CKW

Statistics 8330: Data Analysis III
Boosting

Suggested Reading: James, Witten, Hastie, and Tibshirani (JWHT), 2013, Chapter 8.2.3

Supplemental Reading: Hastie, Tibshirani, and Friedman (HTF), 2009, Chapter 10

Christopher K. Wikle

University of Missouri
Department of Statistics

Boosting

Boosting, like bagging, is a methodology that can improve the predictions from decision trees (and other statistical learning methods). In many ways, boosting is even more powerful.

Recall that bagging involves creating multiple bootstrap copies of the training data set and fitting a separate decision tree to each copy, with the ultimate prediction corresponding to an average of these bootstrap trees. Each tree is built independently of the other trees.

Boosting works similarly, but the trees are not grown independently; rather they are grown *sequentially*. There are no bootstrap samples with boosting as we essentially build trees on the residuals in a step-by-step manner.

We first consider boosting for regression trees more informally and then consider some of the issues associated with the more general procedure.

Boosting (cont.)

CKW

The basic idea is that we do not fit a single large decision tree to the data (known as *"fitting the data hard"* in the literature). Rather, we apply a procedure that *learns slowly*.

Specifically, given the current model, we fit a decision tree to the residuals from the model. So, the residuals (not the outcome Y) are used to fit a decision tree. This helps to improve the parts of the tree that did not perform well in the existing tree structure. We then add this new decision tree into the fitted function, often with a shrinkage parameter (λ) that "slows" the process down so that additional trees can help model these areas that don't fit well (which helps to prevent over-fitting).

Unlike bagging, the construction of each tree depends strongly on the trees that have already been grown.

Boosting: Basic Algorithm

Consider the basic algorithm for boosting regression trees (JWHT, 2013):

Algorithm 8.2 *Boosting for Regression Trees*

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

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Boosting (cont.)

In this framework, boosting typically has three tuning parameters:

- B: the number of trees. Boosting can over fit if B is too large so we use cross-validation to select B.
- λ: the shrinkage parameter. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, with the best choice depending on the data at hand. Typically, the smaller the λ, the larger B must be to achieve good performance.
- d: the number of splits in each tree. Surprisingly, often $d = 1$ works well (in this case, we call the tree a "stump"). With $d = 1$, the boosted ensemble is fitting an additive model since each term only involves one variable. More generally, d is the *interaction depth* and controls the interaction order of the boosted model (since d splits can involve at most d variables).

Boosting (cont.)

Given that boosting is one of the most powerful statistical learning approaches, it is useful to look into the approach in a bit more detail. It was originally designed for classification problems (although, as we have seen, it can easily be applied to regression settings). In the jargon of machine learning, we say that this method combines the output of many "weak" learners to produce a powerful "committee."

A very popular boosting algorithm for classification is known as "AdaBoost.M1" (Freund and Schapire, 1997). We describe this in some detail here (see HTF 2009, 10.1 for more detail).

Boosting (cont.)

Consider a two-class classification code such that $Y \in \{-1, 1\}$, and a vector of predictor variables, X. We have a classifier $G(X)$ that produces a prediction that selects one of the two values $\{-1, 1\}$. The error rate on the training sample is

$$\overline{err} = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq G(x_i)).$$

A weak classifier is one whose error rate is only slightly better than random guessing. Now, boosting sequentially applies a weak classification algorithm to produce a sequence of weak classifiers, $G_m(x)$, $m = 1, 2, \ldots, M$ (note: M is B above). Critically, the predictions from all of these are then combined through a *weighted majority vote* to produce the final prediction:

$$G(x) = \text{sign} \left(\sum_{m=1}^{M} \alpha_m G_m(x) \right),$$

where the weights $\alpha_m, m = 1, \ldots, M$, are computed by the boosting algorithm and give highest influence to the best classifiers.

Boosting (cont.)

CKW

At each boosting step, weights $w_1, \ldots, w_n$ are applied to the training observations $(x_i, y_i), i = 1, \ldots, n$. To start, these are all set equal, $w_i = 1/n$ so there is no preference for the first classification. For each successive iteration, the observation weights at iteration m are modified such that those observations that were misclassified by the classifier $G_{m-1}(x)$ have their weights increased, and those that were correctly classified have their weights decreased.

Thus, as iterations proceed, observations that are difficult to classify correctly receive ever-increasing influence. So, each successive classifier is forced to concentrate on those training observations that are missed by the previous classifiers in the sequence.

This procedure works very well. In fact, Breiman (1996,1998) stated that Adaboost using trees was the "best off-the-shelf classifier in the world"!

Boosting (cont.)

CKW

Consider the sketch of the AdaBoost.M1 algorithm from HTF (2009):

Algorithm 10.1 *AdaBoost.M1.*

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

Boosting (cont.)

Consider the results from a simulated example presented in HTF (2009):

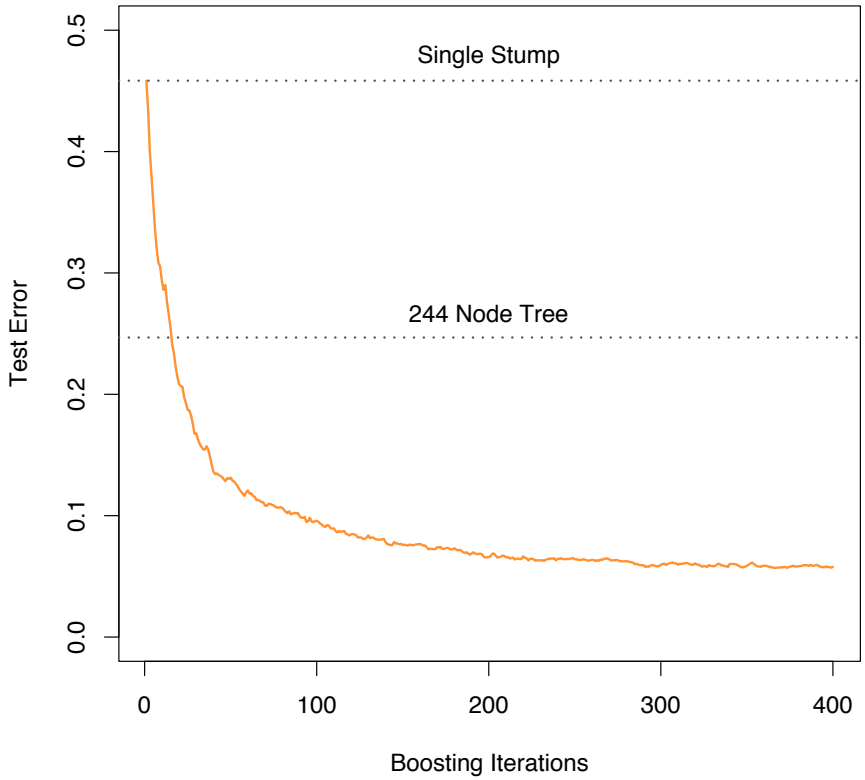

FIGURE 10.2. *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 244-node classification tree.*

Boosting (cont.)

CKW

The great success of boosting since its creation in the early 1990s has led to vigorous scientific debate as to the reasons WHY it is so successful (a good problem to have!). HTF (2009, Section 10.2-10.5) outline one explanation for this, summarized briefly below.

Recall that many of the methods we have discussed so far can be thought of from the perspective of a basis function expansion:

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m),$$

where β_m are the expansion coefficients and $b(x, \gamma)$ are simple functions of the multivariate inputs x and parameters γ. One can consider boosting to be a way of fitting such an additive expansion where the basis functions are the individual classifiers $G_m(x) \in \{-1, 1\}$.

Boosting (cont.)

CKW

As we talked about earlier in the class, models such as these are fit by minimizing some loss function averaged over the training data (e.g., such as squared error loss). More generally,

$$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{n} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\right).$$

For many loss functions $L(y, f(x))$ and basis functions $b(x;\gamma)$ this can require a very computationally expensive numerical optimization algorithm. In some cases, it may be possible to rapidly solve the subproblem of fitting just a single basis function,

$$\min_{\{\beta,\gamma\}} \sum_{i=1}^{n} L\left(y_i, \beta b(x_i;\gamma)\right).$$

Boosting (cont.)

Then, one can approximate the more general minimization problem by sequentially adding new basis functions to the expansion without adjusting the parameters and coefficients that have already been added. This is called *forward stagewise additive modeling*. That is, at each iteration m, one solves for the optimal basis function $b(x;\gamma_m)$ and corresponding coefficient β_m to add to the current expansion $f_{m-1}(x)$. This gives $f_m(x)$ and one repeats the process – previously added terms are not modified.

For example, for squared error loss

$$L(y, f(x)) = (y - f(x))^2,$$

and then

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta_m b(x_i;\gamma_m)) &= (y_i - f_{m-1}(x_i) - \beta_m b(x_i;\gamma_m))^2 \\ &= (r_{im} - \beta_m b(x_i;\gamma_m))^2, \end{aligned}$$

where $r_{im} = y_i - f_{m-1}(x_i)$ is just the residual of the current model on the ith observation. So, the term $\beta_m b(x;\gamma_m)$ that best fits the current residuals is added to the expansion at each step.

Boosting (cont.)

It can be shown (see HTF 2009, 10.4) that AdaBoost.M1 is equivalent to forward stagewise modeling using the loss function

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

which is called "exponential loss."

It can be shown that this loss function is attractive in that it facilitates computation.

It can also be shown that this loss function is closely related to the binomial negative log-likelihood (deviance), although they have different properties in finite data (real-world) settings.

Boosting (cont.)

$\mathcal{CKW}$

There is a substantial (and growing) literature on the choice of different loss functions for statistical learning algorithms. In some cases one prefers to be more robust to certain properties (e.g., outliers) and in other cases, one chooses a loss function to facilitate computation.

Note that when the loss function is differentiable, one can make use of the gradient of the loss function to improve the optimization; this is the basis of *gradient boosting* ("gradient boosting models (GBM)"; formerly, Multiple Additive Regression Trees (MART)).

Partial Dependence Plots

CKW

As mentioned previously, it becomes difficult with bagging and boosting to interpret parameter effects from tree-based models. We discussed the possibility of using summaries of variable relevance last lecture. We can also consider the notion of *partial dependence plots* as a way to help visualize effects associated with a particular variable or variables.

Consider a subvector X_S of $\ell < p$ of the input predictor space X, indexed by $S \subset \{1, 2, \ldots, p\}$. Let C be the complement such that $S \cup C = \{1, 2, \ldots, p\}$. We expect that $f(X) = f(X_S, X_C)$ and define the *partial* dependence of $f(X)$ on X_S to be

$$f_S(X_S) = E_{X_C} f(X_S, X_C),$$

which is just the marginal average of f, and can serve as a useful description of the effect of the chosen subset on $f(X)$ (typically, when elements of X_S do not have strong interactions with elements of X_C).

Partial Dependence Plots (cont.)

We can estimate this expectation by

$$\bar{f}_S(X_S) = \frac{1}{n}\sum_{i=1}^{n} f(X_S, x_{iC}),$$

where $\{x_{1C}, x_{2C}, \ldots, x_{nC}\}$ are the values of X_C occurring in the training data. This can be computationally intensive to evaluate (but, for decision trees, $\bar{f}_S(X_S)$ can be rapidly computed from the tree itself).

Note, the interpretation of the partial dependence plots is that they represent the effect of X_S on $f(X)$ after accounting for the average effects of the other variables X_C on $f(X)$ (recall the partial regression plots in Data Analysis I). You cannot assume that this represents the effect of X_S on $f(X)$ *ignoring* the effects of X_C (unless X_S and X_C are independent)!

Example: California Housing (Pace and Barry, 1997)

HTF (2009)

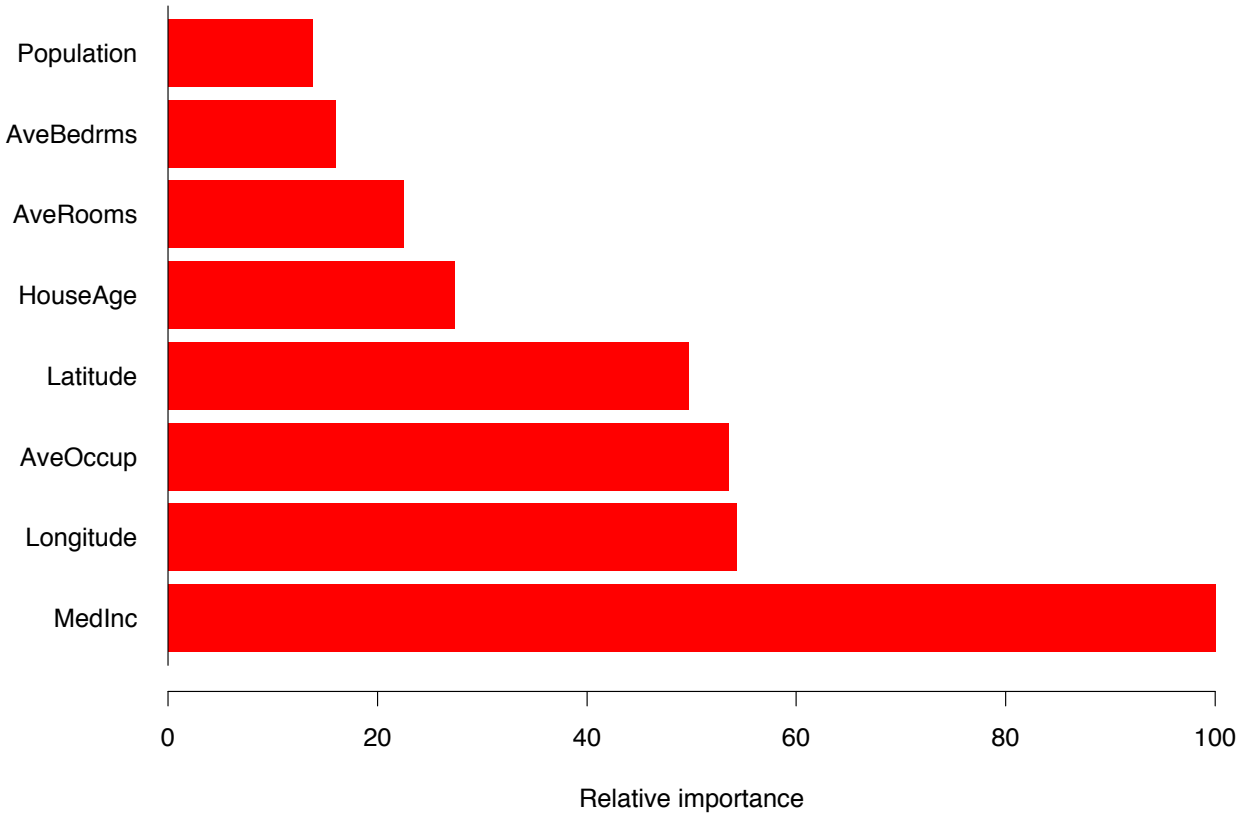

FIGURE 10.14. *Relative importance of the predictors for the California housing data.*

Example: California Housing (cont.)

HTF (2009)

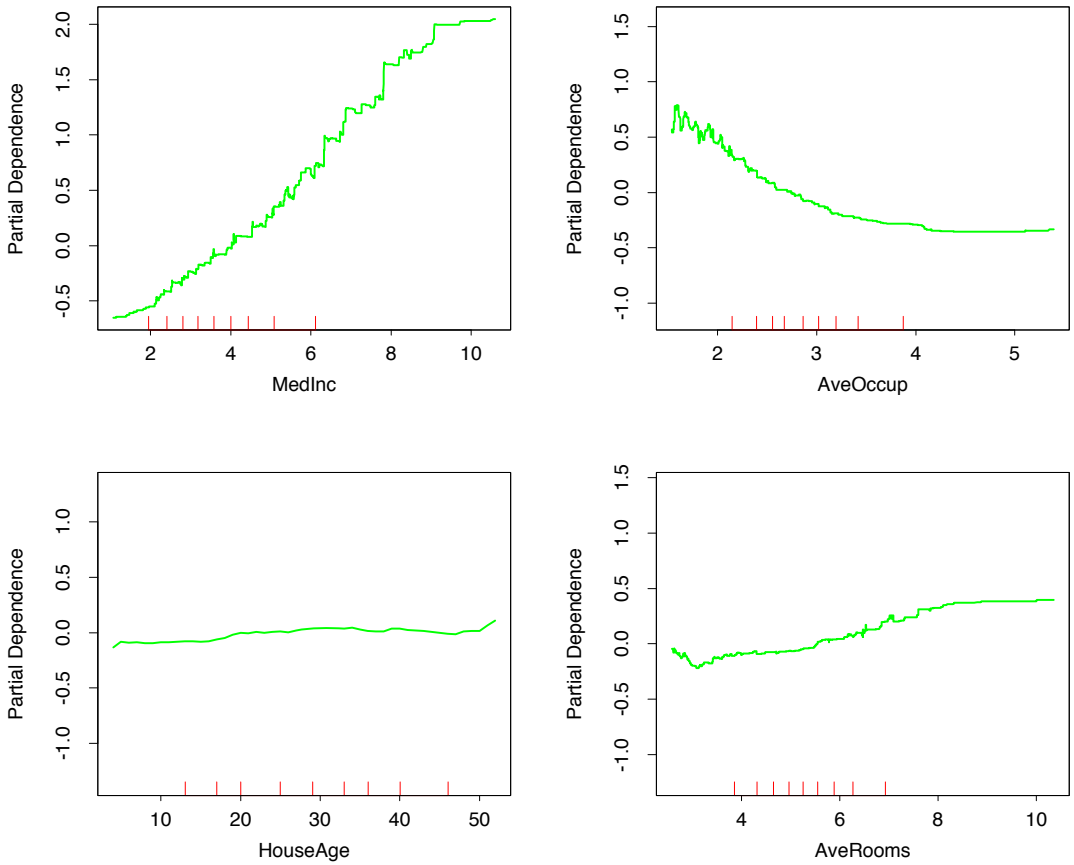

FIGURE 10.15. *Partial dependence of housing value on the nonlocation variables for the California housing data. The red ticks at the base of the plot are deciles of the input variables.*

Example: California Housing (cont.)

CKW

HTF (2009)

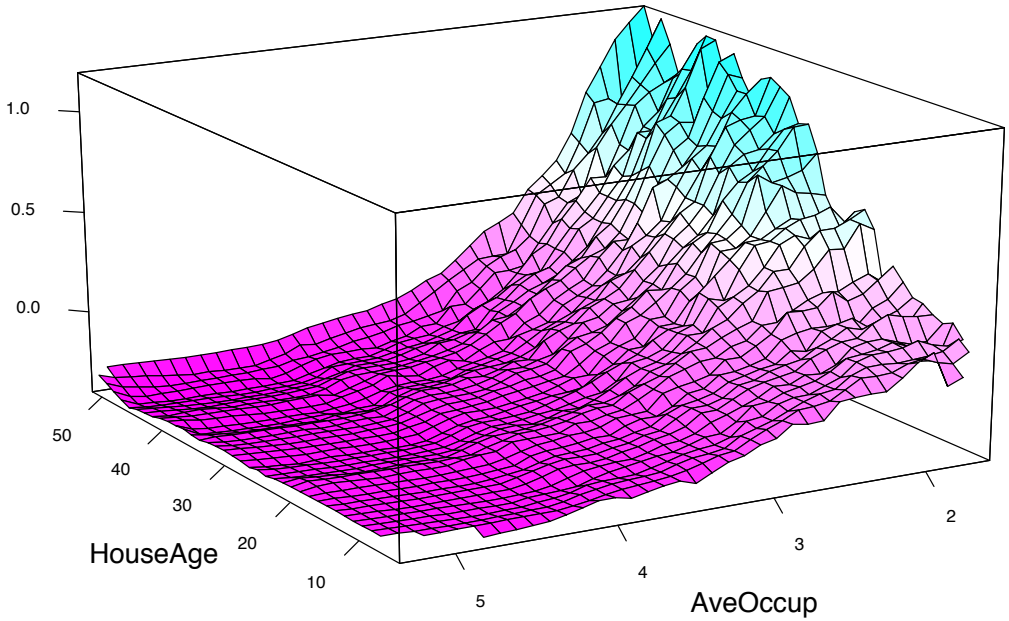

FIGURE 10.16. *Partial dependence of house value on median age and average occupancy. There appears to be a strong interaction effect between these two variables.*

Example: California Housing (cont.)

CKW

HTF (2009)

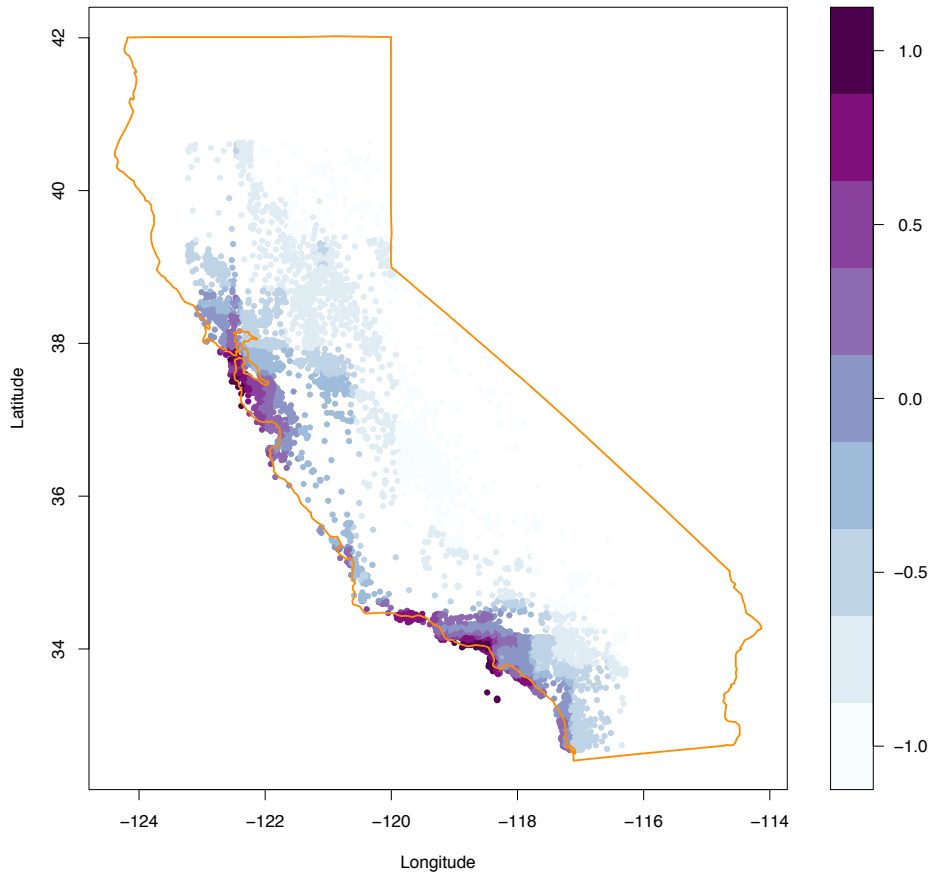

FIGURE 10.17. *Partial dependence of median house value on location in California. One unit is* $\$100,000$*, at* 1990 *prices, and the values plotted are relative to the overall median of* $\$180,000$.