares regression could be used, depending on loss function (e.g., for a quantitative response variable).

1. Gaussian: $y_i - f(x_i)$: the usual regression residual.
2. Bernoulli: $y_i - \frac{1}{1+e^{-f(x_i)}}$: the difference between the binary outcome coded 1 or 0 and the fitted ("predicted") proportion for the conventional logit link function for logistic regression.
3. Poisson: $y_i - e^{f(x_i)}$: the difference between the observed count and the fitted count for the conventional log link function as in Poisson regression
4. Laplace: sign$[y_i - f(x_i)]$: the sign of the difference between the values of the response variable and the fitted medians, a form of robust regression.
5. Adaboost: $-(2y_i - 1)e^{-(2y_i-1)f(x_i)}$: based on exponential loss and closely related to logistic regression.
6. Quantile: $\alpha(I(y > f(x_i)) - (1-\alpha)I(y_i \leq f(x_i))$: for quantile regression the weighted difference between two indicator variables equal to 1 or 0 depending on whether residual is positive or negative with the weights $\alpha$ or $(1-\alpha)$, and $\alpha$ as the percentile target.

There are several other options built into *gbm()*. Hastie and his colleagues (2001: 321) derive the gradient for a Huber robust regression. Ridgeway (2012) offers boosted proportional hazard regression. Other less well-documented options include multinomial logistic regression, and regression based on a t-distribution. No doubt there will additional distributions added in the future.

Stochastic gradient boosting also can be linked to various kinds of penalized regression of the general form discussed in earlier chapters. One insight is that the sequence of results that is 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).

In short, with stochastic gradient boosting, each tree is constructed much as a conventional regression tree. The difference is how the "target" for the fitting is defined. Using disparities defined in particular ways, a wide range of fitting procedures can be boosted. It is with good reason that "gbm" stands for "generalized boosted regression models."

### *6.3.1 Tuning Parameters*

Stochastic gradient boosting has a substantial number of tuning parameters, many of which affect the results in similar ways. There is no analytical way to arrive at an optimal tuning parameter values in part because how they perform is so dependent on the data (Buja et al. 2008). An algorithmic search over values might be helpful in principle, but would be computationally demanding, and there would likely be many sets of tuning parameter values leading to nearly the same performance. Fortunately, the results from stochastic gradient boosting are often relatively robust with respect to sensible variation in the tuning parameters, and common defaults usually work quite well.

The most important tuning parameters provided by *gbm()* are as follows:

1. Number of Iterations — The number of passes through the data is typically the most important tuning parameter and is in practice empirically determined. Because there is no convergence and no clear stopping rule, the usual procedure 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. The error should decline rapidly at first and then level off. If after leveling, there is an inflection point at which the fitting error begins to increase, the number of iterations can be stopped shortly before that point. If there is no inflection point, the number of iterations can be determined by when reductions in the error effectively cease. There is a relatively large margin for error because plus or minus 50–100 iterations rarely lead to meaningful performance differences.

2. Subsample Size — 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). When sampling without replacement, the sample size is a tuning parameter, and the issues are rather like those that arise when one works with split samples. How large should the training sample, evaluation sample, and test sample be? There seems to be no formal or general answer. Practice seems to favor a sample size of $N/2$. But it can make sense for any given data analysis to try sample sizes that also are about 25 % smaller and larger.

3. Learning Rate — A slow rate at which the updating occurs can be very useful. Setting the tuning parameter $\nu$ to be less than 1.0 is standard practice. A value of .001 often seems to work reasonably well, but values larger and smaller by up to a factor of 10 are sometimes worth trying as well. By slowing down the rate at which the algorithm "learns," a larger number of basis functions can be computed. Flexibility in the fitting process is increased, and the small steps increase shrinkage, which improves stability. 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.

4. Interaction Depth — Another tuning parameter that affects the flexibility of the fitting function is the "depth" of the interaction. This name is a little misleading because it does not directly control the order of the interactions allowed. Rather it controls the number of splits allowed. If the interaction depth is 1, there is only a split of the root node data. If the interaction depth is 2, the two resulting data partitions of the root node data are split. If the interaction depth is 3, the four resulting partitions are split. And on it goes. Interaction depth is a way to limit the size of the regression trees, and values from 1 to 10 are used in practice. As such, the interaction depth determines the *maximum* order of any interactions, but the order of the interactions can be less than the interaction depth. For example, an interaction depth of 2 may result in four partitions defined

by a single predictor at different break points. There are no interactions effects because interaction effects are commonly defined as the product of two or more predictors. If interaction depth is set to 2, the largest possible interaction effect is 2 (i.e., a two-way interaction involving two predictors).

5. Terminal Node Size — Yet another tuning parameter that affects fitting function flexibility is the minimum number of observations in each tree's terminal node. For a given sample size, smaller node sizes imply larger trees and a more flexible fitting function. But smaller nodes also lead to less stability for whatever is computed in each terminal node. Minimum terminal node sizes of between 5 and 15 seem to work reasonably well in many settings, but a lot depends on the loss function that is being used.

In practice, the tuning parameters can interact. For example, terminal node size may be set too high for the interaction depth specified to be fully implemented. Also, more than one tuning parameter can be set in service of the same goal. The growth of larger trees, for instance, can be encouraged by small terminal node sizes and by greater interaction depth. In short, sometimes tuning parameters compete with one another and sometimes tuning parameters complement one another.

How one tunes depends heavily on the kind of stochastic gradient boosting being undertaken. The advice available typically depends on craft lore, but the interpolation perspective discussed earlier has more formal implications for classification. It can be useful to set tuning parameters to better approximate an interpolation of the data. Perhaps most important, a minimum size of 1 for terminal nodes is often very effective, at least for classification. In the same spirit, deep trees and a very large number of iterations should be considered. Examples will be provided later.

A major obstacle to effective tuning is the need for test data. Even with the sampling built into stochastic gradient boosting, there is no provision for retaining the unsampled data for performance evaluation. The out-of-bag data may be used only to help determine the number of iterations.[5] Common tuning advice, therefore, is limited to in-sample performance. But recall that for classification, one can have a perfect fit to the data and still reduce generalization error with more iterations. For classification and regression, using the test data for performance evaluation seems like a good idea.

### *6.3.2 Output*

The key output from stochastic gradient boosting is much the same as the key output from random forests. However, unlike random forests, there are not the usual out-of-bag observations that can be used as test data. Consequently, confusion tables depend on resubstituted data; the data used to grow the trees are also used to evaluate their

[5]For gbm(), the data not selected for each tree are called "out-of-bag" data although that is not fully consistent with the usual definition because in *gbm()*, the sampling is without replacement.

performance. The same applies to fitted values for numerical response variables. Consequently, overfitting can be a complication although the updating over lots of trees helps a lot. Ideally, this problem should be addressed with real test data.

Just as for 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 variable importance measures and partial dependence plots that are similar to those used in random forests.

The partial dependence 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. For *gbm()*, partial dependence plots with binary response variables use either a probability or logit scale (i.e., $p_i/(1-p_i)$) on the vertical axis. Both are vulnerable if measures of classification performance are being used to tune. If tuning is done through measures of fit such as the deviance, one has no more than the usual concerns about overfitting. But, in that case, classification accuracy (should one care) will perhaps be sacrificed.

The exact form taken by the variable importance measures depends on options in the software. One common choice is reductions of the loss function that can be attributed to each predictor. The software sums for each tree how much the loss decreases when any predictor defines a data partition. For example, if for a given tree a particular predictor is chosen 3 times to define data partitions, the three reductions in the loss function are summed as a measure of that predictor's contribution to the fit for that tree. Such sums are averaged over trees to provide the contribution that each predictor makes to the overall fit. The contributions can be reported in raw form or as percentages of the overall reduction in loss. In gbm(), there is on a somewhat experimental basis a random shuffling approach to importance based on predictive accuracy. To date, however, out-of-bag observations are not used so that true forecasting accuracy is not represented. Recall that for random forests, importance is defined by contributions to prediction accuracy in the out-of-bag data.

## 6.4 Asymmetric Costs

All of the available loss functions for categorical outcomes use symmetric costs. False positives count the same as false negatives. For stochastic gradient boosting, there are two ways to easily introduce asymmetric costs. The first is to place a threshold on the fitted values that differs from .5 (or on the logit scale that differs from 0.0). This option was discussed earlier for several other procedures. For example, if a positive is coded 1 and a negative is coded 0, placing the threshold at .25 means false negatives are 3 times most costly than false positives. The problems with this approach were also discussed. All other boosting output is still based on symmetric costs. Moreover, there can be complications if the distribution of the fitted values is either very dense or very sparse in the neighborhood of the threshold. If very dense, small changes in the threshold that make no material difference can alter classification performance dramatically. If very sparse, it can be difficult set the threshold so that the desired cost ratio in a confusion tables is produced.

The second alternative is to use weights. This is much like altering the prior for CART. And like with CART, some trial and error is involved before the classification table with the desired cost ratio is produced. But the intent is to upweight the outcome for which classifications errors are more costly relative to the outcome for which classification error is less costly. An example is provided below.

For numerical response variables, the options are more limited. The only loss function for which asymmetric costs are naturally available is the quantile regression loss function. By choosing the appropriate quantile, underestimates can be given different costs from overestimates. For example, if one estimates the 75th percentile, underestimates are 3 times more costly than overestimates. Looking back at the quantile regression residual expression shown earlier when the algorithm for stochastic gradient boosting was introduced, the value of $\alpha$ is set to the target percentile and is the weight given to all positive residuals. The value of $(1 - \alpha)$ is the weight given to all negative residuals. Positive residuals are underestimates, and negative residuals are overestimates.

Figure 6.2 shows the shape of the loss function when the quantile is greater than .50. As illustrated by the red line, the loss grows more rapidly for underestimates (i.e., positive residuals) than overestimates (i.e., negative residuals). For quantiles less than .50, the reverse is true. For a quantile of .50, the red and blue line have the same rate of growth.

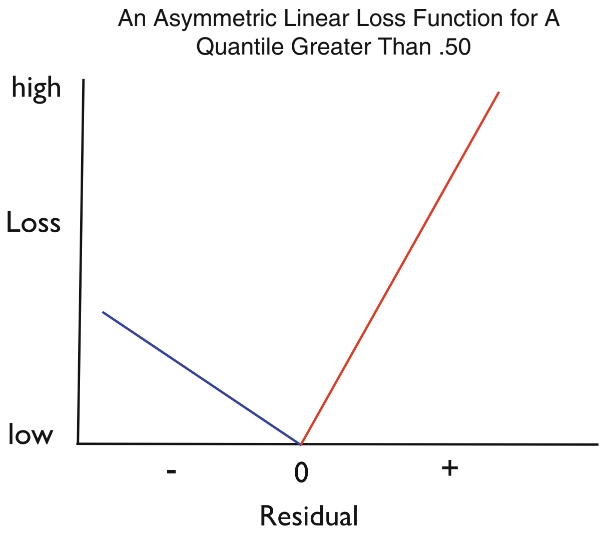

**Fig. 6.2** Asymmetric loss function for the quantile loss function with the quantile set at .75

## 6.5 Boosting, Estimation, and Consistency

The most important output from boosting is the fitted values. For a level I analysis, these are just statistics computed for the data on hand. But, often there is an interest in using the values as estimates of the fitted values in the joint probability distribution responsible for the data. This is a level II analysis. Just as for random forests, no claims are made that boosting will provide accurate estimates of the true response surface. At best, one can get a consistent estimate of generalization error for a given sample size,[6] boosting specification and set of tuning parameters values (Jiang, 2004, Zhang and Yu 2005, Bartlett and Traskin 2007). But existing proofs either impose artificial conditions or are limited to a few of the "easier" loss functions such as exponential loss and quadratic loss. And even then, the implications for practice are not clear. There can be a Goldilocks stopping strategy for a given number of observations at which the number of iterations is neither too few nor too many. But how to find that sweet spot for a given analysis is not explained.

The best one can do in practice is apply some empirical heuristic and hope for the best. As already noted, that heuristic can be the point at which the decrease in the loss function levels off. Some measure of fit between the observed response values and the fitted values can then be used as a rough proxy for generalization error for that sample size, stopping decision, specification, and associated tuning parameter settings. Conventionally, this is an in-sample estimate. A more honest estimate of generalization error can be obtained from a test sample and as described earlier, one can use the nonparametric bootstrap to approximate the variance in that estimate.

## 6.6 A Binomial Example

We return to the Titanic data for some applications of stochastic boosting as implemented in *gbm()*. Recall that the response is whether or not a passenger survived. The predictors we use are gender ("sex"), age, class of cabin ("pclass"), number of siblings/spouses aboard("sibsp"), and the number of parents/children aboard ("parch"). The code used for the analysis with Bernoulli loss is provided in Fig. 6.3. At the top, the data are loaded, and a new data set is constructed. A weighting variable is constructed for later use. All NA entries removed. Removing NAs in advance is required when a procedure does not discard them automatically.

Consistent with our earlier discussion, the minimum terminal node size is set to 1, and the interaction depth set to 3. Setting interaction depth to a larger value (e.g., 8) led to fewer iterations but essentially the same results. Setting it to a smaller value (e.g., 1) led to more iterations, but also essentially same results. The number of iterations was set to 4000 anticipating that 4000 should be plenty. If not, the number could be

[6]Even with the minimum number of observations allowed in terminal nodes specified, with more observations there can be larger trees. There can be more splits before the minimum is reached.

```
# Load and Clean Up Data
library(PASWR)
data("titanic3")
attach(titanic3)
wts<-ifelse(survived==1,1,3) # for asymmetric costs
Titanic3<-na.omit(data.frame(fare,survived,pclass,
                             sex,age,sibsp,parch,wts))

# Boosted Binomial Regression
library(gbm)
out2<-gbm(survived~pclass+sex+age+sibsp+parch,
         data=Titanic3,n.trees=4000,interaction.depth=3,
         n.minobsinnode = 1,shrinkage=.001,bag.fraction=0.5,
         n.cores=1,distribution = "bernoulli")

# Output
gbm.perf(out2,oobag.curve=T,method="OOB",overlay=F) # 3245
summary(out2,n.trees=3245,method=permutation.test.gbm,
        normalize=T)
plot(out2,"sex",3245,type="response")
plot(out2,"pclass",3245,type="response")
plot(out2,"age",3245,type="response")
plot(out2,"sibsp",3245,type="response")
plot(out2,"parch",3245,type="response")
plot(out2,c("subsp","parch"),3245,type="response") # Interaction

# Fitted Values
preds2<-predict(out2,newdata=Titanic3,n.trees=3245,
                type="response")
table(Titanic3$survived,preds2>.5)
```

**Fig. 6.3** R code for Bernoulli regression boosting

increased. All else were the defaults, except that the number of cores available was one. Even with only one core, the fitting took about a second in real time.[7]

Figure 6.4 shows standard *gbm()* performance output. On the horizontal axis is the number of iterations. On the vertical axis is the change in the Bernoulli deviance based on the OOB observations. The OOB observations provide a more honest assessment than could be obtained in-sample. However, they introduce sampling error so that the changes in the loss bounce around a bit. The reductions in the deviance decline as the number of iterations grows and become effectively 0.0 shortly after the 3000th pass through the data. Any of the iterations between 3000 and 4000 lead to about

[7] For these analyses, the work was done on an iMac with a single core. The processor was a 3.4 Ghz Intel Core i7.

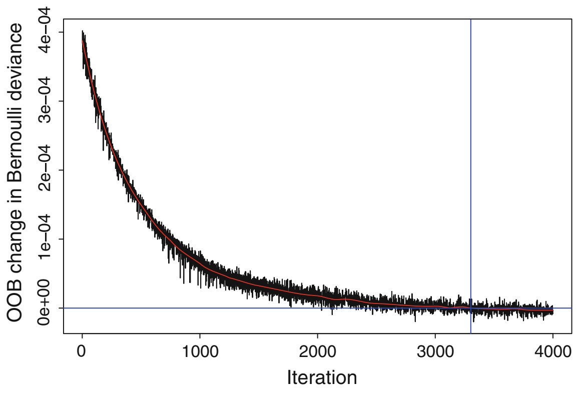

**Fig. 6.4** Changes in Bernoulli deviance in OOB data with iteration 3245 as the stopping point (N = 1045)

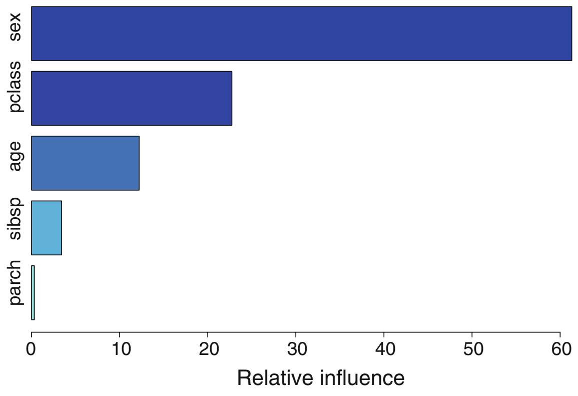

**Fig. 6.5** Titanic data variable importance plot for survival using binomial regression boosting (N = 1045)

the same fit of the data, but the software selects iteration 3245 as the stopping point. Consequently, the first 3245 trees are used in all subsequent calculations.[8]

Figure 6.5 is a variable importance plot shown in the standard *gbm()* format for the predictor shuffling approach. Recall that unlike in random forests, the reductions are for predictions into the full dataset, not the subset of OOB observations. Also the contribution of each input is standardized differently. All contributions are given as percentages of the summed contributions. For example, gender is the most important predictor with a relative performance of 60 (i.e., 60 %). The class of passage is the next most important input with a score of about 25, followed by age with a score of about 12. If you believe the accounts of the Titanic's sinking, these contributions make sense. But just as with random forests, each contribution includes any interaction effects with other variables unless the tree depth is equal to 1 (i.e., *interaction.depth* = 1). So, the contributions in Fig. 6.5 cannot be attributed to each input by itself. Equally important, contributions to the fit are not regression coefficients and or

[8] If forecasting were on the table, it might have been useful to try a much larger number of iterations to reduce generalization error.

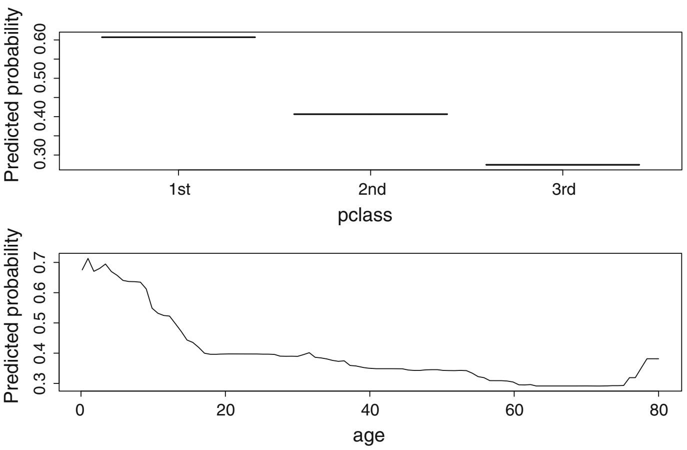

**Fig. 6.6** Titanic data partial dependence plots showing survival proportions for class of passage and age using binomial regression boosting (N = 1045)

contributions to forecasting accuracy. It may not be clear, therefore, how to use them when real decisions have to be made.

Figure 6.6 presents two partial dependence plots with the fitted probability/proportion on the vertical axis. One has the option of reporting the results as probabilities/proportions or logits. One can see that class of passage really matters. The probability of survival drops from a little over 60 to a little under .30 from first class to second class to third class. Survival is also strongly related to age. The probability of survival drops from about .70 to about .40 as age increases from about 1 year to about 18. There is another substantial drop around age 55 and an increase around age 75. But there are very few passengers older than 65, so the apparent increase could be the result of instability.[9]

Figure 6.7 is a partial plot designed to show two-way interaction effects. The two inputs are the number of siblings/spouses aboard and the number of parents/children aboard, which are displayed as a generalization of a mosaic plot. The inputs are shown on the vertical and horizontal axes. The color scale is shown on the far right. A combination of sibsp >5 and parch >4 has the smallest chances survival; about a quarter survived. A combination of sibsp <2 and parch <3 has the largest chance of survival; a little less than half survived.[10] In this instance, there does not seem to be important interaction effects. The differences in the colors from top to bottom are about the same regardless of the value for sibsp. For example, when sibsp is 6, the proportion surviving changes top to bottom from about .25 to about .30. The

[9] The plots are shown just as *gbm()* builds them, and there are very few options provided. But just as with random forests, the underling data can be stored and then used to construct new plots more responsive to the preferences of data analysts.

[10] Because both inputs are integers, the transition from one value to the next is the midpoint between the two.

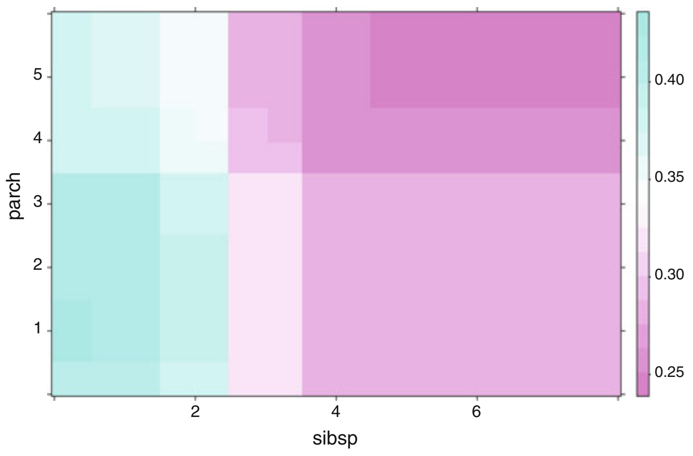

**Fig. 6.7** Interaction partial dependence plot: survival proportions for the number of siblings/spouses aboard and the number of parents/children aboard using binomial regression boosting (N = 1045)

**Table 6.1** Confusion table for Titanic survivors with default 1 to 1 weights (N = 1045)

| | Forecast perished | Forecast survived | Model error |
|---|---|---|---|
| Perished | 561 | 57 | .09 |
| Survived | 126 | 301 | .29 |
| Use error | .18 | .16 | Overall error = .18 |

difference is −.05. When sibsp is 1, the proportion surviving changes from top to bottom from about .35 to about .40. The difference is again around −.05. Hence, the association between sibsp and survival is approximately the same for both values of sibsp.

It is difficult to read the color scale for Fig. 6.7 at the necessary level of precision. One might reach different conclusions if numerical values are examined. But the principle just illustrated is valid for how interaction effects are represented. And it is still true for these two predictors that a combination of many siblings/spouses and many parents/children is the worst combination of these two predictors whether or not their effects are only additive.

Table 6.1 is the confusion table that results when each case is given the same weight. In effect, this is the default. The empirical cost ratio that results is about 2.2 to 1 with misclassification errors for those who perished about twice as costly as misclassification errors for those who survived. Whether that is acceptable depends on how the results would be used. In this instance, there are probably no decisions to be made based on the classes assigned, so the cost ratio is probably of not much interest.

Stochastic gradient boosting does a good job distinguishing those who perished from those who survived. Only 9 % of those who perished were misclassified, and

**Table 6.2** Confusion table for Titanic survivors with 3–1 weights (N = 1045)

| | Forecast perished | Forecast survived | Model error |
|---|---|---|---|
| Perished | 601 | 17 | .03 |
| Survived | 195 | 232 | .46 |
| Use error | .24 | .08 | Overall error = .21 |

only 29 % of those who survived were misclassified. The forecasting errors of 18 % and 16 % are also quite good although it is hard to imagine how these results would be used for forecasting.

Table 6.2 repeats the prior analysis but with survivor observations weighted as 3 times more than the observations for those who perished. Because there are no decisions to be made based on the analysis, there is no grounded way to set the weights. The point is just to illustrate that weighting can make a big difference in the results that, in turn, affect the empirical cost ratio in a confusion table. That cost ratio is now 11.5 so that misclassifications of those who perished are now over 11 times more costly than misclassifications of those who survived. Consequently, the proportion misclassified for those who perished drops to 3 %, and the proportion misclassified for those who survived increases to 46 %. Whether these are more useful results than the results shown in Table 6.1 depend on how the results would be used.[11]

Should one report the results in proportions or probabilities? For these data, proportions seem more appropriate. As already noted, the Titanic sinking is probably best viewed as a one-time event that has already happened, which implies there may be no good answer to the question "probability of what?" Passengers either perished or survived, and treating such an historically specific event as one of many identical, independent trials seems a stretch. This is best seen as a level I analysis.

## 6.7 A Quantile Regression Example

For the Titanic data, the fare paid in dollars becomes the response variable, and the other predictors just as before. Because there are a few very large fares, there might be concerns about how well boosted normal regression would perform. Recall that boosted quantile regression is robust with respect to response variable outliers or a highly skewed distribution and also provides a way to build in relative costs for fitting errors. Figure 6.8 shows the code for a boosted quantile regression fitting the conditional 75th percentile.

There are two significant changes in the tuning parameters. First, the distribution is now "quantile" with alpha as the conditional quantile to be estimated. We

[11]It is not appropriate to compare the overall error rate in the two tables (.18–.21) because the errors are not weighted by costs. In Table 6.2, classification errors for those who perished are about 5 times more costly.

```
# Load Data and Clean Up Data
library(PASWR)
data("titanic3")
attach(titanic3)
Titanic3<-na.omit(data.frame(fare,pclass,
                             sex,age,sibsp,parch))

# Boosted Quantile Regression
library(gbm)
out1<-gbm(fare~pclass+sex+age+sibsp+parch,data=Titanic3,
         n.trees=12000,interaction.depth=3,
         n.minobsinnode = 10,shrinkage=.001,bag.fraction=0.5,
         n.cores=1, distribution = list(name="quantile",
         alpha=0.75))

#Output
gbm.perf(out1,oobag.curve=T) # 4387
summary(out1,n.trees=4387,method=relative.influence)
par(mfrow=c(2,1))
plot(out1,"sex",4387,type="link")
plot(out1,"age",4387,type="link")
plot(out1,"sibsp",4387,type="link")
plot(out1,"parch",4387,type="link")
plot(out1,c("pclass","age"),4448,type="link") # Interaction

# Fitted Values
preds1<-predict(out1,newdata=Titanic3,n.trees=4387,type="link")
plot(preds1,Titanic3$fare,col="blue",pch=19,
      xlab="Predicted Fare", ylab="Actual Fare",
      main="Results from Boosted Quantile Regression
      with 1 to 1 line Overlaid: (alpha=.75)")
      abline(0,1,col="red",lwd=2)
```

**Fig. 6.8** R code for quantile regression boosting

begin by estimating the conditional 75th percentile. Underestimates are taken to be 3 times more costly than overestimates. Second, a much larger number of iterations is specified than for boosted binomial regression. For the conditional 75th percentile, only a little over 4000 iterations are needed. But we will see shortly that for other conditional percentiles, at least 12,000 iterations are needed. There is a very small computational penalty for 12,000 iterations for these data (Fig. 6.9).

Figure 6.10 is the same kind of importance plot as earlier except that importance is now represented by the average improvement over trees in fit for the quantile loss

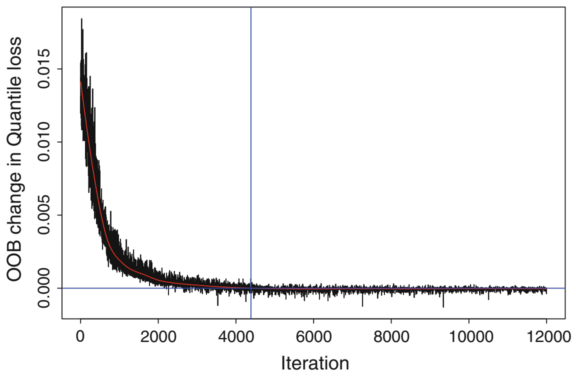

**Fig. 6.9** Changes in the quantile loss function for the fare paid with OOB Titanic data and with iteration 4387 as the stopping point (N = 1045)

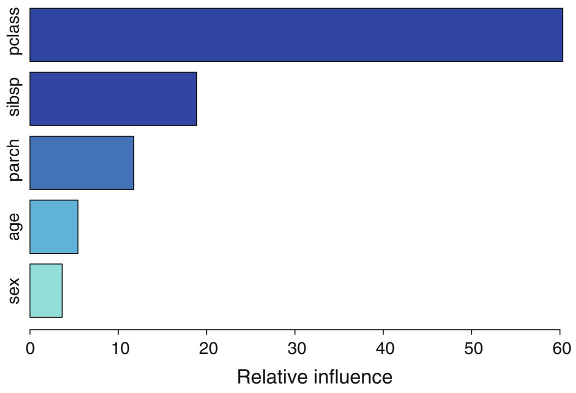

**Fig. 6.10** Variable importance plot for the fare paid using quantile regression boosting with the 75th percentile (N = 1045)

function as each tree is grown. This is an in-sample measure.[12] Nevertheless, the plot is interpreted essentially in the same fashion. Fare is substantially associated with the class of passage, just as one would expect. The number of siblings/spouses is the second most important predictor, which also makes sense. With so few predictors, and such clear differences in their contributions, the OOB approach and the in-sample approach will lead to about the same relative contributions.

Figure 6.11 shows for illustrative purposes two partial response plots. The upper plot reveals that the fitted 75th percentile is about \$46 for females and a little less than \$36 for males with the other predictors held constant. It is difficult to know what this means, because class of passage is being held constant and performs just as one would expect (graph not shown). One possible explanation is that there is variation in amenities within class of passage, and females are prepared to pay more for them. The lower plot shows that variation in fare with respect to age is at most around \$3 and is probably mostly noise, given all else that is being held constant.

Figure 6.12 is another example of an interaction partial plot. The format now shows a categorical predictor (i.e., class of passage) and a numerical predictor

[12]The out-of-bag approach was not available in *gbm()* for boosted quantile regression.

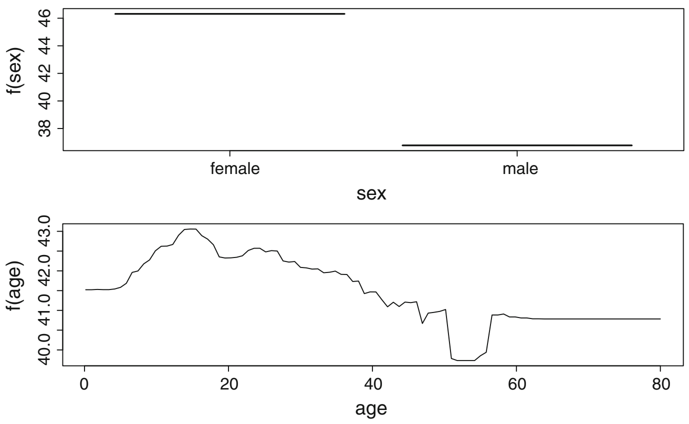

**Fig. 6.11** Partial dependence plot for the Titanic data showing the fare paid for class of passage and age using quantile regression boosting fitting the 75th percentile (N = 1045)

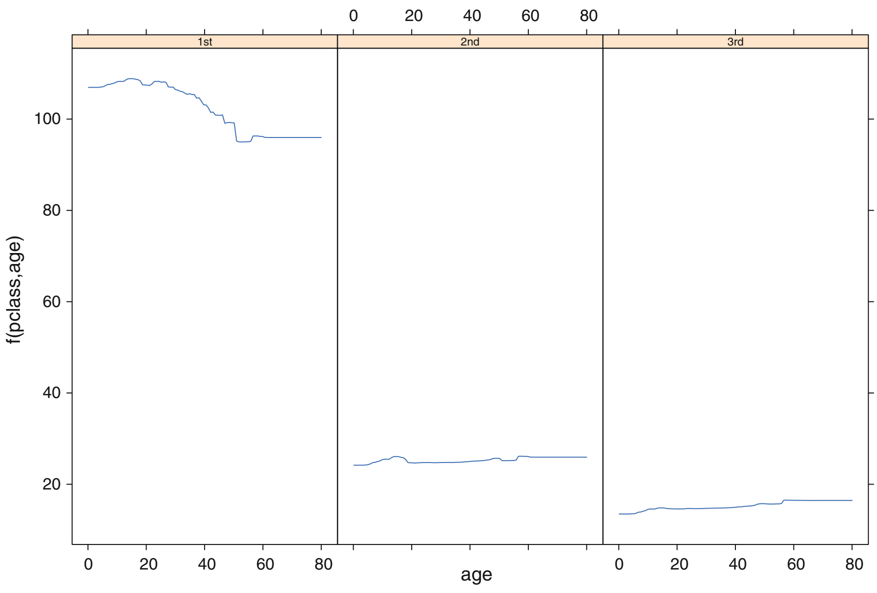

**Fig. 6.12** Titanic data interaction partial dependence plot showing the fare paid for the number of siblings/spouses aboard and the number of parents/children aboard using quantile regression boosting fitting the 75th percentile (N = 1045)

(i.e., age). There are apparently interaction effects. Fare declines with age for a first class passage but not for a second or third class passage. Perhaps older first class passengers are better able to pay for additional amenities. Perhaps, there is only one fare available for second and third class passage.

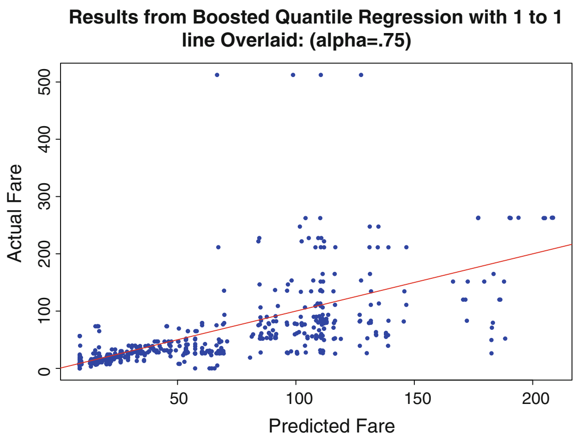

**Fig. 6.13** Actual fare against fitted fare for a boosted quantile regression analysis of the Titanic data with a 1-to-1 line overlaid (alpha = .75, N = 1045)

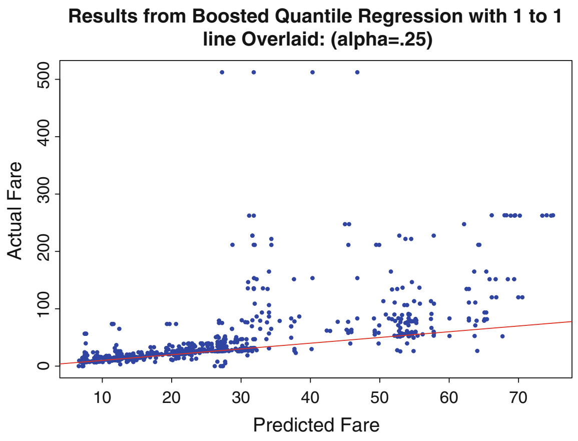

**Fig. 6.14** Actual fare against fitted fare for a boosted quantile regression analysis of the Titanic data with a 1-to-1 line overlaid (alpha = .25, N = 1045)

Figure 6.13 is a plot of the actual fare against the fitted fare for the 75th percentile. Underestimates are 3 times more costly than overestimates. Overlaid is a 1-to-1 line that provides a point of reference. Most of the fitted values fall below the 1-to-1 line, as they should. Still, four very large fares are grossly underestimated. They are few and even with the expanded basis functions used in stochastic gradient boosting, could not be fit well. The fitted values range from near \$0 to over \$200, and roughly speaking, the fitted 75th percentile increases linearly with the actual fares. The correlation between the two is over .70.

Figure 6.14 is a plot of the actual fare against the fitted fare for the 25th percentile. Overestimates now are taken to be 3 times more costly than underestimates. Overlaid again is a 1-to-1 line that provides a point of reference. Most of the actual fares fall above the 1-to-1 line. This too is just as it should be. The fitted values range from a little over $0 to about $75. Overall the fit still looks to be roughly linear, and the correlation is little changed.[13]

Without knowing how the results from a boosted quantile regression are to be used, it is difficult to decide which quantiles should be fitted. If robustness is the major concern, using the 50th percentile is a sensible default. But there are many applications where for subject-matter or policy reasons, other percentiles can be desirable. As discussed earlier, for example, if one were estimating the number of homeless in a census tract (Berk et al. 2008), stakeholders might be very unhappy with underestimates because social services would not be made available where they were most needed. Fitting the 90th percentile could be a better choice. Or, stakeholders might on policy grounds be interested in the 10th percentile if in a classroom setting, there are special concerns about students who are performing poorly. It is the performance of kids who struggle that needs to be anticipated.

## 6.8 Summary and Conclusions

Boosting is a very rich 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 many kinds of data analysis.

However, there are important limitations to keep in mind. First, boosting is designed to improve the performance of weak learners. Trying to boost learners that are already strong is not likely to be productive. Whether a set of learners is weak or strong is a judgement call that will vary over academic disciplines and policy areas. If the list of variables includes all the predictors known to be important, if these predictors are well measured, and if the functional forms with the response variables are largely understood, conventional regression will then perform well and provide output that is much easier to interpret.

Second, if the goal is to fit conditional probabilities, boosting can be a risky way to go. There is an inherent tension between reasonable estimates of conditional probabilities and classification accuracy. Classification with the greatest margins is likely to be coupled with estimated conditional probabilities that are pushed toward the bounds of 0 or 1.

Third, boosting is not alchemy. Boosting can improve the performance of many weak learners, 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 seriously flawed measurement and badly executed data collection. The same applies to all of the statistical learning procedures discussed in this book.

[13]The size of the correlation is being substantially determined by actual fares over $200. They are still being fit badly, but not a great deal worse.

Finally, when compared to other statistical learning procedures, especially random forests, boosting will include a much 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.

## Exercises

### *Problem Set 1*

Generate the following data. The systematic component of the response variable is quadratic.

```
x1=rnorm(1000)
x12=x1^2 ysys=1+(-5*x12)
y=ysys+(5*rnorm(1000))
dta=data.frame(y,x1,x12)
```

1. Plot the systematic part of $y$ against the predictor $x1$. Smooth it using *scatter.smooth()*.The smooth can be a useful approximation of 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 bare-bones code to get you started. But feel free to experiment. For example,

   ```
   out<-gbm(y~x1,distribution="gaussian",n.trees=10000,
         data=dta)
   gbm.perf(out,method="OOB")
   ```

   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, 10000 and the number recommended by the out-of-bag 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 and beyond? Why does this happen?

3. Repeat the analysis with the interaction.depth $= 3$ (or larger). What in the performance of the procedure has changed? What has not changed (or at least not changed much)? Explain what you think is going on. (Along with n.trees, interaction.depth can make an important difference in performance. Otherwise, the defaults usually seem adequate.)

***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(). What is the recommended number of iterations?
2. Construct a graph of the importance of the predictors. Which variables seem to affect the fit substantially and which do not? Make sure your interpretations take the units of importance into account.
3. Construct the partial dependence plot for each predictor. Interpret each plot.
4. Construct all of the two-variable plots. Interpret each plot. Look for interaction effects. (There are examples in the gbm documentation that can be accessed with *help()*.)
5. Construct the three-variable plot. (There are examples in the *gbm()* documentation that can be accessed with *help()*.) Interpret the plot.
6. Consider the quality of the fit. How large is the improvement compared to when no predictors are used? You will need to compute measures of fit. There are none in *gbm.object*.
7. Write a paragraph or so on what the analysis of these data has revealed about correlates of infant mortality at a national level.
8. 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?
9. Repeat the analysis using the quantile loss function. Try values for $\alpha$ of .25, .50, and .75, which represent different relative costs for underestimates compared to overestimates. How do the results differ in the number of iterations, variable importance, partial dependence plots, and fit? How do the results compare to your early analysis using stochastic gradient boosting?

***Problem Set 3***

The point of this problem set is to compare the performance of several different procedures when the outcome is binary and decide which work better and which work worse for the data being analyzed. You also need to think about why the performance can differ and what general lessons there may be.

From the *MASS* library, analyze the dataset called Pima.tr. The outcome is binary: diabetes or not (coded as "Yes" and "No" for the variable "type."). Assume that the costs of failing to identify someone who has diabetes are 3 times higher than the costs of falsely identifying someone who has diabetes. The predictors are all of the other variables in the dataset.

The statistical procedures to compare are logistic regression, the generalized additive model, random forests, and stochastic gradient boosting. For each, you will need to determine how to introduce asymmetric costs. (Hint: for some you will need to weight the data by outcome class.) You will also need to take into account the data format each procedure is expecting (e.g., can missing data be tolerated?). Also feel free to try several different versions of each procedure (e.g., "Adaboost" v. "bernoulli" for stochastic gradient boosting). The intent is to work across material from several earlier chapters.

1. Construct confusion tables for each model. Be alert to whether the fitted values are for "resubstituted" data or not. 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. The units and computational methods differ. 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?