Boosting

Like bagging, **boosting** (or **gradient boosting**) generates predictions from many models and forms a sort of average.

The idea is to produce a strong model by combining many weak ones.

- Unlike with bagging, the weak models are not generated independently, and there is no resampling involved.
- Instead, we sequentially modify the data and obtain a new weak model for each modified dataset.

Boosting was originally developed for classification, but it can equally well be used with regression.

The `gbm()` function in the `gbm` package can handle a wide variety of models for regression and classification. The name stands for "generalized boosted regression models."

For regression trees, the algorithm works as follows:

1. Set $\hat{f}(x_i) = 0$ and $r_i = y_i$ for all $i = 1, \ldots, n$.
2. For $b = 1, \ldots, B$, repeat:
   1. Fit a tree $\hat{f}^b(x)$ with d splits to the training data (x, r).
   2. Update $\hat{f}$ by adding a shrunken version of $\hat{f}^b(x)$ to it:

$$\hat{f}(x_i) \leftarrow \hat{f}(x_i) + \lambda \hat{f}^b(x_i), \quad i = 1, \ldots, n. \tag{1}$$

   3. Update the residuals:

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{2}$$

3. The boosted model is

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{3}$$

Recall that the initial value of $\hat{f}$ was $\mathbf{0}$. Thus all of the explanatory power comes from the $\hat{f}^b(x)$.

This procedure involves estimating a lot of trees when B is large, but they can be quite shallow trees, sometimes just stumps (i.e. trees with just a single split).

There are three tuning parameters:

1. B, the number of (in this case) trees. There is a risk of overfitting if B gets too large, so we need to use cross-validation.
2. The **shrinkage parameter** λ. Typical values are between 0.01 and 0.0001. Smaller seems to be better, but it is costlier.
   - When λ is very small, B will probably need to be very large.
3. The number of splits d, called the **interaction depth**. This tends to be small, perhaps just $d = 1$, but the default is $d = 6$.
   - When $d = 1$, every split is a stump, and the model is additive (but not linear).

In general, λ should not equal $1/B$, even though they should be inversely related.

Modern boosting methods often use **subsampling** each time they grow a new tree.

Subsampling is like the resampling bootstrap, but instead of creating a random sample of size n with replacement, it creates a random sample of size $pn < n$ without replacement.

The author of `gbm` suggests setting p, or `bag.fraction`, to 0.5.

- Subsampling makes computation faster, because there are fewer observations.
- More importantly, it reduces the correlation among the trees and yields better predictions.
- It also allows the model to compute out-of-bag predictions.

Detailed information about `gbm` may be found at:
`https://cran.r-project.org/web/packages/gbm/vignettes/gbm.pdf`

The discussion of boosting in ISLR/ISLP is quite brief, but ESL devotes most of a chapter to it.

Boosting combines additivity, as is evident in (3), and nonlinearity. Because the nonlinear functions are just trees, often shallow trees, they are very easy to fit.

It would never make sense to boost a linear regression model, because the residuals are orthogonal to all the predictors.

For squared error loss, the objective function is

$$\sum_{i=1}^{n} L\big(y_i, f(\boldsymbol{x}_i)\big) = \sum_{i=1}^{n} \big(y_i - f(\boldsymbol{x}_i)\big)^2, \tag{4}$$

where $L\big(y_i, f(\boldsymbol{x}_i)\big)$ denotes the loss associated with the i^{th} observation.

Note that (4) is not the only possible loss function. `gbm` can be used to estimate quite a few models in addition to regression models.

Since we build up the $f(x_i)$ slowly, we see that

$$\begin{aligned} L\big(y_i, \hat{f}(x_i)\big) &\leftarrow L\big(y_i, \hat{f}(x_i) + \lambda \hat{f}^b(x_i)\big) \\ &= \big(y_i - \hat{f}(x_i) - \lambda \hat{f}^b(x_i)\big)^2 \\ &= \big(r_i - \lambda \hat{f}^b(x_i)\big)^2, \end{aligned} \tag{5}$$

where r_i is simply the i^{th} residual for the current model, before we have added the b^{th} term to it.

At each step in the boosting algorithm, we add λ times the term $\hat{f}^b(x_i)$ that best fits the function

$$\sum_{i=1}^{n} \big(r_i - \hat{f}^b(x_i)\big)^2, \tag{6}$$

which just depends on the current residuals and on $\hat{f}^b(x_i)$.

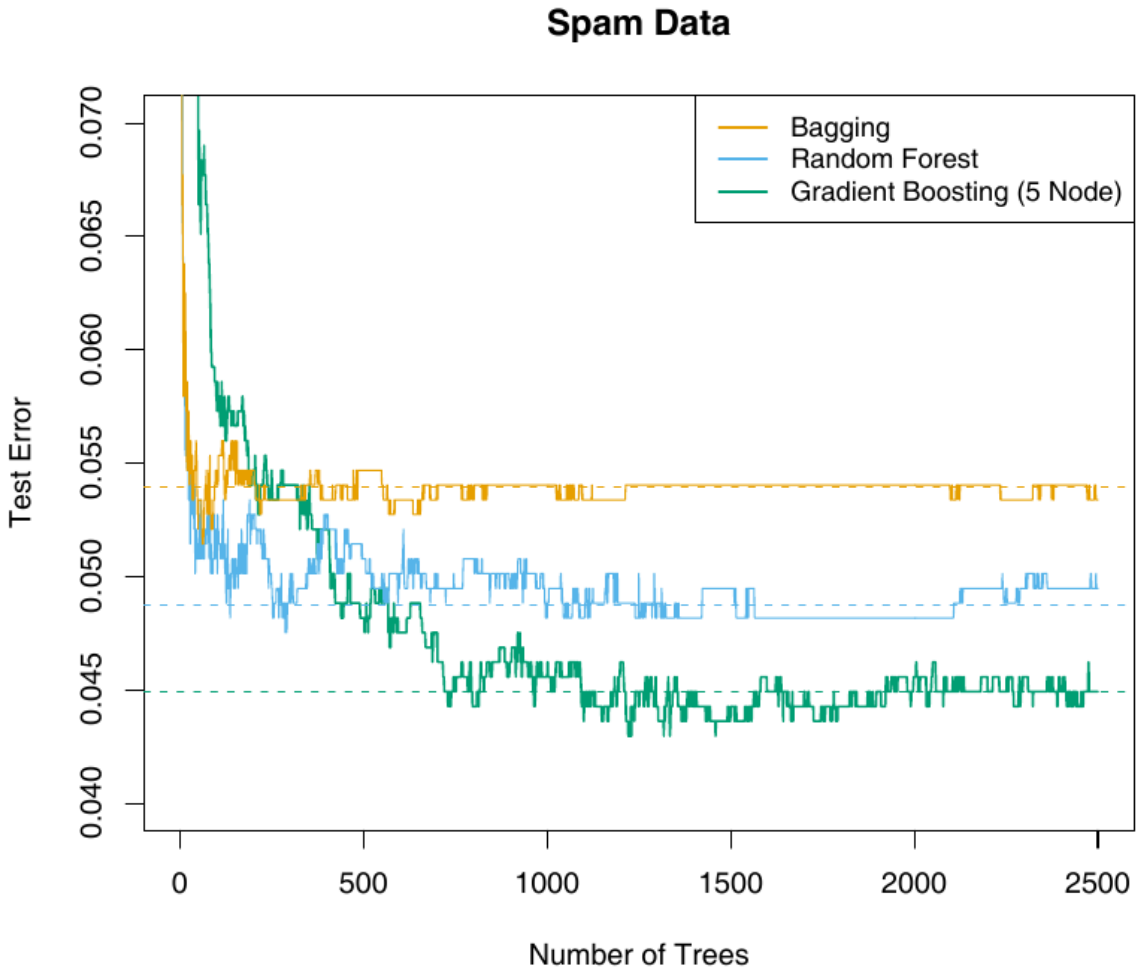


FIGURE 15.1. *Bagging, random forest, and gradient boosting, applied to the spam data. For boosting, 5-node trees were used, and the number of trees were chosen by 10-fold cross-validation (2500 trees). Each "step" in the figure corresponds to a change in a single misclassification (in a test set of 1536).*

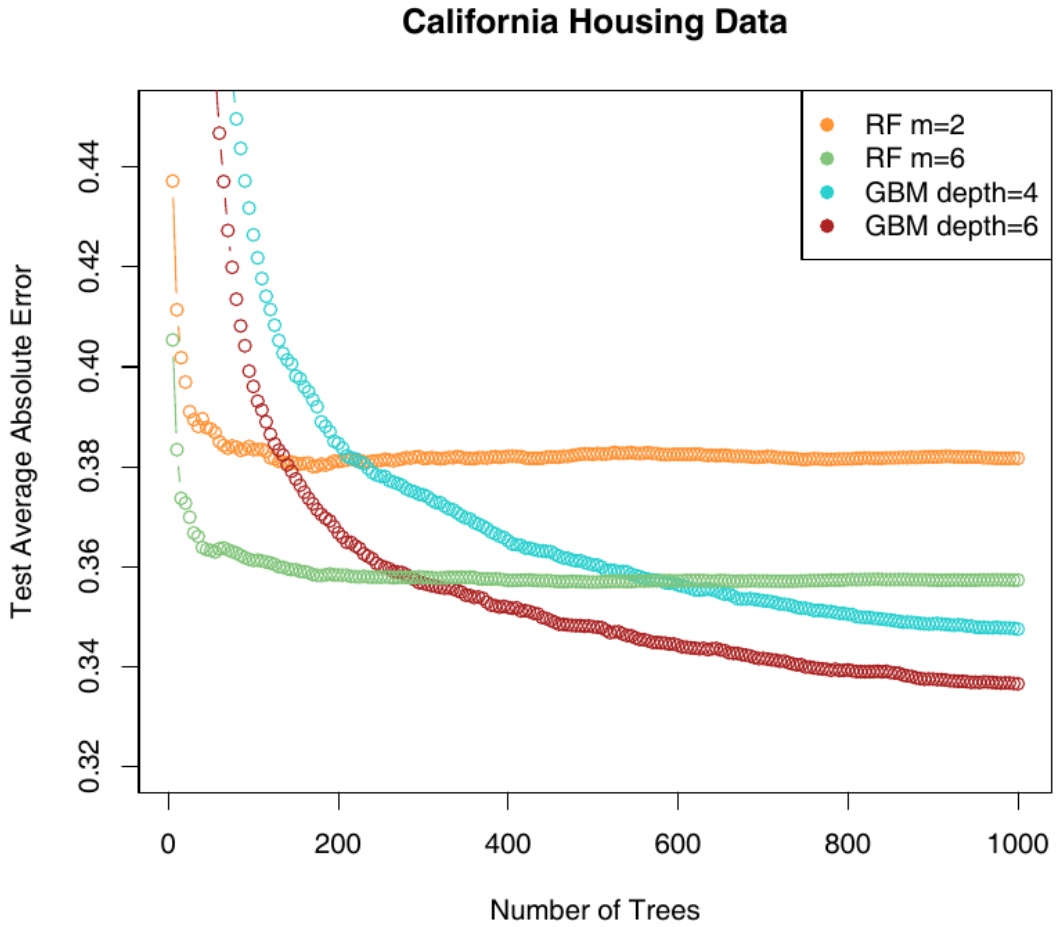


FIGURE 15.3. *Random forests compared to gradient boosting on the California housing data. The curves represent mean absolute error on the test data as a function of the number of trees in the models. Two random forests are shown, with* $m = 2$ *and* $m = 6$*. The two gradient boosted models use a shrinkage parameter* $\nu = 0.05$ *in (10.41), and have interaction depths of* 4 *and* 6*. The boosted models outperform random forests.*

Figures 15.1 and 15.3 from ESL both show that boosting can outperform random forests.

The value of m for the random forests in Figure 15.1 is not specified. It is evidently less than p.

Figure 15.3 illustrates several important features of both random forests and boosting, at least for this example.

- The value of m (number of splits) can matter a lot for random forests.
- The depth of the trees d can matter a lot for boosting.
- The performance of random forests initially improves very rapidly as B increases.
- The performance of boosting initially improves more slowly than that of random forests, but the improvement continues even for very large values of B.
- It looks to me as if the authors should have gone beyond 1000 trees for boosting.

- Because the residuals at each step depend on the other trees that have already been grown, smaller trees are typically sufficient.
- If every tree is a stump, the boosted model is just a sum of stairstep approximations.
- When B is large, there may be quite a few steps for inputs that have a lot of explanatory power.
- In such a case, we might want to investigate other additive models that are smoother functions of the inputs.

Boosting can be used with all sorts of nonlinear basis functions, not just trees.

The `gbm` function can perform K-fold cross-validation; 5-fold is recommended for choosing B, but ESL used 10-fold in Figure 15.1.

The `gbm.perf` function uses out-of-bag (OOB) estimation to estimate performance, so there is no need for a test dataset. This is possible whenever `bag.fraction` is not one.

Boosting for Two-Way Classification

We code the output as $\{-1, 1\}$ instead of $\{0, 1\}$.

This tells us that boosting was invented by a computer scientist (Robert Schapire, in 1990) not a statistician. He works for Microsoft Research and has more than 158,322 citations. :–)

The error rate on the training sample is

$$\overline{\text{err}} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\big(y_i \neq G(\boldsymbol{x}_i)\big), \tag{7}$$

where $G(\boldsymbol{x}_i)$ denotes the classifier, which also takes values $\{-1, 1\}$.

Boosting sequentially applies the weak classifier to repeatedly modified versions of the data.

Eventually, we have B weak classifiers, denoted $G_b(\boldsymbol{x}_0)$ for $b = 1, \ldots, B$, for any point $\boldsymbol{x}_0$.

We then combine them to produce the final prediction:

$$G(\boldsymbol{x}) = \text{sign}\left(\sum_{b=1}^{B} \alpha_b G_b(\boldsymbol{x})\right), \tag{8}$$

where the weights α_b have to be determined.

Instead of using residuals from successive steps, classification uses weights that change as the algorithm proceeds.

Observations that were misclassified get more weight, and observations that were correctly classified get less weight.

ESL describes the `AdaBoost.M1` algorithm of Freund and Schapire (1997; 29,955 Google cites).

Friedman, Hastie, and Tibshirani (2000) proposed "real AdaBoost". It yields real numbers in the $[0, 1]$ interval instead of integers $\{-1, 1\}$.

`gbm` is based on work of Friedman that extended FHT (2000).

Figure 10.2 in ESL shows a simulated example with 2000 training cases and 10,000 test ones.

The weak classifier is a tree with two terminal nodes, a "stump."

The error rate of the initial classifier is 45.8%. After boosting with $B = 400$, it is only 5.8%.

In contrast, a single large tree has an error rate of 24.7%. The figure does not show results for a random forest.

I applied `gbm` to the mileage data. See Figure 17.4.

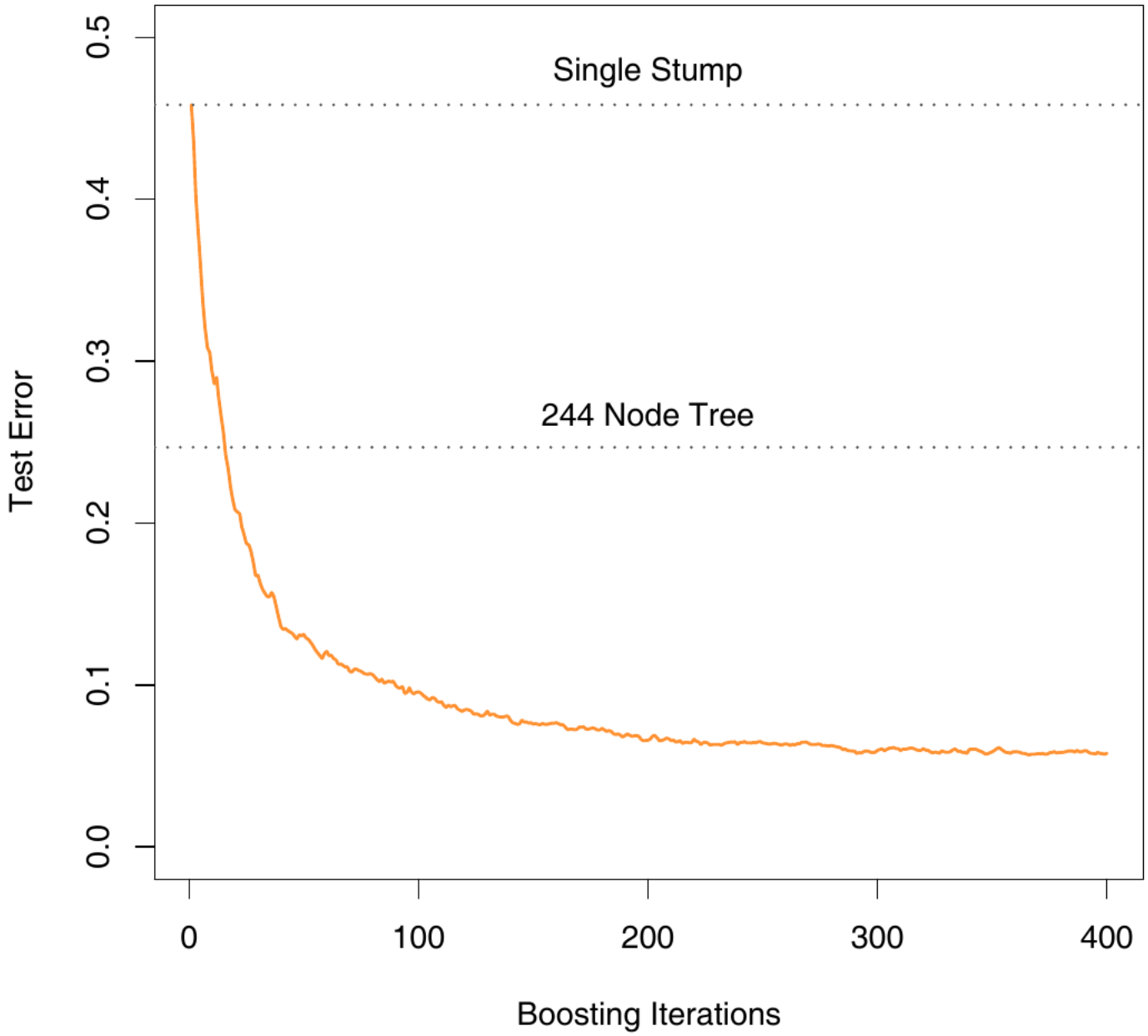
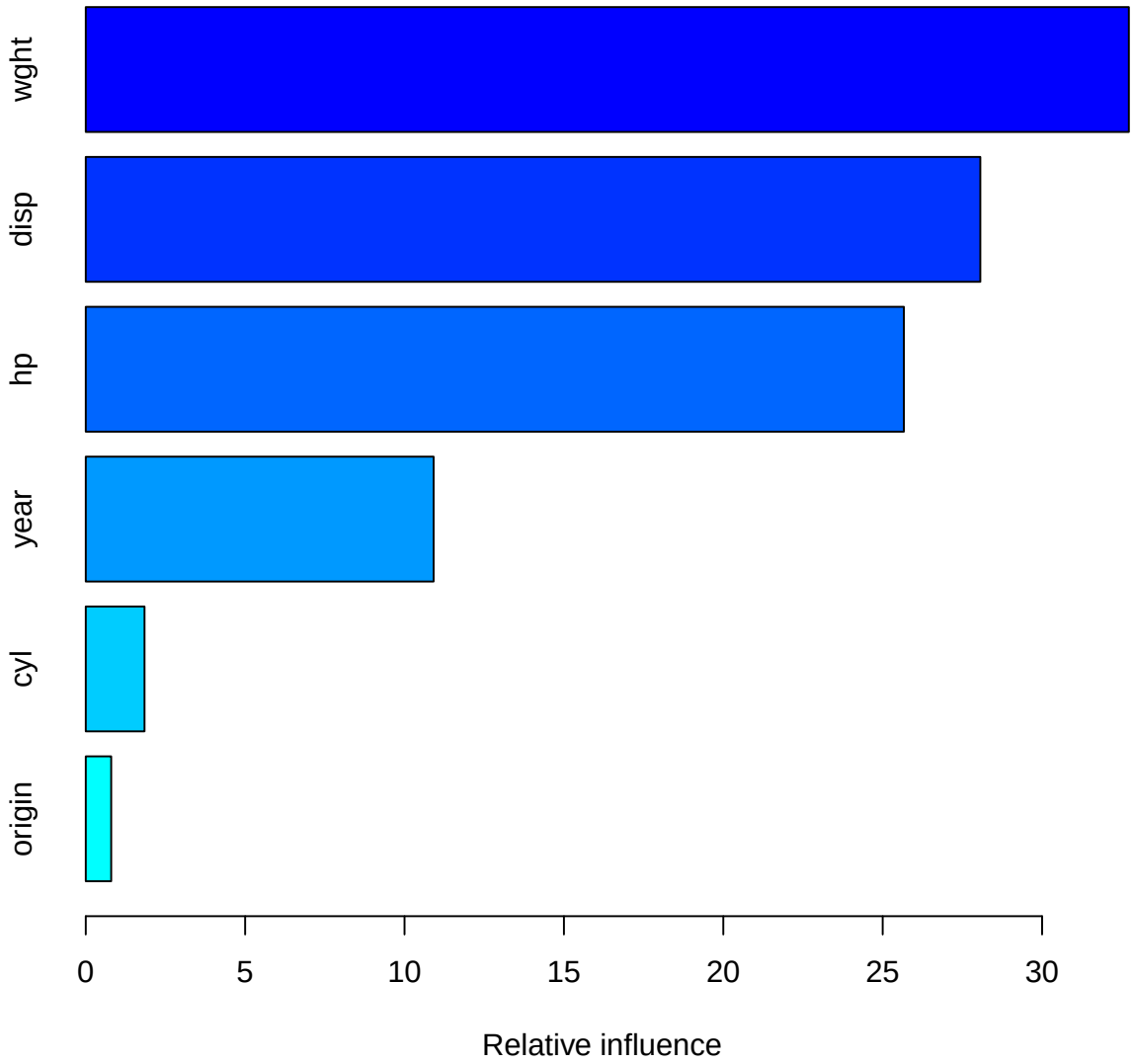

Single Stump
244 Node Tree
Test Error
0.0
0.1
0.2
0.3
0.4
0.5
0
100
200
300
400
Boosting Iterations

Figure 17.4 — Influence of Six Predictors



Boosted trees seem to work very well.

Using an interaction depth (d) of 6, fit improves monotonically as the number of trees (`n.trees`) increases.

$$\begin{aligned} 1000 &\longrightarrow 0.001346 \\ 2000 &\longrightarrow 0.000759 \\ 5000 &\longrightarrow 0.000371 \\ 10,000 &\longrightarrow 0.000315 \\ 20,000 &\longrightarrow 0.000298 \end{aligned} \tag{9}$$

These numbers are in-sample RMSEs, so probably too small due to overfitting.

Interestingly, shallow trees do not work better than deeper trees.

In fact, the optimal depth in this case seems to be 4, 5, or 6. On the training data, 6 works slightly better than anything else.

Here is how the fit changes with d:

$$\begin{aligned} 1 &\longrightarrow 0.002655 \\ 2 &\longrightarrow 0.000578 \\ 3 &\longrightarrow 0.000352 \\ 4 &\longrightarrow 0.000313 \\ 5 &\longrightarrow 0.000302 \\ 6 &\longrightarrow 0.000298 \\ 7 &\longrightarrow 0.000302 \\ 8 &\longrightarrow 0.000299 \end{aligned} \tag{10}$$

So there is a great deal of improvement from 1 to 3, then modest further improvement up to 5, after which it is essentially flat.

The `gbm()` function has many options. It seems to run fast, at least for this modest-sized sample.

Bayesian Methods

Before we can discuss **Bayesian additive regression trees**, or **BART**, we need to say a little bit about Bayesian statistical methods.

The relationships between joint, marginal, and conditional probabilities tell us that, for any random variables A and B,

$$p(A, B) = p(A \mid B)p(B) = p(B \mid A)p(A). \tag{11}$$

Rearranging the second equality yields **Bayes' Rule**:

$$p(B \mid A) = \frac{p(A \mid B)p(B)}{p(A)}. \tag{12}$$

For Bayesian estimation, the sample $\boldsymbol{y}$ plays the role of A, and the parameter vector $\boldsymbol{\theta}$ plays the role of B.

Now replace B by the parameter vector $\boldsymbol{\theta}$ and A by the sample $\boldsymbol{y}$. Notice that we treat both $\boldsymbol{y}$ and $\boldsymbol{\theta}$ as random. In classical statistics, $\boldsymbol{\theta}$ is not treated as random.

Bayes' Rule tells us that

$$p(\boldsymbol{\theta} \mid \boldsymbol{y}) = \frac{p(\boldsymbol{y} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\boldsymbol{y})}. \tag{13}$$

Here $p(\boldsymbol{\theta} \mid \boldsymbol{y})$ is the **posterior density**, $p(\boldsymbol{\theta})$ is the **prior density**, and $p(\boldsymbol{y} \mid \boldsymbol{\theta})$ is the **likelihood function**.

All of the sample information enters via the likelihood function.

Whatever other information the investigator has enters explicitly via the prior and implicitly via the model specification, which determines the likelihood.

The denominator in (13) is the unconditional density of $\boldsymbol{y}$. Since it does not involve $\boldsymbol{\theta}$, we can ignore it for many purposes and simply write

$$p(\boldsymbol{\theta} \mid \boldsymbol{y}) \propto p(\boldsymbol{y} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}). \tag{14}$$

So the posterior is proportional to the likelihood times the prior.

Performing a Bayesian analysis involves three things:

1. Specify the model, and hence the likelihood function $p(\boldsymbol{y} \mid \boldsymbol{\theta})$.
2. Specify the prior $p(\boldsymbol{\theta})$, which may be a density or a discrete distribution. The prior may be:
   - an **informative prior**, which explicitly uses prior information and may rule out some values of $\boldsymbol{\theta}$;
   - a **noninformative prior** or **diffuse prior**, which attempts not to use prior information and may be **improper**, such as $\mathrm{U}[-\infty, \infty]$;
   - A **data-based prior**, which leads to **empirical Bayesian methods**.
3. Obtain the posterior density or distribution, or at least the features of it that we care about. Often, we care about
   - the posterior mean, and perhaps also higher moments;
   - certain quantiles of the empirical distribution, which will allow us to construct **credible intervals**;
   - the shape of the posterior distribution.

A Bayesian credible interval is what most people incorrectly think a confidence interval is. With specified (posterior) probability, a parameter lies within such an interval.

Suppose we are interested in $\gamma = g(\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a parameter vector and $g(\cdot)$ is some function. If we know $p(\boldsymbol{\theta} \mid \boldsymbol{y})$, we should be able to figure out $p(\gamma \mid \boldsymbol{y})$, or at least simulate it.

Then a $(1-\alpha)\%$ credible interval is any interval $[\gamma_l, \gamma_u]$ such that

$$p(\gamma_l \leq \gamma \leq \gamma_u \mid \boldsymbol{y}) = \int_{\gamma_l}^{\gamma_u} p(\gamma \mid \boldsymbol{y}) d\boldsymbol{y} = 1 - \alpha. \tag{15}$$

Whenever $p(\gamma \mid \boldsymbol{y})$ is continuous, there must be an infinite number of credible intervals that satisfy (15).

We could take the interval between the $\alpha/2$ and $1-\alpha/2$ quantiles, but we could also take the interval between the $\alpha/2+\delta$ and $1-\alpha/2+\delta$ quantiles for all δ such that $|\delta| \leq \alpha/2$.

The shortest credible interval is called the **highest posterior density interval**. If $p(\gamma \mid \boldsymbol{y})$ were symmetric around the posterior mean, the limits of this interval would be the $\alpha/2$ and $1-\alpha/2$ quantiles.

But most posterior densities are not symmetric, so HPD intervals are rarely symmetric. Instead, they will have the property that $p(\gamma_l \mid \boldsymbol{y}) = p(\gamma_u \mid \boldsymbol{y})$. They are *not* equal-tail intervals.

- Unless both the prior and the likelihood take certain rather simple functional forms, it is extremely difficult to obtain the posterior analytically.
- However, various procedures called **Markov Chain Monte Carlo**, or **MCMC**, methods can be used to obtain drawings from the posterior distribution.
- Once we have a large number of such draws, we can use standard frequentist methods to obtain estimates of the posterior mean, posterior standard error, posterior quantiles, or any other features of the posterior distribution.

Markov Chain Monte Carlo Methods

Almost all modern Bayesian work uses MCMC methods. BART is just one of many examples.

One class of MCMC methods is based on the **Gibbs sampler**.

Consider a general model, in which $\boldsymbol{\theta}$ is the parameter vector, and

$$p(\boldsymbol{\theta} \mid \boldsymbol{y}) \propto p(\boldsymbol{y} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}), \tag{16}$$

where $\boldsymbol{\theta}$ is a $p-$vector. For the linear regression model, $\boldsymbol{\theta} = [\boldsymbol{\beta} \vdots \sigma]$.

Let $\boldsymbol{\theta}$ be partitioned into B blocks:

$$\left[\boldsymbol{\theta}_{(1)} \vdots \boldsymbol{\theta}_{(2)} \vdots \ldots \vdots \boldsymbol{\theta}_{(B)}\right). \tag{17}$$

For the linear regression model, it is natural to set $B = 2$, with $\boldsymbol{\theta}_{(1)} = \boldsymbol{\beta}$ and $\boldsymbol{\theta}_{(2)} = \sigma$.

Now consider the **full posterior conditional distributions**

$$p\left(\boldsymbol{\theta}_{(1)} \mid \boldsymbol{y}, \boldsymbol{\theta}_{(2)}, \ldots, \boldsymbol{\theta}_{(B)}\right),$$
$$p\left(\boldsymbol{\theta}_{(2)} \mid \boldsymbol{y}, \boldsymbol{\theta}_{(1)}, \boldsymbol{\theta}_{(3)}, \ldots, \boldsymbol{\theta}_{(B)}\right),$$

and so on.

Drawing sequentially from these full posterior conditional distributions is called **Gibbs sampling**.

The trick is that we don't attempt to draw the entire vector $\boldsymbol{\theta}$ at once. Instead, we draw every block conditional on all the others.

Suppose that $B = 2$, and we have a random draw $\boldsymbol{\theta}_{(1)}^{[0]}$ from the joint posterior distribution $p(\boldsymbol{\theta} \mid \boldsymbol{y})$. Then a random draw $\boldsymbol{\theta}_{(2)}^{[1]}$ from

$$p\left(\boldsymbol{\theta}_{(2)} \mid \boldsymbol{y}, \boldsymbol{\theta}_{(1)}^{[0]}\right) \tag{18}$$

must also be a valid draw from $p(\boldsymbol{\theta} \mid \boldsymbol{y})$.

Next, we can draw $\boldsymbol{\theta}_{(1)}^{[1]}$ from

$$p\left(\boldsymbol{\theta}_{(1)} \mid \boldsymbol{y}, \boldsymbol{\theta}_{(2)}^{[1]}\right). \tag{19}$$

Then we draw $\boldsymbol{\theta}_{(2)}^{[2]}$ from

$$p\left(\boldsymbol{\theta}_{(2)} \mid \boldsymbol{y}, \boldsymbol{\theta}_{(1)}^{[1]}\right), \tag{20}$$

and so on.

At each step, we are simply drawing a block from one of the full posterior conditional distributions, rather than drawing the entire vector $\boldsymbol{\theta}$ from the posterior itself.

We generally cannot draw $\boldsymbol{\theta}_{(1)}^{[0]}$ from the joint posterior distribution.

But that does not matter! Under weak conditions, the process converges to a sequence of draws from $p(\boldsymbol{\theta} \mid \boldsymbol{y})$ no matter where we start from.

We just pick $\boldsymbol{\theta}_{(2)}^{[0]}$ in some way, then let the process run for S_0 **burn-in replications**, which are discarded, and finally let it run for a further S replications which are actually used.

If $g(\cdot)$ is a function of interest, then in many cases our estimate is

$$\hat{g}_S \equiv \frac{1}{S} \sum_{s=1}^{S} g(\boldsymbol{\theta}^{(s)}). \tag{21}$$

For quantiles, we have to sort the $g(\boldsymbol{\theta}^{(s)})$ and take appropriate values from the sorted list.

Here the burn-in replications are numbered from $1 - S_0$ to 0, and the ones that are used are numbers 1 through S. Note that $\hat{g}_S$ is a frequentist estimator.

There are two practical problems with Gibbs sampling that do not arise when we can actually draw from the posterior distribution.

1. We need to make sure that the effect of $\boldsymbol{\theta}^{(0)}$ has worn off. Hence the burn-in period.
2. The successive draws of $\boldsymbol{\theta}^{(s)}$, $\boldsymbol{\theta}^{(s+1)}$, and so on will not be independent.
   - Sample averages will be more random than they would be if the draws were independent, and so we will need S to be larger.
   - Conventional standard errors will be wrong. We need to allow for serial dependence.
   - But we may not care about standard errors if S is big enough.

In practice, it is often a good idea to start with more than one initial guess at $\boldsymbol{\theta}_{(1)}^{[0]}$.

If the sequence of random draws converges to the joint distribution, as it should in theory, then we will get essentially the same results whatever values we start from.

The Gibbs sampler is an example of a Markov chain because the draw of any block at time s depends on the draws of other blocks at s or at $s-1$, but not on anything further in the past.

Other posterior simulators, which are generally more complicated, also have the Markov property.

There is a large literature on how to perform MCMC efficiently and to check that inferences based on it are reliable.

In econometrics, John Geweke has been a major contributor to this literature.

Various diagnostic measures (all of them frequentist) can be used to decide whether the chain has settled down.

Sometimes, the posterior may have two or more modes. If they are a long away apart, and the density is very low between them, the Markov chain can get stuck in the neighbourhood of one mode.

Bayesian Additive Regression Trees

This is a newish method not discussed in ISLR1. The key paper is:

H. A. C. Chipman, E. I. George, and R. E. McCulloch, "BART: Bayesian additive regression trees," *Annals of Applied Statistics*, 2010, 4, 266–298.

Like random forests, BART constructs new trees randomly.

Like boosting, it tries to construct new trees in a way that captures signal not already captured by the current model.

It uses an MCMC procedure.

The user decides in advance how many trees (K in ISLR/ISLP, m in the original paper) there will be and how many iterations (B) to let the algorithm run for. [Here $B = S + S_0$]

Typically, m will be a few hundred, B will be a big number, like 2,000 or even 10,000, and we will drop the first L iterations (say 200 or 500) as burn-in ones. [Here $L = S_0$]

The output of BART is a collection of prediction models:

$$\hat{f}^b(\boldsymbol{x}) = \sum_{j=1}^{m} \hat{f}_j^b(\boldsymbol{x}),\ b = 1, \ldots, B. \tag{22}$$

The final prediction is simply the average over all iterations after the burn-in period:

$$\hat{f}(\boldsymbol{x}) = \frac{1}{B-L} \sum_{b=L+1}^{B} \hat{f}^b(\boldsymbol{x}). \tag{23}$$

The BART algorithm is given on page 351 of ISLR/ISLP.

Key features are that all trees start out identical, every tree is restricted by the prior to be quite short, and new trees are fitted one at a time to the current **partial residual**.

New trees involve random perturbations that improve the fit using a method called **Bayesian backfitting**.

The BART package provides the `gbart` function for regression trees and the `pbart` and `lbart` functions for probit and logit trees.

In principle, m (the number of trees), along with various hyper-parameters, could be chosen by cross-validation.

However, the authors claim that $m = 200$ with default values of the hyper-parameters usually works well.

There are several hyper-parameters, so cross-validation can be expensive.

Using a very small value of m works badly, but using a very large value only hurts prediction slightly.

- As for other MCMC methods, the output of BART is a sequence of random draws that can be treated as realizations from a Bayesian posterior distribution.
- For BART, each of these $B - L$ draws is $\hat{f}^b(\boldsymbol{x})$, a sum of K trees. The package uses `nskip` for L and `ndpost` for $B - L$.

One very nice feature is that we can report *any* function of the $\hat{f}^b(\boldsymbol{x})$.

- The usual prediction is the mean, $\hat{f}(\boldsymbol{x})$, given in (23).
- But we could also report the median, the standard deviation, or any quantiles of interest.

Obtaining a 95% prediction interval is trivial. The lower limit is the .025 quantile of the $\hat{f}^b(\boldsymbol{x})$, and the upper limit is the .975 quantile.

To make obtaining these easy, choose $B - L + 1$ to be a multiple of 200.

For example, if $B - L = 3999$, we sort the $\hat{f}^b(\boldsymbol{x})$ and take numbers 100 and 3900 as the .025 and .975 quantiles.

- Another useful thing that can be obtained is the number of times that each variable appears in each sum of trees.
- Variables that appear in many trees presumably matter more than variables that appear in just a few trees.

Of course, since the package simply reports raw results for each of the $B - L$ draws, some `R` programming is required.