9

Data Exploration Using Non-Spatial Methods: Nonparametric Methods

9.1 Introduction

The previous chapter focused on regression models as a means of data exploration. These are called *parametric* methods because they include assumptions about the form of the error terms, typically that these errors are normally distributed with mean zero and fixed variance σ^2, and part of the analysis involves estimation of the value of the parameter σ^2. The methods of this chapter involve nonparametric statistical analysis, in which these assumptions about the distributional form of the error are not made. As with the methods of Chapter 8, we classify these methods as exploratory since one still typically assumes that the errors are independent, and spatial data may violate this assumption. Confirmatory analysis will be carried out in later chapters in which the analysis incorporates assumptions about the spatial relationships among the data and the errors. This chapter presents a collection of methods, any one of which may be applied to spatial data. I have found generalized additive models (GAMs), recursive partitioning (also known as classification and regression trees or CART), and random forest to be particularly useful in a variety of applications, and therefore these are the focus of this chapter. Section 9.2 deals with GAMs, Section 9.3 deals with recursive partitioning, and Section 9.4 deals with random forest.

Before going on, however, it is worthwhile to point out a famous quote by John Tukey (1986): "The combination of some data and an aching desire for an answer does not ensure that a reasonable answer can be extracted from a given body of data." There is an understandable tendency in all of us to, if our analysis does not initially turn out the way we had hoped it would, move on to a more complicated or sophisticated model in the hope that doing so would reveal patterns that the simpler methods had missed, and sometimes this does happen. That said, the methods described in this chapter are not a panacea and are not guaranteed to produce interesting or meaningful results. They will often produce results, however, and it is up to you to make sure that the results make biophysical sense.

9.2 The Generalized Additive Model

This section contains a brief description of the GAM, introduced by Hastie and Tibshirani (1986). The section is intended to introduce the reader to the GAM, and to do some preliminary exploration of the data. It is *not* intended to provide a thorough introduction

to the subject. To do so in a way that would ensure that a GAM could be applied appropriately would require far more space than is available. If, upon reading the material in the section and doing a preliminary exploration such as is presented here, you feel that a GAM would be appropriate for further study, the standard reference, which is an excellent one, is Wood (2006). There are some easily identifiable cases in which the GAM does have certain specific advantages, and for this reason we will look at a simple example to illustrate one of these. This example is based on the plot of blue oak presence vs. elevation in the Coast Range shown in Figure 7.11. Our analysis follows that of Yee and Mitchell (1991).

A good way to introduce the GAM is to compare it to the models we have already seen. Suppose that there are two potential explanatory variables, X_1 and X_2, which do not interact. In this case the linear model is (cf. Equation 8.8).

$$E\{Y_i\} = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2}. \tag{9.1}$$

The generalized linear model (GLM) is (cf. Equation 8.28)

$$g(\pi_i) = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2}, \tag{9.2}$$

where g is an appropriate link function and $\pi_i = E\{Y_i\}$. In this context, the corresponding generalized additive model would be written

$$g(\pi_i) = \beta_0 + f_1(X_{i1}) + f_2(X_{i2}), \tag{9.3}$$

where again g is an appropriate link function and again $\pi_i = E\{Y_i\}$. The functions f_1 and f_2 are smooth functions, which for our purposes means that they are continuous and have continuous first and second derivatives, so that intuitively they are continuous and have no kinks. To ensure that they are unique, it is required that $f_1(0) = f_2(0) = 0$. Of course, these equations can be extended to a larger number of variables X_i and functions $f_i(X)$. The default method used by `ggplot()` for the geom `geom _ smooth()` is a GAM.

In Exercise 8.12, you are asked to construct a generalized linear model analysis for the Coast Range blue oak data. Figure 7.11 indicates that the fraction of sites in which a blue oak occurs reaches a maximum between about 800 m and 1200 m and then declines (leaving aside an apparent anomaly at 1900 m). We will carry out a bit of that analysis here. We begin by fitting generalized linear models to these data (minus the anomaly at 1900 m) using the function `glm()`. To capture the nonmonotonic relationship between elevation and oak presence our initial model uses the link function

$$g(\pi_i) = \beta_0 + \beta_1 X_i + \beta_2 X_i^2. \tag{9.4}$$

The resulting plot is shown as the solid line in Figure 9.1. It fits the data at the lower elevations reasonably well but does a very poor job at the higher elevations. We can attempt to improve the fit by adding another polynomial degree, that is, using the link function

$$g(\pi_i) = \beta_0 + \beta_1 X_i + \beta_2 X_i^2 + \beta_3 X_i^3, \tag{9.5}$$

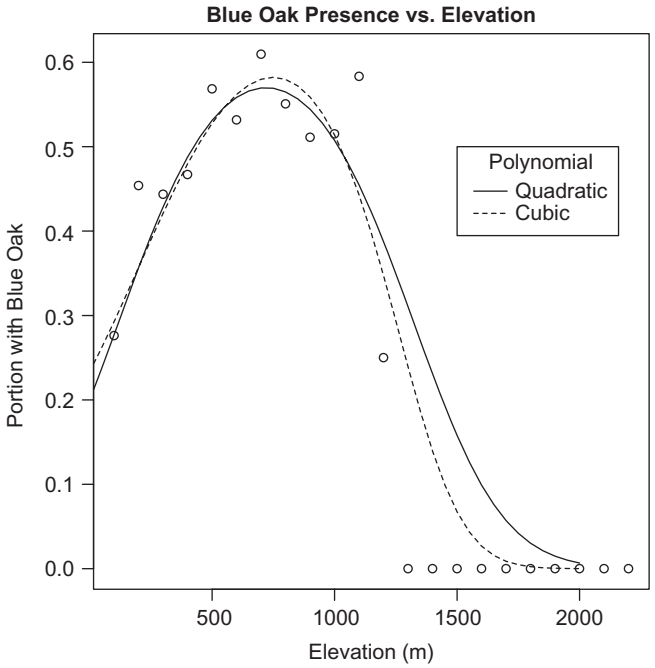

FIGURE 9.1
Plot of generalized linear models of *QUDO* vs. *Elevation* fit to Coast Range subset of the blue oak data from Data Set 2. Also shown are 100 m running averages of fraction of sites occupied by a blue oak.

but, as indicated by the dashed line in the figure, this does not really improve the fit much. Let's see how using a generalized additive model of the form (Equation 9.3) can improve the fit.

The first step is to describe the smooth functions $f_i(X)$. The most commonly used functions, and the only ones we will discuss, are called *splines*. This term comes from a sort of flexible ruler that was used in the days of hand drafting to make irregularly shaped curves. Figure 9.2 shows an example of the most common type of spline (and the one we will use), called a cubic spline, fit to five points. Suppose we have a sequence of points (X_i, Y_i), $i = 1, 2, \ldots, n$. An *interpolating* spline is sequence of (for us) cubic functions, each of which are fit to two successive points in the sequence, such that the entire sequence of functions fits the sequence of points, as shown in Figure 9.2. For two successive points (X_i, Y_i) and (X_{i+1}, Y_{i+1}), the spline is a sequence of cubic functions $f_i(X)$ that satisfy

$$\begin{aligned} &f_i(X_i) = Y_i, f_i(X_{i+1}) = Y_{i+1}, \\ &f'_{i-1}(X_i) = f'_i(X_i), \\ &f''_{i-1}(X_i) = f''_i(X_i). \end{aligned} \tag{9.6}$$

Here the primes and double primes denote first and second derivatives, respectively. The sequence of points (X_i, Y_i) is called the *knots*, so in words, these spline functions interpolate the points, and their first and second derivatives match at the knots. There is nothing to match at the endpoints, so some other condition is imposed. A common one is $f''_1(X_1) = f''_{n-1}(X_n) = 0$, so that the curvature is zero at the end points. It turns out that the

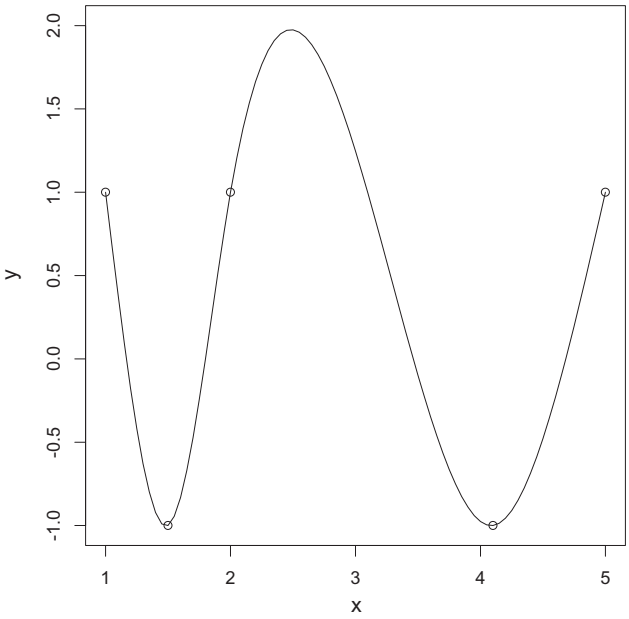

FIGURE 9.2
Plot of a cubic spline interpolation of five data points.

conditions of Equation 9.6 together with the end point conditions are sufficient to uniquely determine the functions (Hastie et al., 2009, Chapter 5).

The interpolating spline function described in the preceding paragraph is not quite what we need for application to the generalized additive model in Equation 9.3. We need something that could fit data, such as that shown in Figure 9.3. This data set was created by using the modulo operator `%%` to select one-twentieth of the data records of January mean temperature *JaMean* vs June minimum temperatures *JuMin* from the Coast Range data set (see Section 6.3.3 for the use of this operator in this manner). The objective with a data set such as this is not to interpolate it but rather to approximate it with a smooth curve. One way of doing this is by the use of *regression splines*.

We begin our discussion of regression splines by introducing the idea of a *basis*. In thinking about a basis, it is useful to think about the classical vectors i, j, and k that we all encountered in physics as describing coordinates in three-dimensional space. To say that these vectors form a basis for that space means that any vector v in this space can be described as a sum $v = i + j + k$. These are not the only possible basis vectors; indeed, any three linearly independent vectors can serve as a basis (cf. Appendix A.1.3).

The idea of a basis in regression splines is analogous. Consider the polynomial function $f(x) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$. The four functions 1, X, X^2, and X^3 form a basis for the space of all possible cubic polynomials. As with the basis vectors i, j, and k of physical three-dimensional space, we can make other bases out of linearly independent combinations of these four functions.

Suppose that we wish to construct cubic spline polynomials that we can use on the right-hand side of an equation like Equation 9.3 to create our generalized additive model. Since the spline functions are cubic polynomials, we can express them in terms of basis functions. Thus, the problem is to find the basis functions that satisfy the conditions (Equation 9.6).

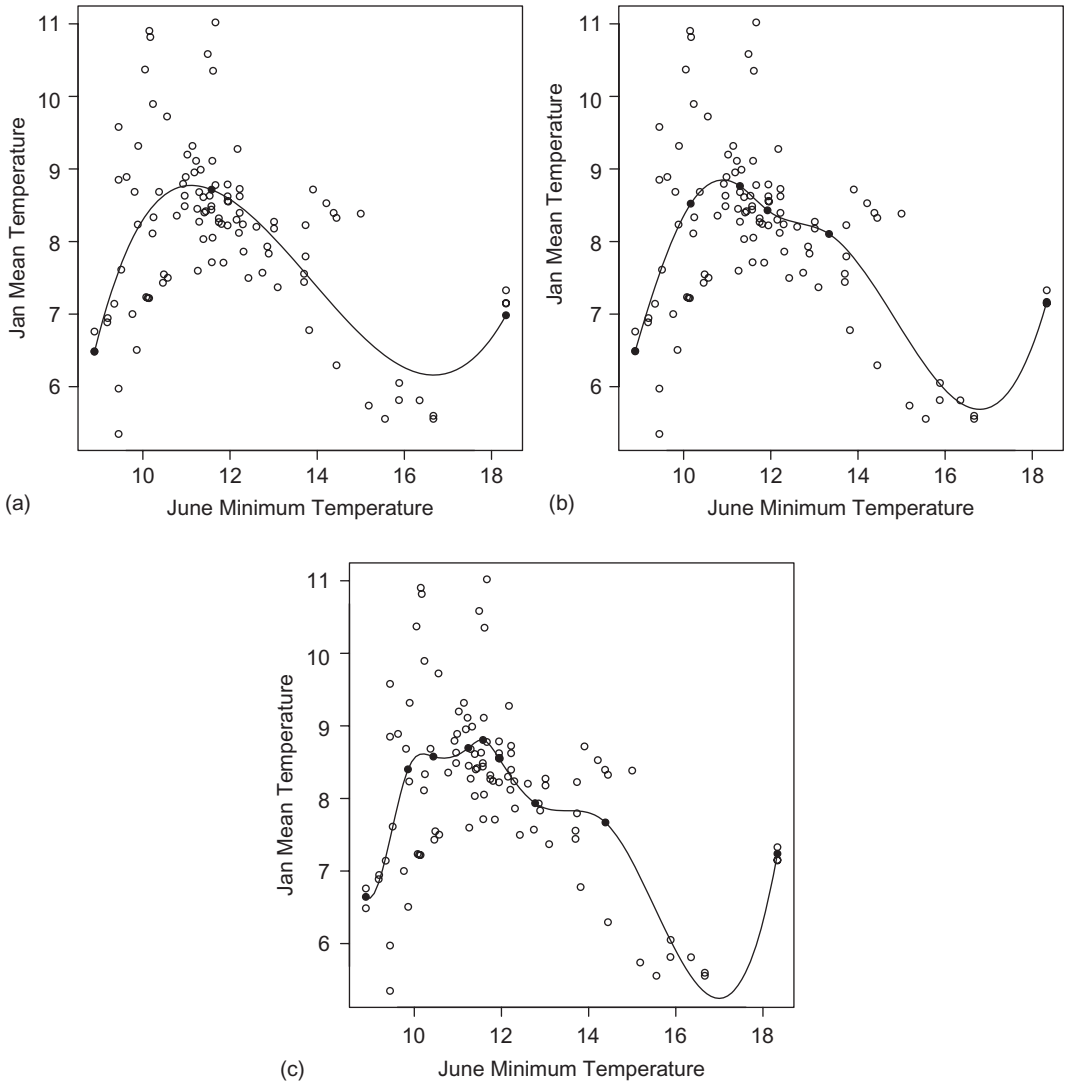

FIGURE 9.3
Cubic spline regression of the variables *JaMean* vs. *JuMin* of the Coast Range subset of the blue oak data from Data Set 2. (a) 1 interior knot; (b) 5 interior knots; (c) 7 interior knots.

A spline written in terms of these basis functions is called a *b-spline*, where the b stands for "basis." The actual form of these basis functions is, to quote Simon Wood (2006, p. 124), "rather opaque." Fortunately, the R package `splines`, which is included with the base R library, includes a function `bs()` that computes the spline basis for set of regression spline functions with a specified number of knots. The knots are placed at the percentiles of the data to ensure that each curve is based on a representative sample.

Because the spline functions are polynomials, a regression based on them is a linear regression (see the discussion following Equation A.37) in Appendix A if you have problems seeing this). Therefore, the function `lm()` from Chapter 8 can be used to compute the regression splines themselves. Figure 9.3a shows a regression with one interior knot. The code to make the data for this plot is as follows.

```
> library(splines)
> ID <- 1:nrow(coast.sf)
> data.plot <- coast.sf[which(ID %% 15 == 0),]
> # Remove a set of anomalous data values
> data.plot <- data.plot[-which((data.plot$JuMin > 11.111) &
+   (data.plot$JuMin < 11.12)),]
> with(data.plot, plot(JuMin, JaMean, xlab = "June Minimum
+   Temperature", ylab = "Jan Mean Temperature")) # Fig. 9.3a
```

The first three lines create the data set by extracting every fifteenth data record. The data set so obtained includes a set of anomalous values, all occurring at *JuMin* = 11.1111, so we remove these values. The code to create the spline is

```
> spl <- lm(JaMean ~ bs(JuMin, df = 4), data = data.plot)
```

The function `bs()` can accept multiple arguments in many forms. The default for the spline degree is 3 (i.e., cubic). The argument `df` specifies the number of degrees of freedom, which is the spline degree plus the number of knots. Thus `df = 4` specifies one knot. Note that the output of the function `bs()` is used directly as an argument of `lm()`. Next, we draw the spline and the knots.

```
> x <- seq(min(data.plot$JuMin), max(data.plot$JuMin), length.out =
+   100)
> lines(x, predict(spl, data.frame(JuMin = x)))
> v1 <- attr(terms(spl), "predvars")[[3]]
> v1
bs(JuMin, degree = 3L, knots = 11.577775, Boundary.knots = c(8.88889,
18.33333), intercept = FALSE)
> x = c(min(data.plot$JuMin),11.5778,max(data.plot$JuMin))
> points(x = x, y = predict(spl1, data.frame(JuMin = x)), pch = 19)
```

The interior knot is placed at the median value of the abscissa values *JuMin*, whose value to four decimal places is 11.577775. The extraction of this value is the subject of Exercise 9.1. As shown in the exercise, it is possible to extract the numerical values of the knots themselves from the function `bs()`.

Figure 9.3b and c show the effect of increasing the number of knots. The fit gets better, but the spline curves get much more "wiggly." If we call the number of knots k, then if k is too small, then the curve is too smooth, and if k is too large, then it is too wiggly. For a simple one-dimensional fit like this it is probably better to select the most appropriate curve by eye, but for more complex, multidimensional fits this may be impossible. Wood (2006, p. 131) recommends using the process of cross validation, which we have already seen in Section 6.3.3, to select the optimal value of k. Let a candidate spline function be denoted $\hat{f}(X)$, and let its value at the data point X_i be $\hat{f}(X_i) = \hat{f}_i$. We would like to choose the number of knots k to minimize the sum

$$M = \frac{1}{n}\sum_{i=1}^{n}(\hat{f}_i - f(X_i))^2. \tag{9.7}$$

The problem is that we do not know $f(X_i)$. Therefore, we estimate it by removing the pair (X_i, Y_i) from the data set and computing $\hat{f}_i^{(i)} = f^{(i)}(X_i)$ as the estimated value of Y_i obtained by fitting splines to this reduced data set. The *ordinary cross validation score* ν_o is then computed as

$$\nu_o = \frac{1}{n}\sum_{i=1}^{n}(\hat{f}_i^{(i)} - Y_i)^2. \tag{9.8}$$

This is sometimes called "leave one out" cross validation because the score is based on leaving one data pair out in turn, estimating that value, and then summing the sum of the squared differences of the results.

Wood (2006, p. 132) shows that a process called *generalized cross validation* has for this application both computational and statistical advantages over leave one out cross validation. It turns out, analogous to the computation of the *PRESS* statistic of Equation 8.17, that the value of v_o can be computed without actually going through the process of leaving out one data set at a time. It is equal to

$$\nu_o = \frac{1}{n}\sum_{i=1}^{n}(\hat{f}_i - Y_i)^2 / (1 - H_{ii})^2, \tag{9.9}$$

where H is the hat matrix of Equation A.36. Wood recommends using a slightly different quantity ν_g, given by

$$\nu_g = \frac{1}{n}\sum_{i=1}^{n}\frac{(\hat{f}_i - Y_i)^2}{(tr(I - H))^2}, \tag{9.10}$$

where the function $tr(A)$ of the matrix A is the *trace*, given by $tr(A) = \Sigma A_{ii}$, that is, the trace is the sum of the diagonal elements. The R core package `stats` contains a function `smooth.spline()` that implements a form of cross validation. This is a bit more sophisticated than the one described here and can return a non-integral value for the number of degrees of freedom. For our purposes, we can just round this value off to the nearest whole number.

```
> spl.sm <- with(data.plot, smooth.spline(JuMin, JaMean))
> round(spl.sm$df,0)
[1] 7
```

The optimal value for `df` is 7, the value shown in Figure 9.3b.

We are now ready to move on to the original purpose of this section, a discussion of the generalized additive model, or GAM, which by analogy with Equation 8.28, we write as

$$g(\pi_i) = \beta_0 + f_1(X_{i1}) + f_2(X_{i2}) + \ldots + f(X_{i,p-1}) \tag{9.11}$$

The GAM will use cubic splines for the functions $f_i(X)$. We begin with the simple application used to introduce the topic and depicted in Figure 9.1, the fitting of a logistic curve

to the Coast Range oak presence/absence vs. elevation data. The figure shows that a cubic polynomial does a better job than a quadratic in fitting the data, although still not a very good one. Since the spline automatically incorporates cubic polynomials, it is not necessary to incorporate them explicitly. Thus for this example we can write

$$g(\pi_i) = \beta_0 + f_1(Elevation_i), \tag{9.12}$$

There are three primary generalized additive model packages in R: `gam` (Hastie, 2017), `mgcv` (Wood, 2006), and `gss` (Gu, 2014). There are many similarities, and we will focus on the package `mgcv`. By default the `mgcv` function `gam()` uses *thin plate* splines, which are different from cubic splines. However, some experimentation (not shown) indicates that cubic splines work better in our particular case. The code to create the GAM fit with six knots shown in Figure 9.4 is

```
> library(mgcv)
> model.gam1 <- gam(QUDO ~ s(Elevation, bs = "cr", k = 8), data =
+    coast.cor, family = binomial)
```

The function call is very similar to that of `glm()`, except that the function `s()` is used to determine the characteristics of the spline function. The package `mgcv` allows for many options, and we will only scratch the surface. The argument `bs` specifies the spline basis. The default, as mentioned, is thin plate splines. The argument `bs = "cr"` specifies cubic splines. The argument `k` specifies the dimension of the basis, which is two more than the

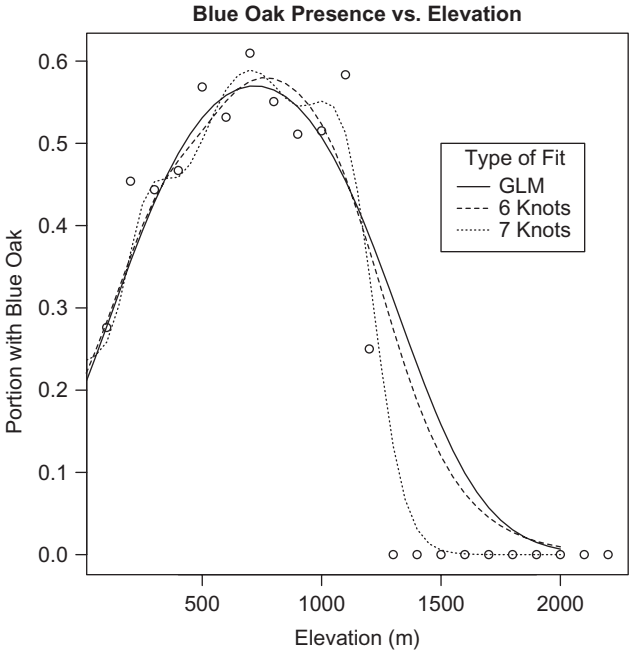

FIGURE 9.4
General linear model (GLM) and GAM fits using cubic splines with 6 and 7 knots, fit to of *QUDO* vs. *Elevation* data from the Coast Range subset of the blue oak data from Data Set 2.

number of knots. The data are difficult to fit because of the steep decline between 1200 and 1300 m. The spline with six knots gives a poor overall fit, while the spline with seven knots give a better overall fit although it is quite wiggly at the lower elevations.

As should be clear from the discussion so far, the objective in developing a GAM is not necessarily to find a better explanatory model, as has been the case with the methods we have discussed earlier. Rather, it is more to find a better fit to the data, that is, a better *predictive* model. In Section 8.4.2, we derived a generalized linear model for the Sierra Nevada data that included *Precip* and *MAT* as explanatory variables. Let's compare this model with a generalized additive model using the same data. We will use the same spline structure as was used for the Coast Range data.

```
> model.glmS2 <- glm(QUDO ~ MAT + Precip, data = sierra.sf, family =
binomial)
> model.gamS2 <- gam(QUDO ~ s(MAT, bs = "cr", k = 9) + s(Precip, bs =
+   "cr", k = 9), data = sierra.sf, family = binomial)
```

How should we do the comparison? The `mgcv` package includes an `anova()` function for GAMs, and since a GLM is based on a lower order polynomial model it is in this sense nested in a GAM, so in principle we can compare the two using this function.

```
> # This should be considered as only a rough approximation
> anova(model.glmS2, model.gamS2, test = "Chisq")
Analysis of Deviance Table

Model 1: QUDO ~ MAT + Precip
Model 2: QUDO ~ s(MAT, bs = "cr", k = 9) + s(Precip, bs = "cr", k = 9)
  Resid. Df Resid. Dev     Df Deviance Pr(>Chi)
1    1765.0    1158.3
2    1756.6    1108.8 8.4468   49.437 8.131e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As it says in the comment, this should be considered as only a rough approximation. The GLM model and the GAM model are computed by two different methods, so basically we are comparing apples and oranges. The p value should therefore not be considered reliable. That said, a value of approximately 10^{-7} is certainly pretty small. We can also compare histograms of the residuals of the two models. These histograms are shown in Figure 9.5 and indicate that the GLM model residuals have a more peaked distribution. A comparison that I find instructive is to plot approximate cross sections of the fit surface. The quantity *Precip* in the Sierra data set ranges from about 325 mm per year to about 2000 mm per year. There are 352 data records whose value is between 900 and 1000 mm. We will take all of these data records, set their value of *Precip* to 1000, and plot the fractional values of *QUDO* at successive intervals of *MAT* between its minimum of 5 and its maximum of 17, and plot the fits of the two models to these data. Figure 9.6 shows the resulting plot. The result is not unexpected. The spline fit of the GAM model is again more wiggly and thus follows the data more closely, for better or worse. Neither the GAM nor the GLM curve is sufficiently influenced by the zero values of *QUDO* at *MAT* values of 16 and 17 to prevent it from taking off into the stratosphere.

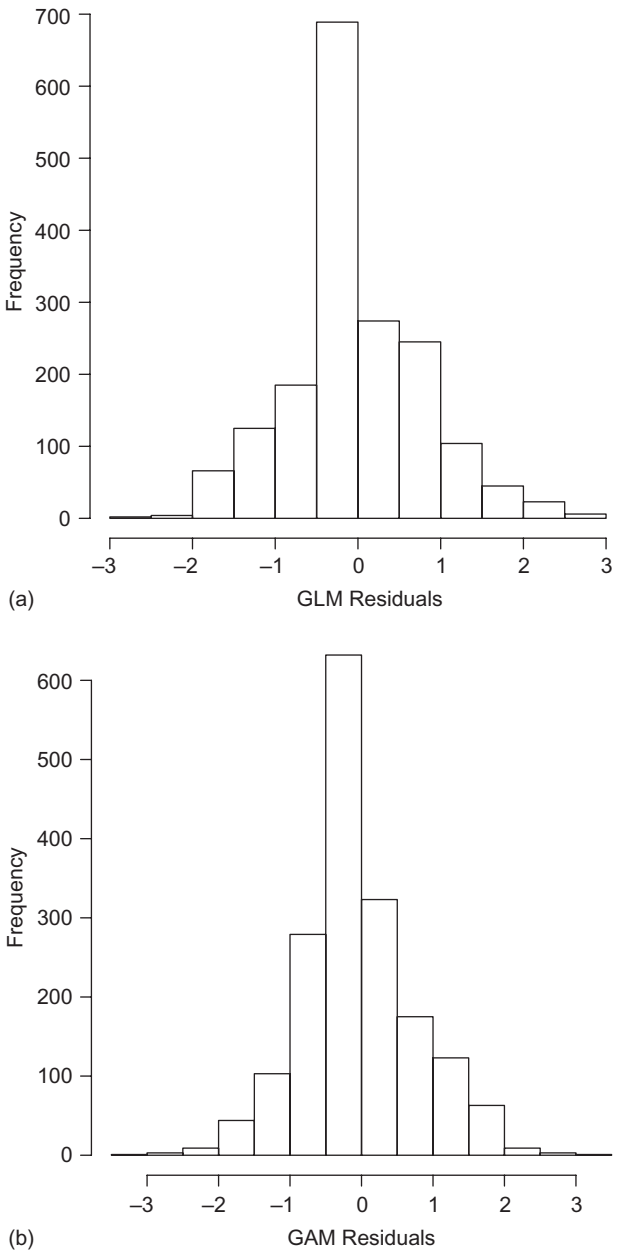

FIGURE 9.5
Histograms of the residuals from (a) GLM and (b) GAM fits of *QUDO* to *Precip* and *MAT* data from the Sierra Nevada subset of the blue oak data from Data Set 2.

In summary, the GAM is clearly better suited for prediction than for explanation, due to its greatly increased flexibility over the GLM. Although most of our interest in this book has been in explanation, there is one application where prediction is the most important, and that is in the estimation of a trend surface. In Exercise 9.3, you are asked to use a GAM to estimate the trend surface computed in Section 3.2.1 using linear regression and median polish. You will be impressed by the result.

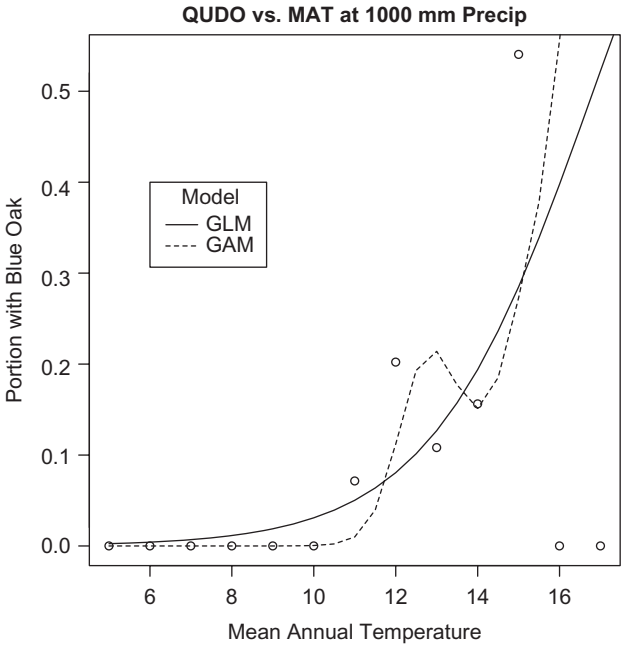

FIGURE 9.6
Cross section at *Precip* = 1000 of the GLM and GAM fits of *QUDO* to *Precip* and *MAT* data from the Sierra Nevada subset of the blue oak data from Data Set 2 (Figure 9.4 Dotplots of the example data in Section 9.2.1).

9.3 Classification and Regression Trees (a.k.a. Recursive Partitioning)

9.3.1 Introduction to the Method

The method described in this section was originally called classification and regression trees, or CART (Breiman et al., 1984), and this is still the better known term. However, the acronym CART was later trademarked and applied to a specific software product. At least in the R community one rarely sees the term "CART." Instead, it is called "recursive partitioning." This name is equally accurate, although unfortunately not as evocative, and we will therefore use it. Recursive partitioning and random forest, which is described in the next section, are two of a group of methods that fall in the category known both as *machine learning* and as *data mining*. These methods are fundamentally different from the other methods in this book in that they focus on an algorithmic approach rather than a data modeling approach to data analysis (Breiman, 2001b). Our application of these methods will not make use of spatial information at all. Instead, we will use our usual tactic, which is to carry out an analysis based solely on the attribute data and then examine the spatial structure of the result as one test of validity.

Recursive partitioning can be illustrated through a simple example. In Section 7.5, a number of factors were identified as potentially being associated with wheat yield in Field 4.1. Two of these were weed infestation level *Weeds*, rated on a scale of 1 through 5, and soil phosphorous content *SoilP*. We will use these not because they are the most important explanatory variables but because they provide a good example of how recursive

partitioning works. The response variable (*Yield*) in this example is a ratio scale (i.e., "quantitative") variable (Section 4.2.1). The tree constructed by recursive partitioning in this case is called a *regression tree*. It is also possible to construct trees, called *classification trees*, in which the response variable is nominal (Exercise 9.4).

The data from the 86 sample locations and the interpolated yield data are loaded as described in Appendix B.4. Figure 9.7 shows a bubble plot of the *SoilP–Weeds* data space in which the size of the bubble corresponds to the magnitude of *Yield*. The horizontal and vertical lines are the partitions. The idea of recursive partitioning is to recursively divide the sample space into two parts in such a way that each subdivision is as homogeneous as possible with respect to the response variable. In a regression tree, homogeneity is measured by the sum of the squares of the deviation from the mean over the partition. The data frame `data.Set4.1` is created using the code in Appendix B.4. We can apply the function `rpart()` from the `rpart` package (Therneau et al., 2011) to the data as follows.

```
> Set4.1.rp1 <- rpart(Yield ~ SoilP + Weeds, data = data.Set4.1,
+     method = "anova")
> plot(Set4.1.rp1) #Fig. 9.8
> text(Set4.1.rp1)
```

The statement in line 1 generates the object `Set4.1.rp`, and the remaining two lines plot the structure of the object and print the text associated with it. The argument `method = "anova"` ensures that a regression tree is constructed. If the argument is `method = "class"` then a classification tree is constructed. If this argument is omitted,

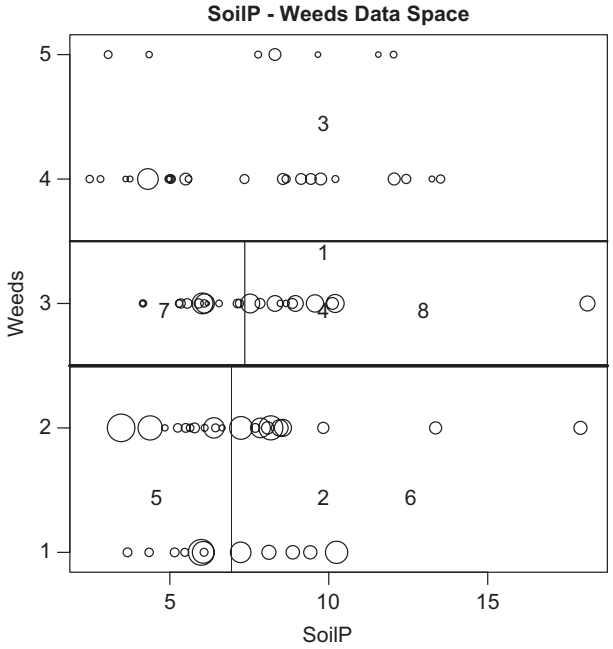

FIGURE 9.7
Plot of the data space of a regression tree for Field 4.1 in which the response variable is *Yield* and the explanatory variables are *SoilP* and *Weeds*. The figure shows the splits that partition the data space into the nodes of the regression tree. The numbers in the figure correspond to the numbers of the splits. The thicker lines correspond to the nodes higher on the tree.

`rpart()` will try to figure out the appropriate method (classification tree or regression tree) based on the form of the response variable (factor or numerical). The regression tree constructed for these explanatory variables and this response variable is shown in Figure 9.8. The terminal nodes of the regression tree are identified by the mean value of the response variable. The terminal nodes of a classification tree are identified by the value of the classification variable having the most data records in that node.

Using Figures 9.7 and 9.8 as a guide, we can see how recursive partitioning works. The first partition occurs at *Weeds* = 2.5. In tree diagrams generated by `rpart()`, the left branch always includes data that satisfies the condition listed in the diagram, and the right branch includes data that do not. The program tests every value of each of the explanatory variables (in this case, *SoilP* and *Weeds*) over the indicated subspace of the sample space to determine which value separates that subspace into the two parts whose sums of squares of deviations from the mean of the response variable is minimized. The subsets defined by *Weeds* ≥ 2.5 and *Weeds* < 2.5 are called *nodes*, and the conditions themselves are called *splits* (Breiman et al., 1984, p. 22). Nodes are numbered top to bottom and left to right (Figure 9.8). Thus, node 1 consists of the entire subspace above the thicker of the two horizontal lines, defined by *Weeds* = 2.5, in Figure 9.7, and node 2 is the subspace below that line. This illustrates that splits occur halfway between attribute values. The algorithm recursively searches each node for the split that defines the two most homogeneous subsets. From Figure 9.8 we see that node 3 is the subspace defined by *Weeds* ≥ 3.5 and node 4 is defined by 2.5 < *Weeds* < 3.5. Node 2, defined by *Weeds* < 2.5, is split into the two most homogeneous partitions at the value *SoilP* < 6.94, to form nodes 5 and 6. Finally, node 4 is split at the value *SoilP* < 7.36, forming nodes 7 and 8. The final subsets are called *terminal nodes*. The set of all nodes emanating from a particular *root node* is called a *branch* (Breiman et al., 1984, p. 64).

FIGURE 9.8
Regression tree of the recursive partitioning example of Figure 9.7. The numbers at the bottom are the mean *Yield* values for each terminal node.

9.3.2 The Mathematics of Recursive Partitioning

The `rpart()` function uses the methods described by Breiman et al. (1984), which we will now briefly discuss. It turns out to be easier to discuss them in the context of regression trees; those interested in the parallel discussion for classification trees are referred to the original source. Our discussion simplifies the actual algorithm a bit; for the full complexity, the reader is referred to Therneau and Atkinson (1997).

The algorithm that constructs a regression tree such as that of Figure 9.7 or a classification tree such as that of Exercise 9.4 requires the following three elements (Breiman et al., 1984, p. 229):

1. a way to select a split at every intermediate node
2. a rule for determining when a node is a terminal node
3. a rule for assigning a value of the response variable to every terminal node

Our terminology will be consistent with the rest of this book and different from that of Breiman et al. (1984). What they call a *case* we call a data record, that is, a pair (X, Y), where X is a vector of explanatory variables in a data space, which Breiman et al. (1984) call a *measurement space*, and Y is a response variable.

The third of the required three elements just given, the rule for assigning a response variable to a terminal node, is the easiest to specify, and has already been mentioned informally. Let L be the set of n data records $(X_1, Y_1), \ldots, (X_n, Y_n)$. Let t denote a terminal node in the regression tree T. In the case in which Y is a categorical variable (the classification tree case), Y is assigned the value of the response variable of the largest number of data records in the node (with some rule for ties). In the case in which Y is a "quantitative" variable (the regression tree case), the rule is defined as follows. For each terminal node t in T we assign the value

$$\bar{Y}(t) = \frac{1}{n(t)} \sum_{X_i \in t} Y_i \tag{9.13}$$

where $n(t)$ is the number of measurements in the subset t.

Having defined the third element of the algorithm, we move on to the first element, the splitting rule. Define the *resubstitution estimate of the relative error* for the tree T to be

$$R(T) = \frac{1}{n} \sum_{t \in \tilde{T}} \sum_{x_i \in t} (Y_i - \bar{Y}(t))^2. \tag{9.14}$$

This same error estimate is used in linear regression, where it is called the mean squared error or *MSE* (Appendix A.2). The notation $R(T)$ emphasizes the dependence on the particular regression tree T. Given this definition, Breiman et al. (1984, p. 231) define the splitting rule (the first element of the three given above) to be the selection of the split that most decreases $R(T)$.

The last remaining element to be determined is the hardest: number 2, the rule to determine when a node is a terminal node. The great advance of the algorithm developed by Breiman et al. (1984) is that, rather than testing at each node to decide whether to stop, recursive partitioning makes this determination by growing the tree beyond the optimal point to a maximal tree $T_{\max}$ and then pruning it back to the optimal. In this way, one avoids the problem of stopping too soon based on incomplete information.

After $T_{\max}$ has been reached by the splitting process, the determination of the optimal split is made by rewarding a tree for providing a good fit to the data and penalizing the tree for being complex. This concept is related to the *bias-variance tradeoff* that is described more fully in Section 8.2.1 in the regression context. For any subtree T of $T_{\max}$, Breiman et al. (1984, p. 66) define the *complexity* of T, denoted $|\tilde{T}|$, as the number of terminal nodes in T. Let $\alpha \geq 0$ be a real number called the complexity parameter. The *cost complexity measure* $R_\alpha(T)$ is defined as

$$R_\alpha(T) = R(T) + \alpha |\tilde{T}|. \tag{9.15}$$

The cost complexity measure is reduced by trees that reduce the mean square error (the bias), but it is increased by trees that do so by becoming too complex (increasing the variance). In order to find the tree that it declares optimal, the recursive partitioning algorithm begins with the maximal tree and constructs a sequence of trees with increasing cost complexity values, and therefore with decreasing complexity, moving toward the tree that minimizes $R_\alpha(T)$ as defined in Equation 9.15.

The value of $R_\alpha(T)$ is generally estimated by the process of *tenfold cross validation*. We have already seen cross validation once in this chapter, in Section 9.2. Unlike the case with GAM, cross validation is carried out explicitly in recursive partitioning as follows. Let the *sample set* L be defined as the set of all data records, that is, of pairs (X_i, Y_i). Divide the sample set into 10 subsets, each containing (as closely as possible) the same number of cases. Denote these subsets $L_1, L_2, .., L_{10}$. For each k, $k = 1, ..., 10$, the tenfold cross validation algorithm computes a regression tree $T^{(k)}$ by applying the algorithm to the set $L - L_k$. The cross-validation estimate $R^{CV}(T)$ is

$$R^{CV}(T) = \frac{1}{n} \sum_v \sum_{X_i \in L_v} (Y_i - T^{(k)}(X_i))^2, \tag{9.16}$$

where $T^{(k)}(X_i)$ is the recursive partitioning estimate of Y_i by the tree $T^{(k)}$.

We now have the three elements of the recursive partitioning algorithm in hand. In summary, at each node, the algorithm determines the next split to be the one that minimizes the estimated resubstitution error $R(T)$, given by Equation 9.14. The tree is grown to a maximal tree $T_{\max}$ and then pruned back to a size that optimizes the cost-complexity measure given in Equation 9.15, as estimated using the tenfold cross-validation estimate given in Equation 9.16. At that point, the value of the terminal node of the regression tree is the mean value over the node of the response variable. We can now apply the method to our data sets. We first consider Data Set 2.

9.3.3 Exploratory Analysis of Data Set 2 with Regression Trees

The use of the recursive partitioning algorithm to generate classification and regression trees also generates a great deal of auxiliary information that can be very useful in exploring a data set. The function `rpart()` employs two control parameters defined in Section 9.3.2. These are $n_{\min}$, the minimum number of cases allowed in a terminal node, and α, the cost complexity measure defined in Equation 9.15. They are represented in `rpart()` by the parameters `minsplit` and `cp`, which have default values of 20 and 0.05, respectively. The parameter `minsplit` is the smallest node size that will be considered for a split, and thus corresponds exactly to $n_{\min}$. The parameter `cp` (for complexity parameter) is the value of α scaled by $R(T_{root})$, the resubstitution error of the root node T_{root}

of the entire tree. That is, in comparison with Equation 9.15, the resubstitution estimate is given by (Therneau and Atkinson, 1997, p. 22)

$$R_{cp}(T) = R(T) + cp\,|\,\tilde{T}\,|\,R(T_{root}). \tag{9.17}$$

The parameters `minsplit` and `cp` govern the complexity of the tree, and therefore the first step in a recursive partitioning analysis is to determine whether the default values for these parameters are appropriate. In particular, it is important to determine whether the parameter `cp`, which controls the complexity of the optimal tree, actually does give us the tree having the most information. The objective is to achieve the correct balance between fit and complexity. Too simple a tree will lead to underfitting, which generates a suboptimal result by not using all of the available data. Too complex a tree leads to overfitting, which uses too much data specific only to this particular data set.

We can obtain information about the sequence of trees generated by `rpart()` in the pruning process by first setting `minsplit` and `cp` to very small values to generate the maximal tree and the using the functions `plotcp()` and `printcp()` to view the properties of the sequence leading to this maximal tree. In our analysis of Data Set 2, we will continue to analyze the Sierra Nevada and Coast Range data separately, focusing on the Sierra Nevada, entered as an `sf` object, in the text and the Coast Range in the exercises. The tree will only be computed for endogenous explanatory variables, and the parent material classes are aggregated.

```
> data.Set2Srp <- with(data.Set2S.sf, data.frame(MAT,
+    Precip, JuMin, JuMax, JuMean, JaMin, JaMax, JaMean, TempR, GS32,
+    GS28, PE, ET, Texture, AWCAvg, Permeab, SolRad6, SolRad12,
+    SolRad, QUDO))
> data.Set2Srp$PM100 <- as.numeric(data.Set2Sierra.sf$PM100 > 0)
> data.Set2Srp$PM200 <- as.numeric(data.Set2Sierra.sf$PM200 > 0)
> data.Set2Srp$PM300 <- as.numeric(data.Set2Sierra.sf$PM300 > 0)
> data.Set2Srp$PM400 <- as.numeric(data.Set2Sierra.sf$PM400 > 0)
> data.Set2Srp$PM500 <- as.numeric(data.Set2Sierra.sf$PM500 > 0)
> data.Set2Srp$PM600 <- as.numeric(data.Set2Sierra.sf$PM600 > 0)
```

We will initially leave `minsplit` at its default value of 20, but reduce the size of `cp` from the default value of 0.05.

```
> library(rpart)
> cont.parms <- rpart.control(minsplit = 20,cp = 0.002)
> Set2S.rp <- rpart(QUDO ~ ., data = data.Set2Srp,
+   control = cont.parms, method = "anova")
```

Now we use `plotcp()` and `printcp()` to determine the appropriate value of `cp`.

```
> plotcp(Set2S.rp) # Fig. 9.9
> printcp(Set2.rp)
Regression tree:
rpart(formula = "QUDO ~ .", data = data.Set2Srp, method = "anova",
    control = cont.parms)
Variables actually used in tree construction:
 [1] AWCAvg   ET       GS28     GS32     JaMax    JaMean   JuMax
```

```
 [8] JuMin     MAT       PE        Permeab   PM300     Precip    SolRad
[15] SolRad12 SolRad6   TempR

Root node error: 422.22/1768 = 0.23881

n= 1768

          CP nsplit rel  error xerror     xstd
1  0.4516262      0   1.00000 1.00038 0.010300
2  0.0613849      1   0.54837 0.58686 0.025282
   *    *    *   DELETED   *    *    *
7  0.0102024      6   0.39782 0.45256 0.022612
8  0.0082386      8   0.37741 0.43808 0.022814
9  0.0077734      9   0.36918 0.43222 0.022589
10 0.0069921     10   0.36140 0.43489 0.022877
11 0.0066791     11   0.35441 0.43432 0.022967
12 0.0062327     12   0.34773 0.43606 0.023056
13 0.0059766     13   0.34150 0.43707 0.023163
14 0.0055890     14   0.33552 0.43782 0.023411
15 0.0055266     15   0.32993 0.43473 0.023438
   *    *    *   DELETED   *    *    *
31 0.0020000     36 0.25092 0.45350 0.024702
```

The function `printcp()` gives a table of values of the optimal tree if the corresponding value of `cp` is used. The table includes the value of `cp` at which a split occurs; `nsplit`, the number of splits; `rel error`, the resubstitution estimate $R(T)$ given in Equation 9.14 of the relative error; `xerror`, the cross validation estimate $R^{CV}(T)$ given in Equation 9.16 of the error; and `xstd`, the corresponding cross-validation estimate of the standard error. In the example, the first split occurs if a value of `cp` less than 0.4516262 is specified. One minus the relative error can be used to interpret the "portion of variance explained" by the regression tree, analogous to the R^2 of linear regression. Thus, prior to making any splits (`nsplit` = 0) the relative error is 1, and the regression tree, which consists of a single node, explains none of the variance.

In multiple linear regression, as the number of explanatory variables increases, the coefficient of determination R^2 cannot decrease (Kutner et al., 2005, p. 226). Analogously, as the complexity of the tree increases, the resubstitution estimate $R(T)$ cannot decrease. For this reason, the cross-validation estimate $R^{CV}(T)$ (`xerror` in the table) is considered a better estimate of the "portion of the variance explained by the tree" (Breiman et al., 1984, p. 225). The derivation of the standard error `xstd` estimate is complex (Breiman et al., 1984, p. 304). The greatest usefulness of this statistic is in interpreting the output of the function `plotcp()`.

The function `plotcp()` plots the relative error $R^{CV}(T)$, together with standard error bars, that are obtained in a tree grown using the value of `cp` on given the abscissa. The values of `cp` on the abscissa are the geometric means of successive values of `cp` printed by `printcp()`. Thus, for example in the listing above, the first split occurs at `cp` = 0.452 (to three decimal places), and the second at cp = 0.061. The first cross-validation computation (which is actually the second point from the left in Figure 9.9) is therefore carried out at `cp` = $0.166 = \sqrt{0.452 \times 0.061}$, and successive cross validations are carried out at successive geometric means.

The lowest value of $R^{CV}(T)$ occurs at cp = 0.0055890, at which value the tree has 14 splits. The error bars in the output of `plotcp()` (Figure 9.9) represent one standard error, and the

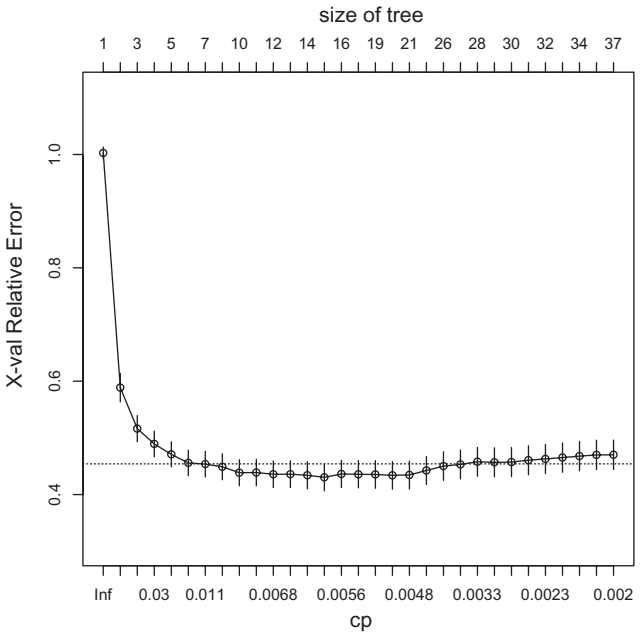

FIGURE 9.9
Output of the function plotcp() for a regression tree grown for Data Set 2 with *QUDO* as the response, showing the relative error for decreasing values of cp.

horizontal dotted line in the figure is drawn at one standard error above the lowest value of the cross validation estimate of the relative error. Therneau and Atkinson (1997, p. 13) recommend using the "1 *SE*" rule to choose the value of `cp` governing the pruning of the tree. This rule states that a good choice of `cp` for pruning is often the leftmost value of `cp` for which the error bar lies entirely below the dotted line. In our case, this is when the "size of the tree" `nsplit` has the value 9, at which point, from the output of `printcp()`, cp = 0.0078 to four decimal places.

We can make the regression trees somewhat more attractive (Figure 9.10a) with some arguments to the functions `plot()` and `text()`.

```
> cont.parms <- rpart.control(minsplit = 20, cp = 0.0078)
> Set2S.rp <- rpart(QUDO ~ ., data = data.Set2Srp,
+   control = cont.parms, method = "anova")
> plot(Set2S.rp,branch = 0.4,uniform = T,margin = 0.1,
+   main = "Data Set 2 Sierra Nevada Regression Tree",
+   cex.main = 2) # Fig. 9.10a
> text(Set2S.rp,use.n = T,all = T, cex = 0.65)
```

The argument `uniform = T` makes the vertical distance between nodes uniform. The argument `branch = 0.4` tilts the branches from vertical. The `argument margin = 0.1` creates a small margin around the tree so the bottom values do not get lost. In the `text()` function, the argument `use.n = T` causes the value of the number of cases in each node to be printed and the argument `all=T` puts the values of the response variable and number of cases in all nodes, not just the terminal nodes. As usual, the argument `cex` controls the size of the text. This value is a compromise between legibility and forcing the text into the tree.

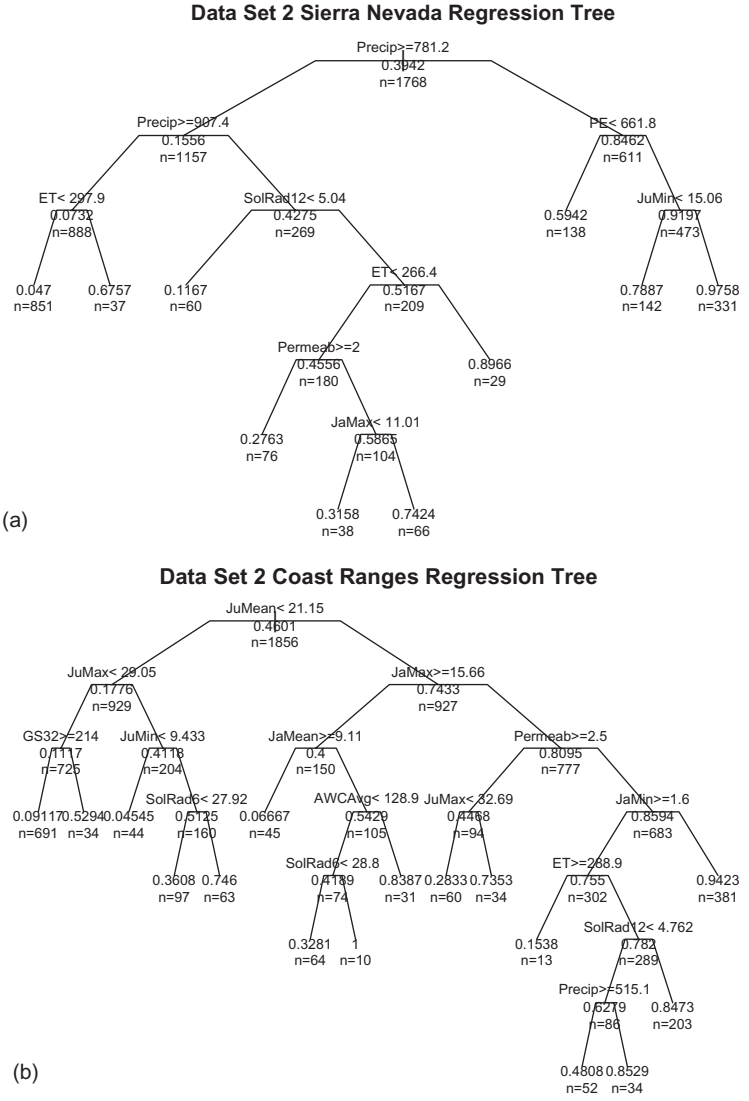

FIGURE 9.10
(a) Regression tree with QUDO as the response variable for the Sierra Nevada subset of Data Set 2. (b) Same as (a) for the Coast Range subset of Data Set 2.

We can get more information with a call to the function `summary()`, a portion of whose result is shown here.

```
> summary(Set2S.rp)
Call:
rpart(formula = "QUDO ~ .", data = data.Set2Srp, method = "anova",
    control = cont.parms)
  n= 1768
Node number 1: 1768 observations,    complexity param=0.4516262
  mean=0.3942308, MSE=0.2388129
```

```
  left son=2 (1157 obs) right son=3 (611 obs)
  Primary splits:
     Precip < 781.1516  to the right, improve=0.4516262, (0 missing)
     JuMean < 24.17917  to the left,  improve=0.4052687, (0 missing)
     JuMax  < 33.70834  to the left,  improve=0.3875277, (0 missing)
     MAT    < 14.85555  to the left,  improve=0.3826863, (0 missing)
     JaMax  < 11.03334  to the left,  improve=0.3647264, (0 missing)
  Surrogate splits:
     JuMax  < 34.90555  to the left,  agree=0.876, adj=0.642, (0 split)
     JaMax  < 11.95     to the left,  agree=0.859, adj=0.592, (0 split)
     JuMean < 24.51945  to the left,  agree=0.853, adj=0.574, (0 split)
     MAT    < 15.325    to the left,  agree=0.837, adj=0.527, (0 split)
     JaMean < 6.62083   to the left,  agree=0.836, adj=0.525, (0 split)

Node number 2: 1157 observations,    complexity param=0.06138488
  mean=0.1555748, MSE=0.1313713
  left son=4 (888 obs) right son=5 (269 obs)
  Primary splits:
     Precip < 907.4277  to the right, improve=0.1705170, (0 missing)
     JuMax  < 33.125    to the left,  improve=0.1572847, (0 missing)
     ET     < 297.942   to the left,  improve=0.1528268, (0 missing)
     JaMax  < 11.01389  to the left,  improve=0.1465055, (0 missing)
     JuMean < 23.42223  to the left,  improve=0.1360703, (0 missing)
  Surrogate splits:
     JuMax  < 34.09166  to the left,  agree=0.834, adj=0.286, (0 split)
     JaMax  < 11.625    to the left,  agree=0.831, adj=0.275, (0 split)
     JuMean < 23.56111  to the left,  agree=0.814, adj=0.201, (0 split)
     MAT    < 14.76666  to the left,  agree=0.812, adj=0.190, (0 split)
     PE     < 719.836   to the left,  agree=0.808, adj=0.175, (0 split)

Node number 3: 611 observations,   complexity param=0.02680101
  mean=0.8461538, MSE=0.1301775
  left son=6 (138 obs) right son=7 (473 obs)
  Primary splits:
     PE     < 661.797   to the left, improve=0.1422702, (0 missing)
     JuMean < 24.82639  to the left, improve=0.1357518, (0 missing)
     GS28   < 267.35    to the left, improve=0.1325340, (0 missing)
     JuMin  < 15.05556  to the left, improve=0.1260423, (0 missing)
     MAT    < 15.82222  to the left, improve=0.1239181, (0 missing)
  Surrogate splits:
     JaMin  < 0.76388   to the left, agree=0.895, adj=0.536, (0 split)
     GS28   < 259.1     to the left, agree=0.887, adj=0.500, (0 split)
     JuMin  < 14.47778  to the left, agree=0.881, adj=0.471, (0 split)
     GS32   < 198.4     to the left, agree=0.866, adj=0.406, (0 split)
     JuMean < 24.81111  to the left, agree=0.856, adj=0.362, (0 split)
```

The most important new information is a list of "primary splits" and "surrogate splits." The list of primary splits is easy to explain. Recall that the optimal split is chosen to maximally reduce the estimated resubstitution error $R(T)$. The split at the top of the list of primary

splits is the one that satisfies this criterion, and the remainder are those that reduce $R(T)$, but not as much. These other splits are called the *competitors*. Thus, for example the first competitor for `Precip` in the first node is `JuMean`. The *surrogate split* is the split that best predicts the optimal split s^* in the sense that it sends cases to the left and right in closest proportions to that of s^* (Breiman et al., 1984, p. 141). Surrogate splits are useful in identifying variables that may be *masked*. This phenomenon is similar to multicollinearity in multiple linear regression and has the same negative consequences. A variable X_k may have almost as much explanatory power at a given split as the variable that produces the optimal split s^*, and may over the entire tree have more explanatory power, but never appear in the tree because locally at each split it has less explanatory power than some other variable. We can informally determine whether a variable is being masked by examining the surrogates at each split and seeing if any variables either appear reasonable from an ecological perspective or appear repeatedly in several splits.

For the Sierra Nevada data in the tree of Figure 9.10a, the output indicates that the main surrogates and competitors of `Precip` in the first split are `JuMean`, `JuMax`, `MAT`, and `JaMax`. This is not surprising, given the high degree of correlation among these variables. Interestingly, the variables that determine the splits are almost exclusively climatic, with `Permeab` entering in one split very far down the tree. Since QUDO is a binary variable ($QUDO = 1$ if blue oak is present, and $QUDO = 0$ if blue oak is absent), the tree could be generated either as a classification tree or a regression tree. The argument `method = "anova"` in the statement calling the function `rpart()` specifies that a regression tree will be generated. The regression tree is easier to interpret because it indicates the fraction of the data records in each node for which $QUDO = 1$.

The interpretation of the tree in terms of the response variable *QUDO* is given by the numbers just below the rule defining the split. This indicates the mean value of `QUDO` over the node. Thus, for example, in Figure 9.10a the mean value of *QUDO* at the root node is 0.3942, indicating that 39.42% of the data records in the root node (the whole Sierra Nevada data set) have blue oaks present. In node 2, which contains those data records in which `Precip` >= 781.2, about 15% of the records have blue oak present, whereas in node 3, where `Precip` < 781.2, about 84% of the data records have blue oak present. In node 3, the splitting variable is `PE` (potential evapotranspiration); 92% of the sites where `Precip` < 781.2 and `PE` ≥ 15.06 mm contain a blue oak. Evidently precipitation is a very important factor in the Sierra Nevada. Considering that blue oaks are known for their drought tolerance, it is not surprising that a higher fraction of them are found in data locations with low mean annual precipitation and high potential evaporation.

The situation is different in the Coast Ranges (Figure 9.10b, generated in Exercise 9.5). Here, `Precip` only enters far down the tree, and temperature-related variables seem most important. In the Coast Ranges, precipitation is generally lower (almost all of the records are below 1100 mm), and the data records with low precipitation values span a broader range of elevations. It is possible that the regression tree does not identify `Precip` as an important splitting variable because virtually all of the sites are located in areas where precipitation is sufficiently low. Recursive partitioning identifies `JuMean` as the first splitting variable (Figure 9.10b). According to the figure, 75% of the sites where `JuMean` ≥ 21.5 have blue oaks present, while only 18% of the sites where `JuMean` < 21.5 have blue oaks present. Both the competitors and the surrogate splits of `JuMean` are temperature related.

In summary, recursive partitioning has identified precipitation, potential evaporation, and temperature as the most important endogenous variables in the Sierra Nevada, with low precipitation and high potential evaporation especially favoring blue oak presence. In the Coast Range, where precipitation is generally lower, high mean June temperatures

and low January maximum temperatures favor blue oak presence. The extremely high correlation values of all of the temperature-related variables, however, render this conclusion highly tentative at best.

9.3.4 Exploratory Analysis of Data Set 3 with Recursive Partitioning

The preliminary analysis of Data Set 3 in Chapter 7 indicated that one of the three regions of rice production, the central region, consistently lagged behind the other two regions in terms of yield. There was, however, considerable variability among the farmers in the northern and central region, and only one farmer was surveyed in the southern region. Climate does not appear to have had an important influence on year-to-year yield variability in the three seasons in which data were collected, and terrain also does not appear to have much influence. The primary characteristic distinguishing soils in the central region was high silt content relative to soils in the northern and southern region. Yields appeared to be more determined by the farmer than by location or year in those cases where different farmers farmed the same field in different years. The farmers in the north applied more fertilizer and appear to have irrigated and managed weeds more effectively (Table 7.4).

The contents of the file *Set3data.csv* are loaded into the data frame `data.Set3`. The data are a mixture of exogenous and management variables. No endogenous plant-based variables were recorded in Data Set 3. Recall that the goal of our analysis is to separate out the management practices from the field conditions in determining the factors associated with high yields. As a step in this direction, we develop a recursive partitioning analysis using only the exogenous variables (Figure 9.11). The procedure is identical to that used in the previous section. *Silt* is the first splitting variable, followed by *pH* in the high silt fields, and then by *Corg* and *Clay*. All of the northern and southern fields have silt content below 44%, and almost all of the central fields have soil silt content above this level. Thus, it may be that the reason silt is identified as the first splitting variable is simply because it separates

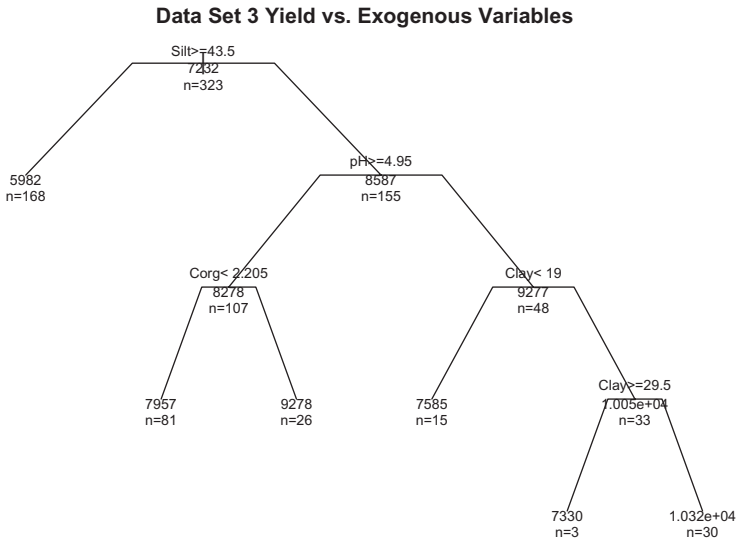

FIGURE 9.11
Regression tree for Yield as a function of the exogenous variables of Data Set 3.

the central region from the other two. What about pH? Soils in the northern region tend to be more acidic than those in the central and southern region. From an agronomic perspective, however, flooding tends to move all soil pH levels towards neutrality (Williams, 2010, p. 10), so that rice in general is less sensitive to soil pH that are terrestrial crops.

Is either silt content or pH a determining factor in yield in these fields? We can address this by first sorting the farmers by mean *Yield* and then by mean *Silt* and mean *pH*.

```
> print(sort(with(data.Set3, tapply(Yield, Farmer, mean))), digits = 4)
    I     E     K     G     H     F     J     L     B     A     D     C
 4982  5112  5245  5559  6011  6331  6958  7948  8219  8738  9542 11456
```

Remember that Farmers *A* through *D* are in the north, Farmer *L* is in the south, and farmers *E* through *K* are in the central region.

```
> print(sort(with(data.Set3, tapply(Silt, Farmer, mean))), digits = 3)
   B    A    D    C    L    E    F    G    K    H    I    J
29.7 33.3 34.0 36.3 37.9 58.6 59.7 59.8 60.0 61.7 63.2 63.9
```

While it is true that the southern farmers have higher silt content than the northern farmers, the farmer with the highest mean silt content is *J*, the highest yielding central farmer, and the northern farmer with the highest mean silt content is *C*, the highest yielding northern farmer.

```
> print(sort(with(data.Set3, tapply(pH, Farmer, mean))), digits = 3)
   C    D    B    A    E    G    F    H    K    L    I    J
4.68 4.73 5.14 5.60 5.73 5.77 5.79 5.88 5.89 5.89 5.94 6.02
```

Farmer *J* also has the most basic soil, while Farmer *C* has the most acidic. Evidently, if there is a relation between *Silt*, *pH*, and *Yield*, it is a complex one. In Exercise 9.6, you are asked to plot these means. These plots are interesting as an example of the sort of "barbell"-shaped plots that can produce regression results that are difficult to interpret. Another complicating factor is the issue of spatial scale. Geographically, Data Set 3 has a three level spatial hierarchy: the sample point scale, the field scale, and the region scale. We have seen so far that within each region, the yields at the field scale do not depend on silt content or soil acidity, but between regions the field scale yields do depend on these quantities. We need to try to determine whether this relationship is truly a direct influence or simply an apparent association.

As in Section 9.3.3, we first do a preliminary run with a small value of `cp` and then use `plotcp()` and `printcp()` to select the appropriate value. Figure 9.12 shows the resulting regression tree that includes all of the explanatory variables, both exogenous and management. Exogenous variables play a relatively minor role. The first split is on *N*, the amount of applied nitrogen fertilizer. This split also almost perfectly divides the northern and southern regions from the center: almost all of the former report a fertilizer application rate greater than 60 kg/ha (indeed, all of the northern farmers report the same rate, 61 kg/ha), while all of the central farmers apply at a lower rate (Exercise 9.7). The next splits on each branch are on the variable *Farmer*. We can see this from the output of the function `summary()`.

```
> cont.parms <- rpart.control(minsplit = 20,cp = 0.003)
> Set3.rp2 <- rpart(Yield ~ DPL + Cont + Irrig +
+     N + P + K + Variety + pH + Corg + SoilP + SoilK + Sand +
```

```
+     Silt + Clay + Farmer, data = data.Set3rp, control = cont.parms)
> summary(Set3.rp2)
   *   *   *   DELETED   *   *   *
Node number 1: 323 observations,    complexity param=0.565231
  mean=7232.111, MSE=3271954
  left son=2 (193 obs) right son=3 (130 obs)
  Primary splits:
      N      < 61.3  to the left,      improve=0.5652310, (0 missing)
      Farmer splits as  RRRRLLLLLLLR,  improve=0.5236007, (0 missing)
      Silt   < 43.5  to the right,     improve=0.5176631, (0 missing)
      K      < 11.7  to the left,      improve=0.4108487, (0 missing)
      Cont   < 4.5   to the left,      improve=0.3882669, (0 missing)
  Surrogate    splits:
      K      < 11.7  to the left,      agree=0.923, adj=0.808, (0 split)
      Farmer splits as  RRRRLLLLLLLL,  agree=0.923, adj=0.808, (0 split)
      Silt   < 43.5  to the right,     agree=0.916, adj=0.792, (0 split)
      P      < 67.2  to the left,      agree=0.910, adj=0.777, (0 split)
      Cont   < 4.5   to the left,      agree=0.885, adj=0.715, (0 split)

Node number 2: 193 observations,      complexity param=0.1026634
 mean=6115.995, MSE=1187620
  left son=4 (103 obs) right son=5 (90 obs)
  Primary splits:
      Farmer splits as  ----LRLLLRLR,  improve=0.4733587, (0 missing)
      N      < 44.7  to the left,      improve=0.4578913, (0 missing)
      P      < 67.2  to the right,     improve=0.4578913, (0 missing)
      Irrig  < 2.5   to the left,      improve=0.3517589, (0 missing)
      Cont   < 3.5   to the left,      improve=0.2455773, (0 missing)
  Surrogate  splits:
      N      < 53.5  to the left,  agree=0.720, adj=0.400, (0 split)
      Corg   < 1.52  to the left,  agree=0.674, adj=0.300, (0 split)
      Silt   < 39.5  to the right, agree=0.663, adj=0.278, (0 split)
      Sand   < 30.5  to the left,  agree=0.642, adj=0.233, (0 split)
      Irrig  < 3.5   to the left,  agree=0.637, adj=0.222, (0 split)

Node number 3: 130 observations,   complexity param=0.06751958
  mean=8889.115, MSE=1771316
  left son=6 (120 obs) right son=7 (10 obs)
  Primary splits:
      Farmer  splits as  LLRL-------L,  improve=0.3098846, (0 missing)
      Clay    < 16.5  to the left,      improve=0.2583827, (0 missing)
      Sand    < 50.5  to the right,     improve=0.2272371, (0 missing)
      Irrig   < 4.5   to the left,      improve=0.1897808, (0 missing)
      Variety splits as  RLRR,           improve=0.1585697, (0 missing)
  Surrogate splits:
      SoilP < 1.45 to the right, agree=0.938, adj=0.2, (0 split)
```

Not all farmers are identified in the tree as being in one or the other node, but all are in the tree. For example, in node 3 Farmer *C* is in the right split along with the other

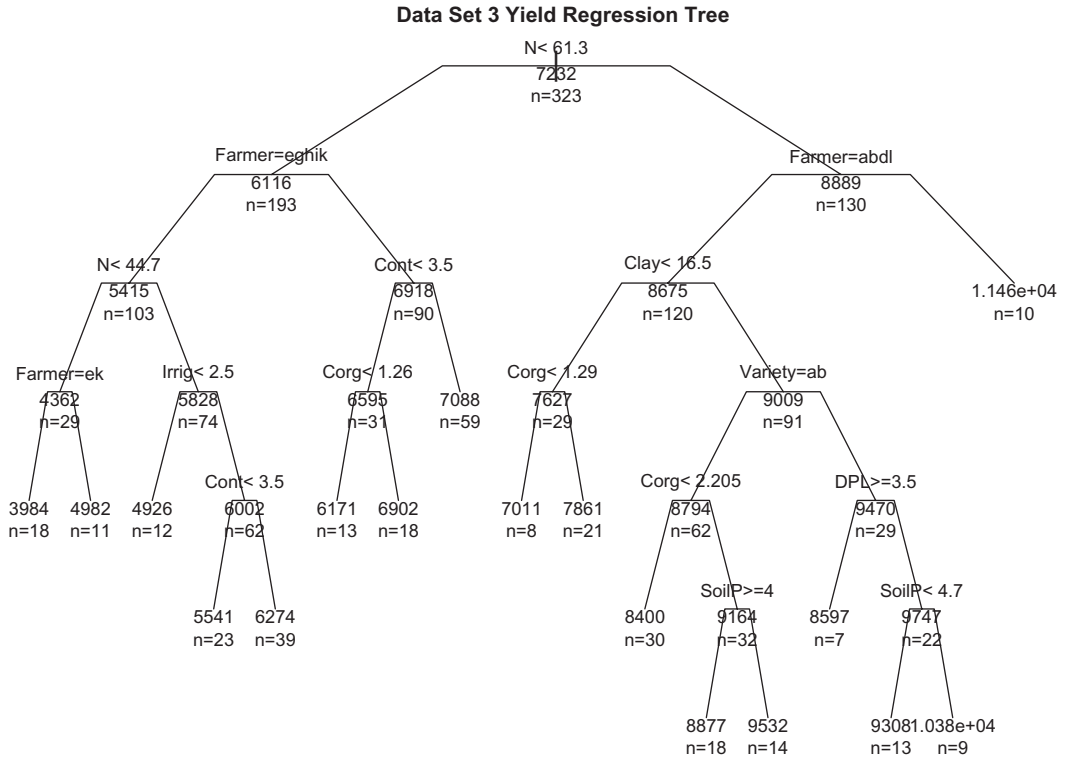

FIGURE 9.12
Regression tree for Yield as a function of both the exogenous and management variables of Data Set 3.

northern and southern farmers. He does not show up in the node label because it identifies the values of splitting variable (*Farmer*) that split to the left, and he splits to the right.

The surrogate and primary splits of the first node indicate that the most important initial splitting factor is the region, with northern separated from central, and southern somewhere in between. Within regions, the most important splitting factor is *Farmer*. We can take advantage of the fact that recursive partitioning works equally well with nominal scale variables as with ratio or interval scale variables to use *Farmer* as a response variable and see what characterizes each individual farmer. Figure 9.13 is a classification tree of exogenous variables with *Farmer* as the response variable. Referring to Figure 7.15, Farmer *C* has the highest median yield among the northern farmers, and *B* has the lowest, while in the central region Farmer *J* has the highest median yield and *I* has the lowest. Farmer *B* does have some sites with very low clay content, but other than that the values of the exogenous variables of Farmers *B* and *C* seem very similar. Farmer *J* farmed fields with a wide variety of properties, some of which are very similar to those of the field farmed by *I*. Let us explore the variables that characterize the management practices and the field environment of each farmer. We have already seen evidence that variety has no substantial effect on yield. In Exercise 9.8, you are asked to examine the regression tree for all management variables including *Variety* and provide further evidence for this lack of effect. Therefore, we remove it from the list of explanatory variables and construct the classification tree (Figure 9.14). The first split, which separates three of

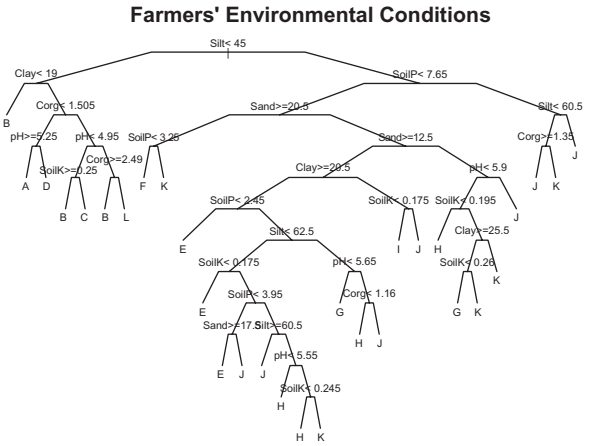

FIGURE 9.13
Classification tree for Data Set 3 with Farmer as the response variable and the exogenous variables as explanatory variables.

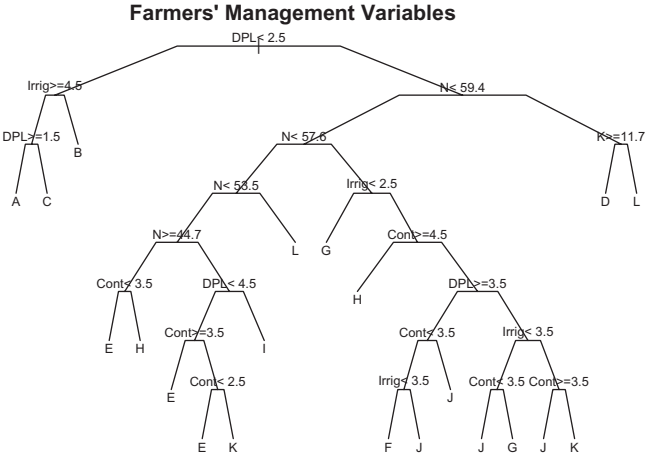

FIGURE 9.14
Classification tree for Data set 3 with Farmer as the response variable and the management variables as explanatory variables.

the northern farmers, is on *DPL*. In the left split, Farmer *C* has the most effective irrigation and the earliest planting date.

Among the farmers with $DPL > 2.5$, the next split, on $N < 59.4$, separates the remaining two non-central farmers. Working our way along the tree, the higher yielding central farmers, *J* and *F*, generally are distinguished by higher values of *Cont* and *Irrig*, while the lower yielding farmers, *E*, *I*, and *K*, are distinguished by lower values of *Cont* and *Irrig*, and, in the case of the lowest yielding farmer, *I*, by a high value of *DPL*. A call to the function `summary()` (not shown) indicates that the competitors and surrogates at the first split are *N*, *P*, *K*, *Irrig*, and *Cont*.

It appears that the explanatory variables *N*, *P*, *K*, *Cont*, *Irrig*, and *DPL*, play an important role in the trees determining the farmers. In particular, the primary distinguishing characteristics of the northern and southern farmers are an earlier planting date and higher

N fertilization rate. The farmers with better yields tend to irrigate more effectively, plant earlier, and have better weed control. We will conclude the recursive partitioning analysis of Data Set 3 by creating a table that shows the average of these variables as well as *Yield* for each field/farmer/season combination. First, we use the R function `ave()` to average each variable over all combinations of *Farmer, Season, RiceYear,* and *Field*. Here are the statements for *Yield* and *N*; the statements for *P, K, Cont, Irrig*, and *DPL* are analogous.

```
> Yield <- with(data.Set3, ave(Yield, Farmer, Season, RiceYear, Field))
> N <- with(data.Set3, ave(N, Farmer, Season, RiceYear, Field))
```

Next, we create a data frame with all of these averages except *Farmer,* which we have to treat separately because it is a factor.

```
> df.C <- data.frame(N, P, K, Irrig, Cont, DPL, Var, Yield)
```

This data frame contains many duplicate records because it has an identical record for each of the combinations of *Name, Season, RiceYear,* and *Field*. We can eliminate duplicate records from the data frame using the function `remove.dup.rows()` from the `cwhmisc` package (Hoffmann, 2015).

```
> library(cwhmisc)
> Table.C <- remove.dup.rows(df.C)
```

The function `remove.dup.rows()` doesn't work too well with non-numerical data such as *Farmer*. To include this variable in the table, first we paste together the names of all of the combinations of *Name, Season, Rice,* and *Field* and then eliminate duplicates using the function `unique()`.

```
> Comb <- with(data.Set3, paste(as.character(Farmer),
+   as.character(Season), as.character(RiceYear), as.character(Field)))
> Table.C2 <- data.frame(FrmRySnFld = unique(Comb), Table.C)
```

Finally, we display the table, ordered in terms of yield from highest to lowest.

```
> print(Table.C2[order(Table.C2$Yield, decreasing = TRUE),],
+   digits = 3, right = TRUE)
    FrmRySnFld    N    P    K Irrig Cont  DPL  Var Yield
283    C 3 1 2 62.2 72.0 23.4  5.00 5.00 1.00 2.00 11456
323    D 3 1 4 62.2 72.0 23.4  4.60 4.85 3.35 1.00  9542
200   L 2 1 15 62.8 72.8  0.0  4.12 4.64 3.00 2.00  8832
150    A 2 1 1 62.2 72.0 23.4  4.76 4.76 2.00 2.36  8738
175    B 2 1 3 62.2 72.0 23.4  3.64 4.68 2.00 4.00  8473
248    B 3 2 3 62.2 72.0 23.4  3.68 4.64 1.00 4.00  7965
54     J 1 1 5 58.4 55.2  0.0  3.45 3.45 3.00 2.00  7404
52    J 1 1 14 58.4 55.2  0.0  3.83 3.83 3.00 2.00  7153
273   L 3 2 16 58.8 55.5  0.0  3.48 3.56 3.44 1.00  7064
91    J 1 2 12 58.4 55.2  0.0  3.69 3.69 4.00 2.00  6898
221   J 2 2 14 58.4 55.2  0.0  3.83 3.83 3.00 2.00  6742
78    J 1 2 13 58.4 55.2  0.0  4.00 3.14 4.00 2.00  6408
```

```
31     F 1 1 7 58.4 55.2  0.0  3.50 3.60 4.00 2.00  6331
303   H 3 1 12 50.2 46.0  0.0  3.15 3.77 3.92 3.85  6023
113   H 1 1 10 58.4 55.2  0.0  3.80 3.80 3.50 2.00  5995
21     K 1 1 6 58.4 55.2  0.0  3.81 3.67 3.00 2.00  5904
290   E 3 1 13 50.2 46.0  0.0  3.00 2.57 3.29 2.57  5698
71     G 1 1 8 58.4 55.2  0.0  2.53 3.59 3.00 2.00  5559
223    I 2 2 5 39.2 72.0  0.0  2.45 2.64 4.45 1.00  4982
125   E 1 1 11 48.8 63.6  0.0  3.42 3.33 3.50 2.00  4771
103    K 1 1 9 39.2 72.0  0.0  2.67 3.00 4.00 1.00  4091
```

None of the central (or southern) farmers apply any potassium fertilizer. Nitrogen fertilizer shows a definite downward trend as one moves down the table, as does irrigation effectiveness. The northern farmers plant earlier in general, and the latest planting northern farmer gets the lowest yields. The northern farmers have uniformly good weed control. Again, variety does not seem to matter too much (note, however, that we are treating a nominal scale quantity as if it were ratio scale). In Exercise 9.9, you are asked to construct an analogous table using the exogenous variables. The only corresponding pattern that distinguishes the central fields from the others is soil texture.

In summary, we don't yet have any evidence that climatic or environmental factors play as important a role in determining yield as do the farmers' management practices. The data can be analyzed at three levels of spatial hierarchy (the region, the field, and the individual sample point), and the relationships between yield and explanatory variables in this data set appear to change at different levels of this hierarchy. We will explore these differences further in subsequent chapters.

9.3.5 Exploratory Analysis of Field 4.1 with Recursive Partitioning

The recursive partitioning analysis of Field 4.1 proceeds along the same lines as those already carried out in Sections 9.3.2 and 9.3.3. We will include only exogenous variables in the model. Here is the code to specify the `rpart()` model. In `rpart()`, when we specify the first argument separately, we use a character string. This differs from the `lm()` family, where it is a `formula` object.

```
> Set4.1.model <- "Yield ~ Clay + Silt + Sand + SoilpH + SoilTOC +
+    SoilTN +SoilP + SoilK + Weeds + Disease"
```

We first use the functions `printcp()` and `plotcp()` to select values of `cp` and `minsplit`, and then apply these values to the recursive partitioning function `rpart()`. The resulting regression tree is shown in Figure 9.15. The use of the function `summary()` results in the following list of competitors and surrogates (only a part of the output is shown).

```
> cont.parms <- rpart.control(minsplit = 5, cp = 0.02)
> Set4.1.rp <- rpart(Set4.1.model, data = data.Set4.1,
+   control = cont.parms, method = "anova")
> summary(Set4.1.rp)
   *   *   *   DELETED   *   *   *
Node number 1: 86 observations,    complexity param=0.6553524
  mean=2879.28, MSE=1984255
  left son=2 (62 obs) right son=3 (24 obs)
```

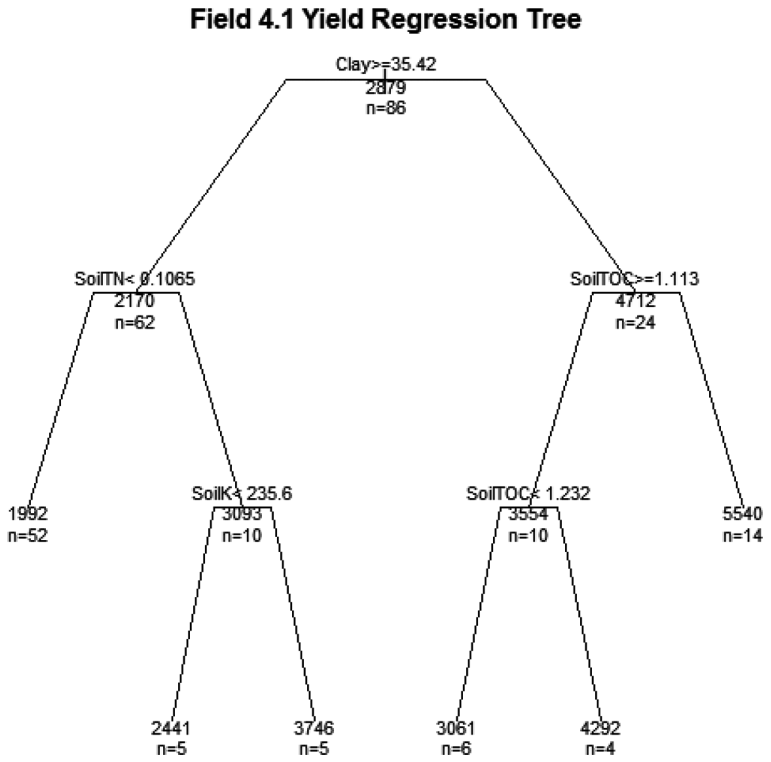

FIGURE 9.15
Regression tree for Yield in Field 4.1 with agronomic explanatory variables.

```
  Primary splits:
      Clay    < 35.425  to the right, improve=0.6553524, (0 missing)
      Sand    < 31.245  to the left,  improve=0.5887001, (0 missing)
      SoilK   < 156.85  to the right, improve=0.5454879, (0 missing)
      SoilTOC < 0.8955  to the right, improve=0.3534404, (0 missing)
      SoilTN  < 0.0785  to the right, improve=0.2345896, (0 missing)
  Surrogate splits:
      Sand    < 27.15  to the left,  agree=0.953, adj=0.833, (0 split)
      SoilK   < 156.85 to the right, agree=0.872, adj=0.542, (0 split)
      SoilTOC < 0.8955 to the right, agree=0.802, adj=0.292, (0 split)
      SoilTN  < 0.062  to the right, agree=0.767, adj=0.167, (0 split)
      Silt    < 41.57  to the left,  agree=0.744, adj=0.083, (0 split)

Node number 2: 62 observations,    complexity param=0.0595666
  mean=2169.791, MSE=421854.4
  left son=4 (52 obs) right son=5 (10 obs)
  Primary splits:
      SoilTN   < 0.1065 to the left, improve=0.3886372, (0 missing)
      SoilTOC  < 1.375  to the left, improve=0.3193074, (0 missing)
      SoilK    < 252.85 to the left, improve=0.2276935, (0 missing)
      SoilP    < 7.81   to the left, improve=0.2078210, (0 missing)
      Disease  < 2.5    to the left, improve=0.1187364, (0 missing)
```

```
  Surrogate splits:
       SoilTOC < 1.343   to the left, agree=0.968, adj=0.8, (0 split)
       SoilP   < 15.72   to the left, agree=0.871, adj=0.2, (0 split)
       SoilK   < 230.9   to the left, agree=0.871, adj=0.2, (0 split)

Node number 3: 24 observations,    complexity param=0.134846
  mean=4712.128, MSE=1360739
  left son=6 (10 obs) right son=7 (14 obs)
  Primary splits:
       SoilTOC < 1.113  to the right, improve=0.7046088, (0 missing)
       SoilTN  < 0.0855 to the right, improve=0.6868612, (0 missing)
       SoilK   < 163.8  to the right, improve=0.4792320, (0 missing)
       Sand    < 34.18  to the left,  improve=0.4779311, (0 missing)
       SoilP   < 7.965  to the right, improve=0.4476148, (0 missing)
  Surrogate splits:
       SoilTN  < 0.0855 to the right, agree=0.958, adj=0.9, (0 split)
       SoilP   < 7.965  to the right, agree=0.917, adj=0.8, (0 split)
       SoilK   < 163.8  to the right, agree=0.917, adj=0.8, (0 split)
       Clay    < 30.16  to the right, agree=0.833, adj=0.6, (0 split)
       Sand    < 31.04  to the left,  agree=0.833, adj=0.6, (0 split)
```

The first split subdivides the field into high *Clay* and low *Clay* regions, which is consistent with earlier exploration. The main competitor and surrogate of *Clay* is, unsurprisingly, *Sand*. The second surrogate is *SoilK*. In the high *Clay* region (node 2), the split is on *SoilTN*. The primary competitors and surrogates of *SoilTN* are associated with other mineral nutrients or organic carbon. In the low *Clay* (high *Yield*) region (node 3), the split is on *SoilTOC*, with the main competitors and surrogates again being other mineral nutrient levels.

One of the most useful actions involving a regression tree such as that of Figure 9.15 with spatial data is to create a map of the terminal nodes of the tree. We use a Thiessen polygon map created in ArcGIS for this purpose. Figure 9.16a shows this map. It was created in the same manner as the maps in Chapter 7. The map serves as a reality check on the tree results. The regression tree (Figure 9.15) was created without the use of any spatial information, and displaying the spatial context of the results is an indication of their validity. One expects that the data records in each terminal node will have a spatial structure; a random spatial arrangement would be a clear counter-indication of the validity of the tree. Thus, the map of Figure 9.16a provides both a validation step for the results and a means of interpreting these results spatially. In this context is important to emphasize that the recursive partitioning models of this chapter and the regression models of Chapter 8 do not treat the data in the same way. A regression model incorporates the contribution of each explanatory variable to the response variable over the whole data space. The recursive partitioning model, on the other hand, recursively splits the data space, and at each split separately incorporates the contributions of the response variables only over that subset of the data space. It is of particular interest to determine whether these splits in the data space correspond to splits in the geographic space. The corresponding yield map of Figure 9.16b provides a spatial representation of the response variable of the tree. One counterintuitive result is that in the high-yielding southern end of the field, *Yield* is higher in the presence of low *SoilTOC* than high *SoilTOC*. This likely indicates that in this part of the variable *SoilTOC* is serving as a surrogate for some other variable, and the most likely candidates are *SoilTN*, *SoilP*, and *SoilK*.

We conclude this section by demonstrating a property of regression trees that can cause very serious problems. We would expect that the results of our analysis should not be

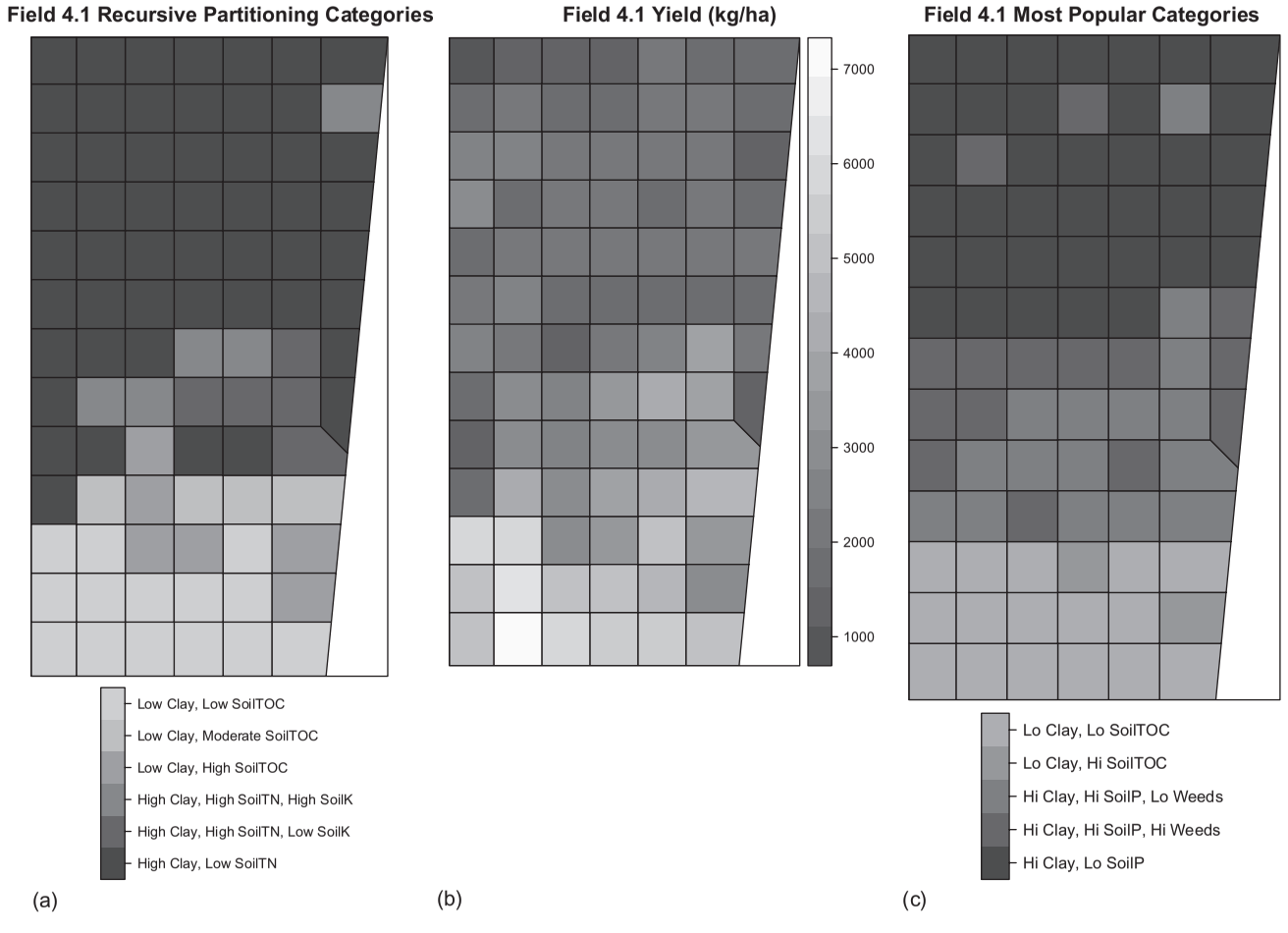

FIGURE 9.16
(a) Thematic map of the regression tree of Figure 9.15. (b) Thematic map of yield values at sample points in Field 4.1. (c) Thematic map of the ninth perturbation in Table 9.1 as a representative of the most "popular" regression tree.

sensitive to small perturbations of the data. We will test this property by employing a form of sensitivity test commonly used in engineering: one applies small random perturbations to the data and observes the effect (Clements, 1989, p. 129; Boyd, 2001; Ravindran et al., 2006, p. 590). Although this process could be automated, we will simply run the following code sequence (except the first two lines) repeatedly and enter the results into a table. The function `rnorm()` in the sixth line generates a random matrix, and the operation in the eighth row multiplies each element of the data frame by one plus a small random number. Note that in R the matrix operation `A * B` creates a matrix whose elements are $a_{ij} \times b_{ij}$.

```
>set.seed(123)
> n <- 0
> data.Perturb <- with(data.Set4.1, data.frame(Yield, Clay, Silt, Sand,
+    SoilpH, SoilTOC, SoilTN, SoilP, SoilK, CropDens, Weeds, Disease))
> epsilon <- 0.05
> mm <- matrix(rnorm(nrow(data.Perturb) * ncol(data.Perturb)),
+   nrow = nrow(data.Perturb))
> df2 <- data.Perturb * (1 + epsilon * mm)
> Perturb.rp <- rpart(Set4.1.model, data = df2, control = cont.parms,
+    method = "anova")
> n <- n + 1
> plot(Perturb.rp,branch = 0.4,uniform = T,margin = 0.1,
+    main = paste("Field 4-1 Yield Regression Tree ",
+    as.character(n)), cex.main = 2)
> text(Perturb.rp,use.n = T,all = F)
```

The R function `jitter()` could also be used, but in this case the explicit generation of the matrix seems more transparent. The ordinal scale variables *Weeds*, *Disease*, and *CropDens* are treated as if they were ratio scale, so that this analysis is very rough and ready. Table 9.1 shows the results of compiling the output of nine regression trees. Although nodes 1, 2, and 5 generally contain the same splitting variable, the other nodes do not. This illustrates the fact that regression trees can be *unstable.* That is, trees grown with small changes in the parameters or their values can change dramatically. This often occurs in cases where the splitting

TABLE 9.1

Splitting Variables at Each Node of a Succession of Regression Trees for Field 4.1, Each Generated with a Small Random Perturbation of the Data

Split	1	2	3	4	5	6	7	8	9
1	Clay	Clay	Clay	Sand	Clay	Clay	Clay	Clay	Clay
2	SoilTN	SoilP	SoilP	SoilP	Silt	SoilP	SoilP	SoilK	SoilK
3	SoilP	SoilTOC	SoilTOC		SoilTOC	SoilTOC	SoilK	SoilTOC	SoilTOC
4	SoilP				SoilP				
5		Weeds	Weeds	Weeds		Weeds	Weeds	SoilTN	SoilTN
6	SoilpH								
7						Disease	Weeds	Weeds	Weeds
8									
9	Weeds	SoilP			Weeds		SoilTOC	SoilTOC	
>9									

Note: Nodes left blank were not generated.

variable that defines a particular node changes, because this change tends to propagate and amplify as one moves through the branch emanating from that node. A second indication of Table 9.1 is that some trees are more "popular" than others. Figure 9.16c shows a thematic map of Tree 3 in Table 9.1 as a representative of the "popular" trees. Breiman (2001a) recognized these properties, instability and popularity, and both he and others have developed procedures to reduce the impact of tree instability by focusing on popular trees. In the next section, we will briefly describe *random forest*, one of the most useful of these procedures.

9.4 Random Forest

9.4.1 Introduction to Random Forest

In the previous section's sensitivity analysis, we used a method common to engineering, the perturbation of a data set by applying a small random variable to its values. The outcome of the sensitivity analysis, shown in Table 9.1, is that the regression tree is unstable, that is, it is highly sensitive to these perturbations. The *random forest* method developed by Breiman (2001a) also generates large numbers of trees, each of which is built based on a small perturbation of the original data set. The perturbations, however, are of a different form from those in Section 9.3.5. Rather than adding a small random number to each of the n data values, a random sample of the data records is drawn by sampling from these records *with replacement*. This procedure, repeatedly sampling with replacement from the existing data set and then carrying out a statistical analysis of the sample of samples, is called *bootstrapping* and is discussed in Chapter 10. The application of bootstrapping to an algorithm like recursive partitioning is called *bagging* (for "bootstrap aggregation") (Breiman, 1996). One practical advantage of bootstrapping over the sensitivity analysis of Section 9.3.5 is immediately apparent. As we saw in that section, adding a small random number to the data is an inappropriate modification for nominal or ordinal scale data such as the *Weeds*, *CropDens*, and *Disease* data fields in Data Set 4. The bootstrap procedure just described does not affect the form of the data themselves (see Exercise 9.12).

The random forest algorithm works with bootstrap samples as follows. Suppose n_{boot} bootstrap samples (i.e., samples with replacement from the original data set) are drawn. For each bootstrap sample, a classification or regression tree is generated using a modification of the recursive partitioning algorithm described in Section 9.3.2. The modification differs from the recursive partitioning algorithm discussed in that section in two ways. First, the maximal tree is grown and not pruned back. Second, somewhat surprisingly, the random forest algorithm is found to work better if not every possible explanatory variable is tested at each node to determine the next split. Therefore, at each node only a subset of size n_{test} of the variables is tested, and the splitting variable and value are selected from this subset.

The action of drawing n_{boot} samples with replacement and using them to grow a maximal tree on a randomly selected subset n_{test} of the variables is repeated N_T times. In this way, a "random forest" of N_T trees is grown. Of course, due to the instability of recursive partitioning, each of these trees may have different splitting variables at each node. The splitting variable or value ultimately selected at each node is determined by having the trees "vote." For classification trees, the predicted variable is that of the majority of the trees and for regression trees the predicted value is the average value of the majority variable.

One of the advantages of the random forest algorithm is that it provides a built-in data set that can be used for validation. Each of the n_{boot} bootstrap samples omit some data records. It turns out that on average about 36% of the data records are not included in each bootstrap sample (Breiman, 2001a). Since these data records are not used in constructing that particular tree, they can be used as a validation set by predicting the value of the response variable for each of them and comparing the predicted and actual values. For reasons explained by Breiman (2001a), the data records not included in the bootstrap sample are called *out of bag* (or OOB) data records.

The nodal structure of the random forest is not nearly as important as other information that the algorithm provides. To describe this, we need to have an example, and we will use the data from Field 4.1. After loading the data into the data frame `data.Set4.1`, we enter the following code.

```
> library(randomForest)
> data.rf <- with(data.Set4.1, data.frame(Yield, Clay, Silt, Sand,
+    SoilpH, SoilTOC, SoilTN, SoilP, SoilK, CropDens, Weeds, Disease))
> Set4.1.rf <- randomForest(Yield ~ ., data = data.rf,
+   importance = TRUE, proximity = TRUE)
```

The variable *Sand* is excluded on agronomic grounds that it is simply the reflection of *Clay*. Two points must be noted, one subtle and one obvious. The obvious point is that there are two arguments, `importance` and `proximity`, that we have not discussed yet. The more subtle point is that when we used the function `rpart()` to do recursive partitioning we first adjusted the tuning parameters `minsplit` and `cp`; here we did not adjust anything. There are two tuning parameters for the random forest algorithm, N_T and n_{test} defined above, but the algorithm is not very sensitive to them and we won't bother to adjust them from their default values.

The arguments `importance` and `proximity` instruct the random forest algorithm to compute the *variable importance* and the *proximity matrix*. The variable importance can be defined in two different ways, which are described in the `?importance` file of the function `importance()`. We will discuss this only for regression trees; the concept is analogous for classification trees. The first way to compute the importance is to rank the *increase in node purity*. For regression trees, this is the reduction in the residual sums of squares over the nodes. The second way is to rank the *percent increase in the mean square error*. Without going into detail, this involves computing the error based on the OOB data records. The function `varImpPlot()` generates a dotplot of the variable importance.

```
> varImpPlot(Set4.1.rf,
      main = "Data Set 4.1 Variable Importance") # Fig. 9.17
```

Figure 9.17 supports the notion that soil texture is the most important variable (or, perhaps better said, is the most important variable over the largest area of the field), and that mineral nutrients are also important. The variable *Weeds* is not listed as very important, which shows that with spatial data a variable that is very important in one region may not be judged important because it is not important over the whole area.

The proximity matrix, computed when the argument `proximity` is set at `TRUE`, is defined as follows (Liaw and Weiner, 2002). The (i, j) element of the proximity matrix is the fraction of the trees in which data records i and j fall in the same terminal node. The elements of the proximity matrix thus give a measure of the attribute proximity of two data records. The proximity matrix for Field 4.1 is an 86 by 86 matrix, so obviously it is not

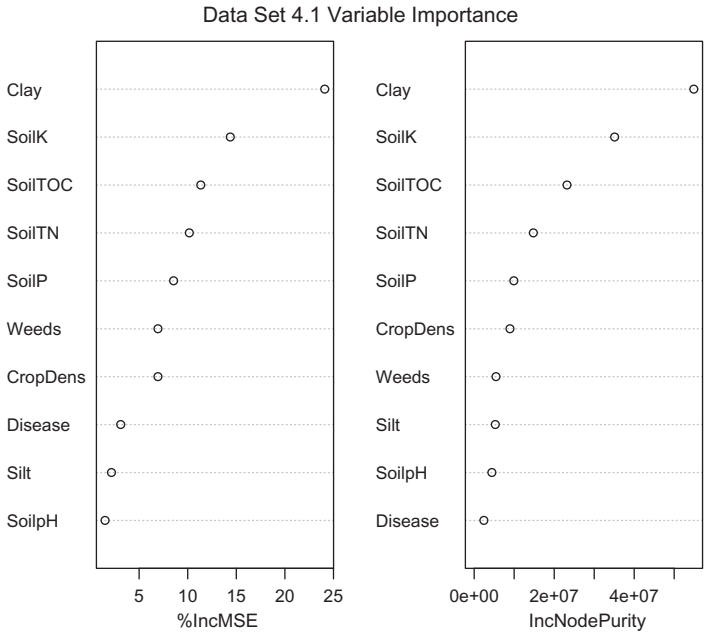

FIGURE 9.17
Variable importance values of the agronomic variables of Field 4.1.

very useful to just display it in its raw form. Hastie et al. (2009, p. 595) describe a graphical device called the proximity plot that can in principle be used to visualize the proximity matrix, but they point out that these plots are often of limited utility. With spatial data, however, one can obtain some information from this matrix. For example, for each data record we can identify the data record with the highest proximity. This provides an indication of the spatial structure. Unlike the spatial weights matrix discussed in Chapter 4, the proximity matrix generated by `randomForest()` has the number 1 on the diagonal, so we first need to eliminate these values. The function `closest()` defined here eliminates the diagonal elements and then identifies for a given row or column of the proximity matrix the largest off-diagonal element.

```
closest <- function(x){
   x[which(x > 0.999)] <- 0
   which(x == max(x))
}
```

We can use the function `apply()` to apply this function row-wise (or column-wise) to the proximity matrix.

```
> apply(Set4.1.rf$proximity, 1, closest)
 1  2  3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
18 17 10 7 4 5 4 9 8  3 56 10 60 52 39 22  2 17 20 21 20 23 22 29

25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
27 28 25 26 30 29 29 27 27 40 29 37 38 37 22 34 40 44 63 42 49 53 45 45
```

```
49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
45 62 57 54 54 53 53 61 51 61 65 64 56 50 67 59 59 72 68 67 86 71 79 74

73 74 75 76 77 78 79 80 81 82 83 84 85 86
74 80 76 84 76 70 71 74 82 83 82 76 86 71
```

Blank lines have been inserted in the output to make it easier to read. In each pair of rows, the first row is the data record number and the second row is the corresponding closest data record. Although the proximity matrix is symmetric, the closest proximity relation is not reflexive. That is, for example, although data record 18 is closest to data record 1, meaning that data record 18 has the most nodes in common with data record 1, the data record having the most nodes in common with 18 is data record 17. One can use this proximity matrix together with the record number map of Figure 4.5a to obtain a sense of the spatial structure of these attribute value relationships.

9.4.2 Application to Data Set 2

Turning to Data Set 2, we will again consider the Sierra Nevada and Coast Ranges separately. The `SpatialPointsDataFrame data.Set2S` created in Section 7.3 for the Sierra Nevada is loaded as described in Appendix B.2. Since the presence-absence variable *QUDO* is binary, we will convert it to a factor so that `randomForest()` develops a forest of classification trees rather than regression trees. Because there are so many data records, we will not develop the proximity matrix for this case. Here is the code to produce Figure 9.18a.

```
> library(randomForest)
> attr.S <- slot(data.Set2S, "data")
> data.rfS <- with(attr.S, data.frame(QUDO = factor(QUDO),
+   MAT, Precip, SolRad6, SolRad12, Texture, AWCAvg,
+   Permeab, PM100, PM200, PM300, PM400, PM500, PM600))
> set.seed(123)
> Set2.rf <- randomForest(QUDO ~ ., data = data.rfS,
+   importance = TRUE)
> varImpPlot(Set2.rf, # Fig. 9.18a
+   main = "Data Set 2 Sierra Variable Importance")
```

The soil-related explanatory variables are identified as the least important. Sometimes random forests can be used to indicate unimportant variables that may be candidates for elimination from the model (Liaw and Wiener, 2002). We can determine the effect of eliminating the soil variables as follows. The `randomForest` object itself contains the relative error, which is determined from attempting to predict the out-of-bag data records.

```
> Set2.rf
        OOB estimate of error rate: 12.39%
Confusion matrix:
    0   1 class.error
0 969 102   0.0952381
1 117 580   0.1678623
```

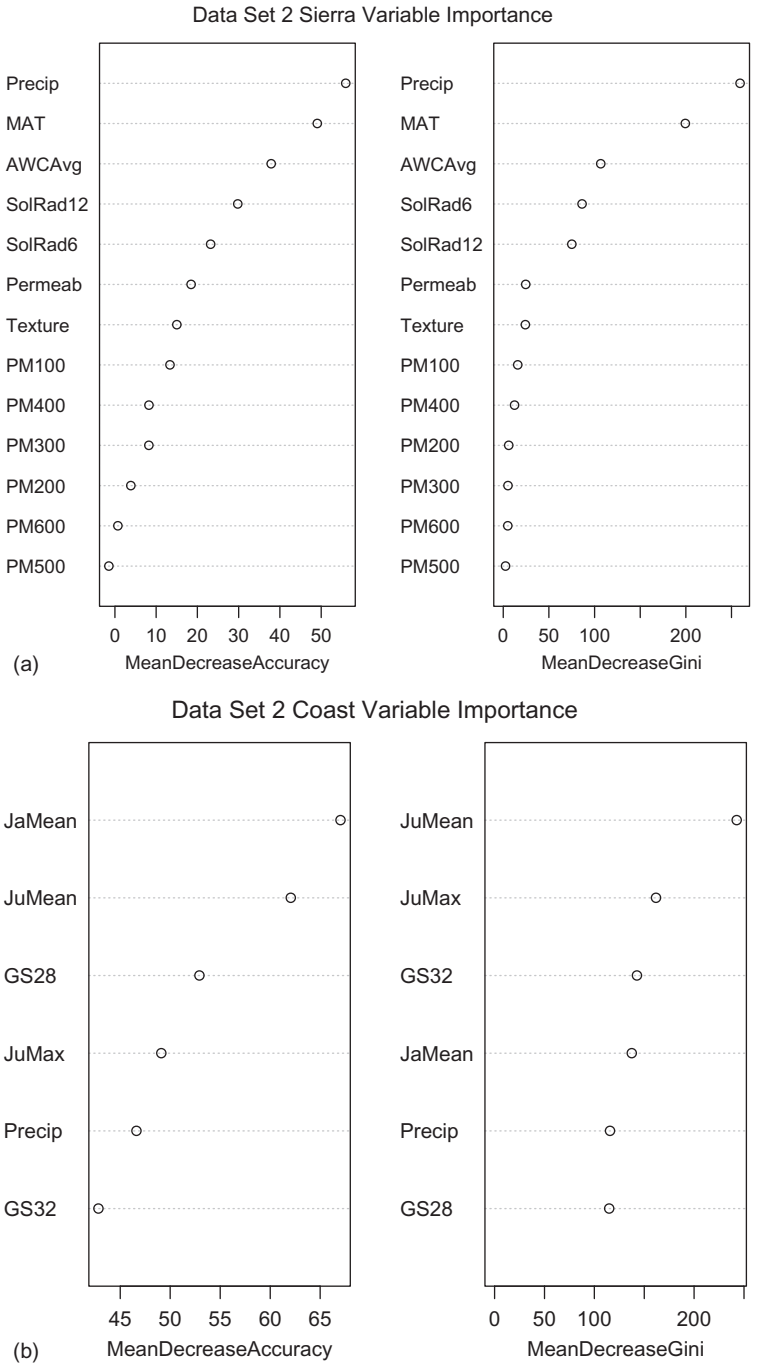

FIGURE 9.18
(a) Variable importance values of the Sierra Nevada subset of Data Set 2. (b) Same as (a) for the Coast Range subset of Data Set 2.

It turns out that if we eliminate all the soil variables from the model, then the prediction accuracy declines, but leaving one in retains a reasonable accuracy. By a process of trial and error, we find that the following model has almost as much predictive accuracy as the full model.

```
> set.seed(123)
> Set2.rf4 <- randomForest(QUDO ~ MAT + Precip + SolRad6 +
+   AWCAvg + Permeab, data = data.rfS, importance = TRUE)
> Set2.rf4
        OOB estimate of error rate: 12.9%
Confusion matrix:
    0   1 class.error
0 963 108   0.1008403
1 120 577   0.1721664
```

Carrying out the same exercise with the Coast Ranges data set (Exercise 9.14) produces Figure 9.18b and the following output.

```
> Set2.rf
        OOB estimate of error rate: 14.33%
Confusion matrix:
    0   1 class.error
0 874 128   0.1277445
1 138 716   0.1615925
```

Again, by using a process of trial and error to eliminate explanatory variables, we find that the following model has about as much predictive accuracy as the full model.

```
> set.seed(123)
> Set2.rf2 <- randomForest(QUDO ~ JuMax + JuMean + JaMean + GS32 +
+     GS28 + JaMean + Precip, data = data.rfC,
+    importance = TRUE)
> Set2.rf2
        OOB estimate of error rate: 15.95%
Confusion matrix:
    0   1 class.error
0 855 147   0.1467066
1 149 705   0.1744731
```

It is important to emphasize that this exercise in variable selection does not guarantee that the surviving variables are the most important explanatory variables for prediction of habitat suitability for blue oak. There is evidence that the importance measures show a bias toward correlated explanatory variables. We can draw the tentative conclusion from the recursive partitioning and random forest analyses that precipitation is important for explanation of blue oak presence in both the Sierra Nevada and the Coast Range, although precipitation is generally lower throughout most of the Coast Range. Some combination of temperature variables may also be important, particularly in the Coast Range, and soil properties seem less important.

9.5 Further Reading

As mentioned in Section 9.2, Wood (2006) is an excellent source of information on the generalized additive model. The standard source for recursive partitioning is Breiman et al. (1984), although discussions of the method are beginning to make it into standard regression texts (e.g., Kutner et al., 2005, p. 453). Two excellent discussions of machine learning techniques are Hastie et al. (2009) and James et al. (2013). In particular, Hastie et al. present in Chapter 5 a nice discussion of basis splines. For examples of the application of recursive partitioning to ecological problems, see for example Plant et al. (1999), De'ath and Fabricus (2000), De'ath (2002), Roel and Plant (2004a,b), and Lobell et al. (2005).

Classification and regression trees provide a means for the detection of boundaries, delimiting what ecologists call "patches" and agronomists call "management zones." For a more extensive treatment of this topic and the use of other methods to identify patches, see Fortin and Dale (2005). It is also worth noting that recursive partitioning has a natural application in conjunction with boundary line analysis (Webb, 1972; Lark, 1997). This is briefly discussed in Section 17.3.

Exercises

9.1 Determine the value of the knot displayed in Figure 9.3a directly from the function `bs()`. What is the class of the resulting quantity? Note that it is possible for elements of this class to have names.

9.2 Repeat Exercise 9.2 for the knots of Figure 9.3b and determine the difference between each of them.

9.3 Generalized additive models can be a very effective way to compute trend surfaces. In Section 3.2.1, we computed trend surfaces for a set of data shown in Figure 3.2a. Compute a trend surface using a GAM, plot the result as a perspective plot, and compare the result with the actual trend data shown in Figure 3.2b.

9.4 This exercise involves the construction of a classification tree. As mentioned in Chapter 2, R contains a number of built-in data sets, which are invoked through the function `data()`. One of these is Fisher's iris data, which we obtain by typing `data(iris)`. This is a famous data set employed by R.A. Fisher (1936) in a publication in a journal whose name reveals a dark side of the thinking of some prominent early twentieth-century statisticians, including Fisher, Galton, and others. The data set consists of 50 samples from each of three species of iris (*I. setosa*, *I. virginica*, and *I. versicolor*). Four features were measured from each sample: the length and width of the sepal (the sepal is the green part of the flower that separates the petals) and the length and width of the petal. Construct a classification tree using only the variables *Sepal width* and *Sepal length* and construct a plot of the data space showing the partitions.

9.5 Carry out a recursive partitioning analysis of the factors influencing the presence or absence of blue oak in the Coast Range of Data Set 2. Construct the regression tree shown in Figure 9.10b.

9.6 With the data of Data Set 3, display sorted means by farmer of yield, soil pH, and silt content.

9.7 With the data of Data Set 3, plot yield vs. nitrogen rate N with separate symbols for the northern, central, and southern regions.

9.8 With the data of Data Set 3, construct the regression tree for all management variables.

9.9 With the data of Data Set 3, construct a table in which the rows are mean values of the exogenous variables for each field and the columns are arranged in order of decreasing mean yield over the field.

9.10 Carry out a recursive partitioning with *Yield* as the response variable for Field 4.2.

9.11 Instead of checking the stability of a regression tree by perturbing the data by adding a small value to each attribute value, one can perturb them by replacing the set of n data records with a set of n records randomly selected from the original n with replacement. Use the function `sample()` to carry out this method to check for stability of the regression tree generated with the Field 4.1 data in Section 9.3.5 and compare the results with those obtained in that section.

9.12 Using a perturbation method similar to that employed in Exercise 9.11, check the Data Set 2 trees for stability. What does this imply about how the size of the data set influences the stability of the recursive partitioning?

9.13 Using *Yield* as a response variable, carry out a random forest analysis of Field 2 of Data Set 4.

9.14 Carry of a random forest analysis of the data from the Coast Range subset of Data Set 2.