CHAPTER 9
Classification and Regression Trees

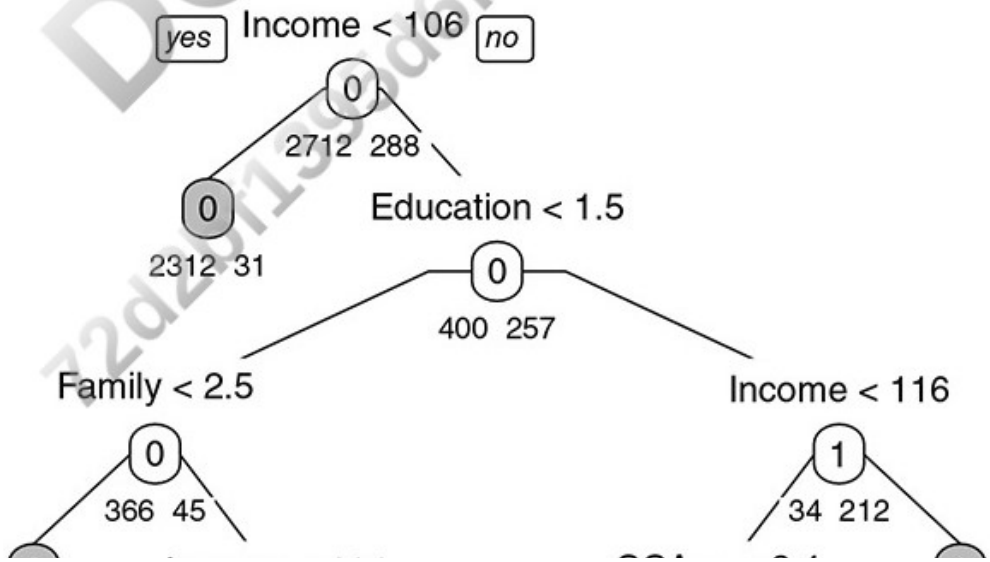
This chapter describes a flexible data-driven method that can be used for both classification (called *classification tree*) and prediction (called *regression tree*). Among the data-driven methods, trees are the most transparent and easy to interpret. Trees are based on separating records into subgroups by creating splits on predictors. These splits create logical rules that are transparent and easily understandable, for example, "IF Age < 55 AND Education > 12 THEN class = 1." The resulting subgroups should be more homogeneous in terms of the outcome variable, thereby creating useful prediction or classification rules. We discuss the two key ideas underlying trees: *recursive partitioning* (for constructing the tree) and *pruning* (for cutting the tree back). In the context of tree construction, we also describe a few metrics of homogeneity that are popular in tree algorithms, for determining the homogeneity of the resulting subgroups of records. We explain that pruning is a useful strategy for avoiding overfitting and show how it is done. We also describe alternative strategies for avoiding overfitting. As with other data-driven methods, trees require large amounts of data. However, once constructed, they are computationally cheap to deploy even on large samples. They also have other advantages such as being highly automated, robust to outliers, and able to handle missing values. In addition to prediction and classification, we describe how trees can be used for dimension reduction. Finally, we introduce *random forests* and *boosted trees*, which combine results from multiple trees to improve predictive power.

9.1 Introduction

If one had to choose a classification technique that performs well across a wide range of situations without requiring much effort from the analyst while being readily understandable by the consumer of the analysis, a strong contender would be the tree methodology developed by Breiman et al. (1984). We discuss this classification procedure first, then in later sections we show how the procedure can be extended to prediction of a numerical outcome. The program that Breiman et al. created to implement these procedures was called CART (Classification And Regression Trees). A related procedure is called C4.5.

What is a classification tree? Figure 9.1 shows a tree for classifying bank customers who receive a loan offer as either acceptors or nonacceptors, based on information such as their income, education level, and average credit card expenditure. One of the reasons that tree classifiers are very popular is that they provide easily understandable classification rules (at least if the trees are not too large). Consider the tree in the example. The gray *terminal nodes* are marked with 0 or 1 corresponding to a nonacceptor (0) or acceptor (1). The values above the white nodes give the splitting value on a predictor; those below give the counts of the two classes (0 and 1) in that node. This tree can easily be translated into a set of rules for classifying a bank customer. For example, the bottom-left rectangle node under the "Family" circle in this tree gives us the following rule:

IF(*Income* ≥ 106) AND (*Education* < 1.5) AND (*Family* < 2.5)
THEN *Class* = 0 (nonacceptor).



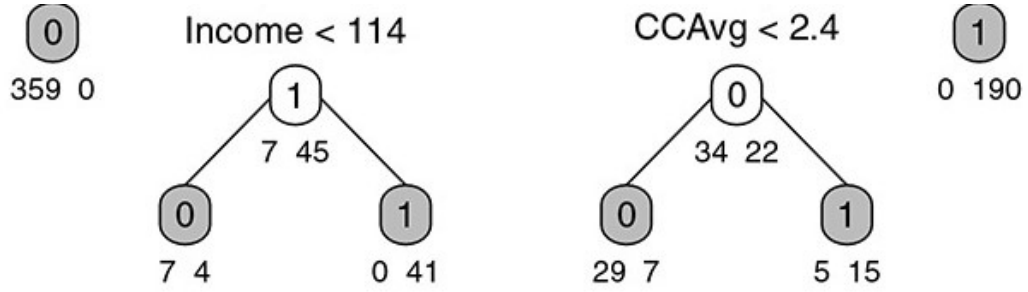


Figure 9.1 Best-pruned tree obtained by fitting a full tree to the training data

In the following, we show how trees are constructed and evaluated.

9.2 Classification Trees

Two key ideas underlie classification trees. The first is the idea of *recursive partitioning* of the space of the predictor variables. The second is the idea of *pruning* using validation data. In the next few sections, we describe recursive partitioning and in subsequent sections explain the pruning methodology.

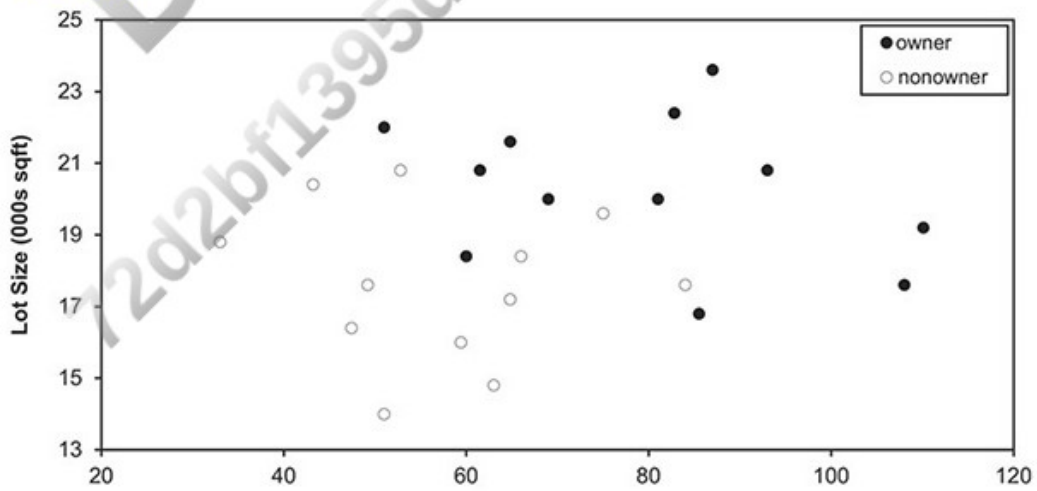
Recursive Partitioning

Let us denote the outcome variable by Y and the input (predictor) variables by $X_1, X_2, X_3, \ldots, X_p$. In classification, the outcome variable will be a categorical variable. Recursive partitioning divides up the p-dimensional space of the X predictor variables into nonoverlapping multidimensional rectangles. The predictor variables here are considered to be continuous, binary, or ordinal. This division is accomplished recursively (i.e., operating on the results of prior divisions). First, one of the predictor variables is selected, say X_i, and a value of X_i, say s_i, is chosen to split the p-dimensional space into two parts: one part that contains all the points with $X_i < s_i$ and the other with all the points with $X_i \geq s_i$. Then, one of these two parts is divided in a similar manner by again choosing a predictor variable (it could be X_i or another variable) and a split value for that variable. This results in three (multidimensional) rectangular regions. This process is continued so that we get smaller and smaller rectangular regions. The idea is to divide the entire X-space up into rectangles such that each rectangle is as homogeneous or "pure" as possible. By *pure*, we mean containing records that belong to just one class. (Of course, this is not always possible, as there may be records that belong to different classes but have exactly the same values for every one of the predictor variables.)

Let us illustrate recursive partitioning with an example.

Example 1: Riding Mowers

We again use the riding-mower example presented in Chapter 3. A riding-mower manufacturer would like to find a way of classifying families in a city into those likely to purchase a riding mower and those not likely to buy one. A pilot random sample of 12 owners and 12 nonowners in the city is undertaken. The data are shown and plotted in Table 9.1 and Figure 9.2.



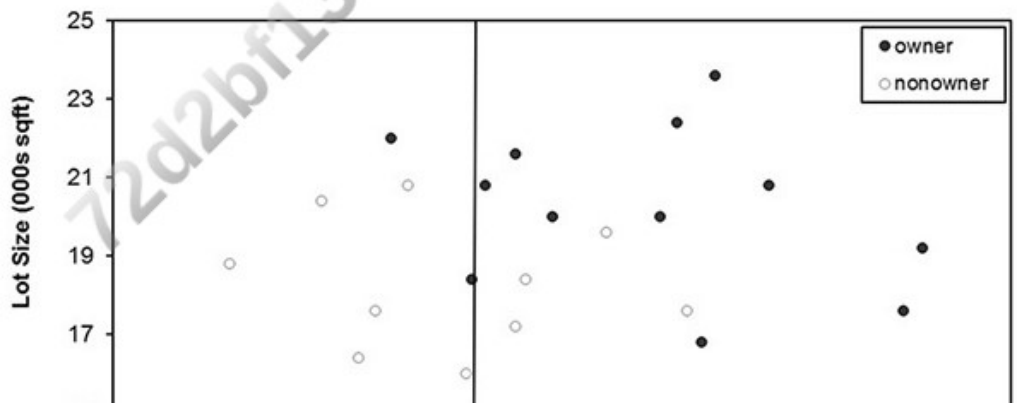
Income ($000s)

Figure 9.2 Scatter plot of Lot Size Vs. Income for 24 owners and nonowners of riding mowers

Table 9.1 Lot Size, Income, and Ownership of a Riding Mower for 24 Households

Household Number	Income ($000s)	Lot Size (000s ft^2)	Ownership of Riding Mower
1	60.0	18.4	Owner
2	85.5	16.8	Owner
3	64.8	21.6	Owner
4	61.5	20.8	Owner
5	87.0	23.6	Owner
6	110.1	19.2	Owner
7	108.0	17.6	Owner
8	82.8	22.4	Owner
9	69.0	20.0	Owner
10	93.0	20.8	Owner
11	51.0	22.0	Owner
12	81.0	20.0	Owner
13	75.0	19.6	Nonowner
14	52.8	20.8	Nonowner
15	64.8	17.2	Nonowner
16	43.2	20.4	Nonowner
17	84.0	17.6	Nonowner
18	49.2	17.6	Nonowner
19	59.4	16.0	Nonowner
20	66.0	18.4	Nonowner
21	47.4	16.4	Nonowner
22	33.0	18.8	Nonowner
23	51.0	14.0	Nonowner
24	63.0	14.8	Nonowner

If we apply the classification tree procedure to these data, the procedure will choose *Income* for the first split with a splitting value of 60. The (X_1, X_2) space is now divided into two rectangles, one with Income < 60 and the other with Income ≥ 60. This is illustrated in Figure 9.3.



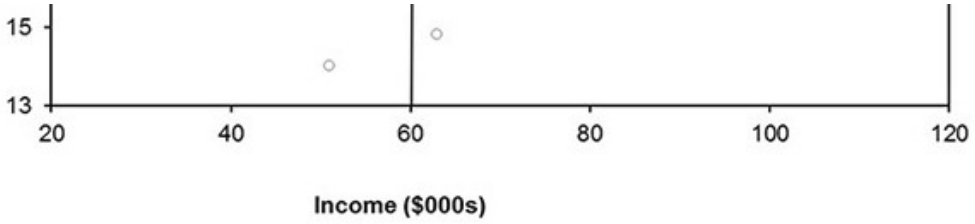


Figure 9.3 Splitting the 24 records by Income value of 60

Notice how the split has created two rectangles, each of which is much more homogeneous than the rectangle before the split. The left rectangle contains points that are mostly nonowners (seven nonowners and one owner) and the right rectangle contains mostly owners (11 owners and five nonowners).

How was this particular split selected? The algorithm examined each predictor variable (in this case, Income and Lot Size) and all possible split values for each variable to find the best split. What are the possible split values for a variable? They are simply the values for each predictor. The possible split points for Income are {33.0, 43.2, 47.4, …, 110.1} and those for Lot Size are {14.0, 14.8, 16.0, …, 23.6}. These split points are ranked according to how much they reduce impurity (heterogeneity) in the resulting rectangle. A pure rectangle is one that is composed of a single class (e.g., owners). The reduction in impurity is defined as overall impurity before the split minus the sum of the impurities for the two rectangles that result from a split.

Categorical Predictors

The previous description used numerical predictors; however, categorical predictors can also be used in the recursive partitioning context. To handle categorical predictors, the split choices for a categorical predictor are all ways in which the set of categories can be divided into two subsets. For example, a categorical variable with four categories, say {a, b, c, d}, can be split in seven ways into two subsets: {a} and {b, c, d}; {b} and {a, c, d}; {c} and {a, b, d}; {d} and {a, b, c}; {a, b} and {c, d}; {a, c} and {b, d}; and finally {a, d} and {b, c}. When the number of categories is large, the number of splits becomes very large.

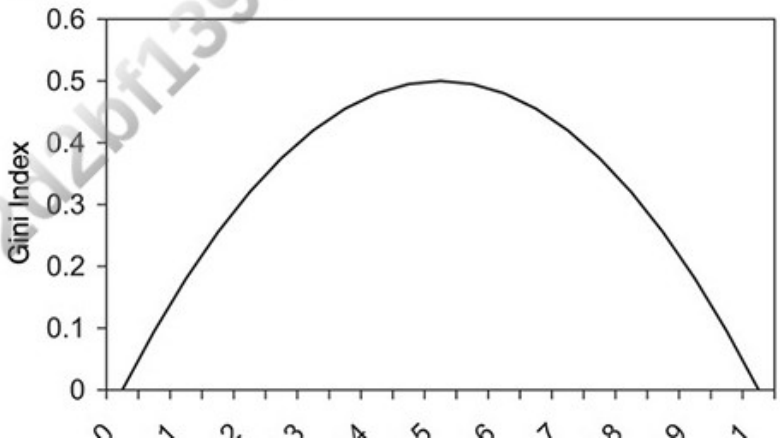
Measures of Impurity

There are a number of ways to measure impurity. The two most popular measures are the *Gini index* and an *entropy measure*. We describe both next. Denote the m classes of the response variable by $k = 1, 2, \ldots, m$.

The Gini impurity index for a rectangle A is defined by

$$I(A) = 1 - \sum_{k=1}^{m} p_k^2,$$

where p_k is the proportion of records in rectangle A that belong to class k. This measure takes values between 0 (when all the records belong to the same class) and $(m - 1)/m$ (when all m classes are equally represented). Figure 9.4 shows the values of the Gini index for a two-class case as a function of p_k. It can be seen that the impurity measure is at its peak when $p_k = 0.5$ (i.e., when the rectangle contains 50% of each of the two classes).



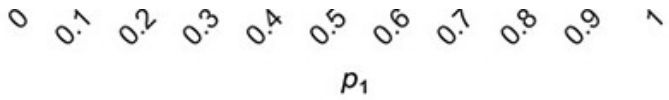


Figure 9.4 Values of the Gini index for a two-class case as a function of the proportion of records in class 1 (p_1)

A second impurity measure is the entropy measure. The entropy for a rectangle A is defined by

$$\text{entropy}(A) = -\sum_{k=1}^{m} p_k \log_2(p_k)$$

(to compute $\log_2(x)$ in R, use function *log2()*). This measure ranges between 0 (most pure, all records belong to the same class) and $\log_2(m)$ (when all m classes are represented equally). In the two-class case, the entropy measure is maximized (like the Gini index) at $p_k = 0.5$.

Let us compute the impurity in the riding mower example before and after the first split (using Income with the value of 60). The unsplit dataset contains 12 owners and 12 nonowners. This is a two-class case with an equal number of records from each class. Both impurity measures are therefore at their maximum value: Gini = 0.5 and entropy = $\log_2(2) = 1$. After the split, the left rectangle contains seven nonowners and one owner. The impurity measures for this rectangle are:

$$\text{Gini\_left} = 1 - (7/8)^2 - (1/8)^2 = 0.219$$

$$\text{entropy\_left} = -(7/8)\log_2(7/8) - (1/8)\log_2(1/8) = 0.544$$

The right rectangle contains 11 owners and five nonowners. The impurity measures of the right rectangle are therefore

$$\text{Gini\_right} = 1 - (11/16)^2 - (5/16)^2 = 0.430$$

$$\text{entropy\_right} = -(11/16)\log_2(11/16) - (5/16)\log_2(5/16) = 0.896$$

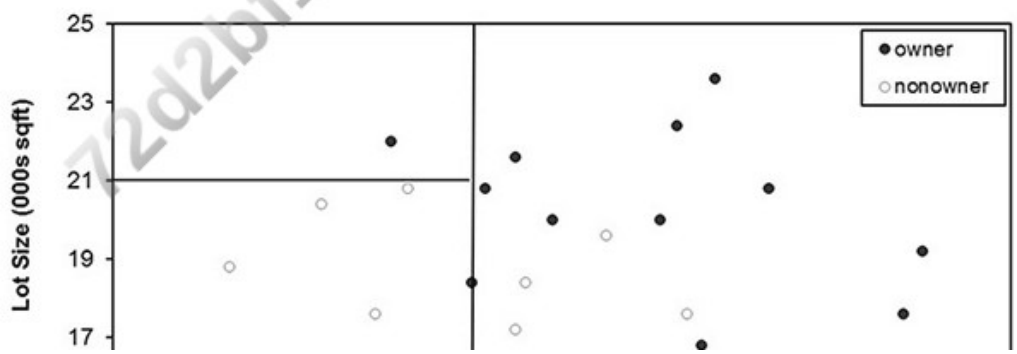
The combined impurity of the two rectangles that were created by the split is a weighted average of the two impurity measures, weighted by the number of records in each:

$$\text{Gini} = (8/24)(0.219) + (16/24)(0.430) = 0.359$$

$$\text{entropy} = (8/24)(0.544) + (16/24)(0.896) = 0.779$$

Thus, the Gini impurity index decreased from 0.5 before the split to 0.359 after the split. Similarly, the entropy impurity measure decreased from 1 before the split to 0.779 after the split.

By comparing the reduction in impurity across all possible splits in all possible predictors, the next split is chosen. If we continue splitting the mower data, the next split is on the Lot Size variable at the value 21. Figure 9.5 shows that once again the tree procedure has astutely chosen to split a rectangle to increase the purity of the resulting rectangles. The lower-left rectangle, which contains data points with Income < 60 and Lot Size < 21, has all points that are nonowners; whereas the upper left rectangle, which contains data points with Income < 60 and Lot Size ≥ 21, consists exclusively of a single owner. In other words, the two left rectangles are now "pure." We can see how the recursive partitioning is refining the set of constituent rectangles to become purer as the algorithm proceeds. The final stage of the recursive partitioning is shown in Figure 9.6. Notice that each rectangle is now pure: it contains data points from just one of the two classes.



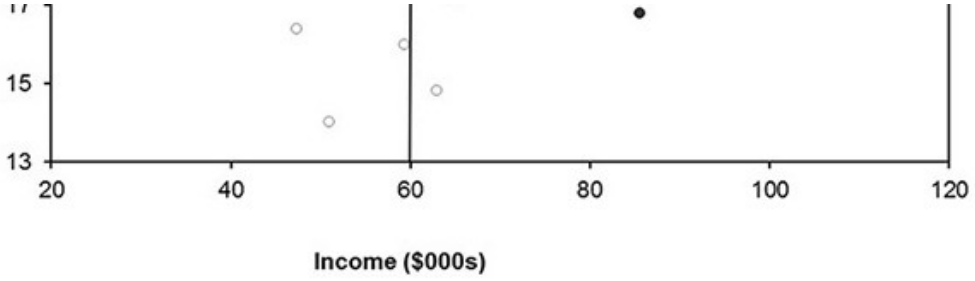


Figure 9.5 Splitting the 24 records first by Income value of 60 and then Lot size value of 21

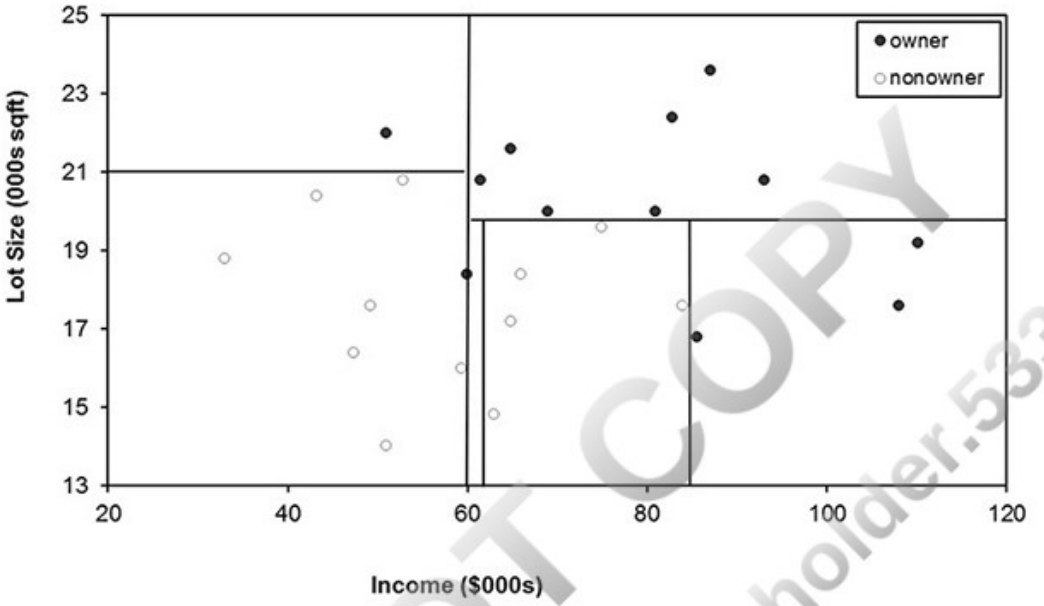


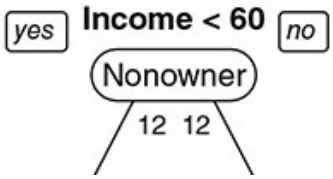
Figure 9.6 Final stage of recursive partitioning; each rectangle consisting of a single class (owners or nonowners)

The reason the method is called a *classification tree algorithm* is that each split can be depicted as a split of a node into two successor nodes. The first split is shown as a branching of the root node of a tree in Figure 9.7. The full-grown tree is shown in Figure 9.8. (Note that in R the split values are integers).

code for running and plotting classification tree with single split

```
library(rpart)
library(rpart.plot)
mower.df <- read.csv("RidingMowers.csv")

# use rpart() to run a classification tree.
# define rpart.control() in rpart() to determine the depth of the tree.
class.tree <- rpart(Ownership ~ ., data = mower.df,
    control = rpart.control(maxdepth = 2), method = "class")
## plot tree
# use prp() to plot the tree. You can control plotting parameters such as color, shape,
# and information displayed (which and where).
prp(class.tree, type = 1, extra = 1, split.font = 1, varlen = -10)
```



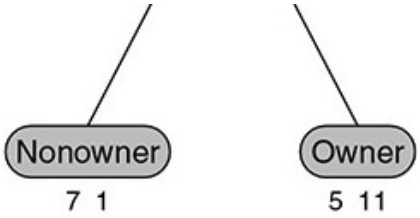


Figure 9.7 Tree representation of first split (corresponds to Figure 9.3)

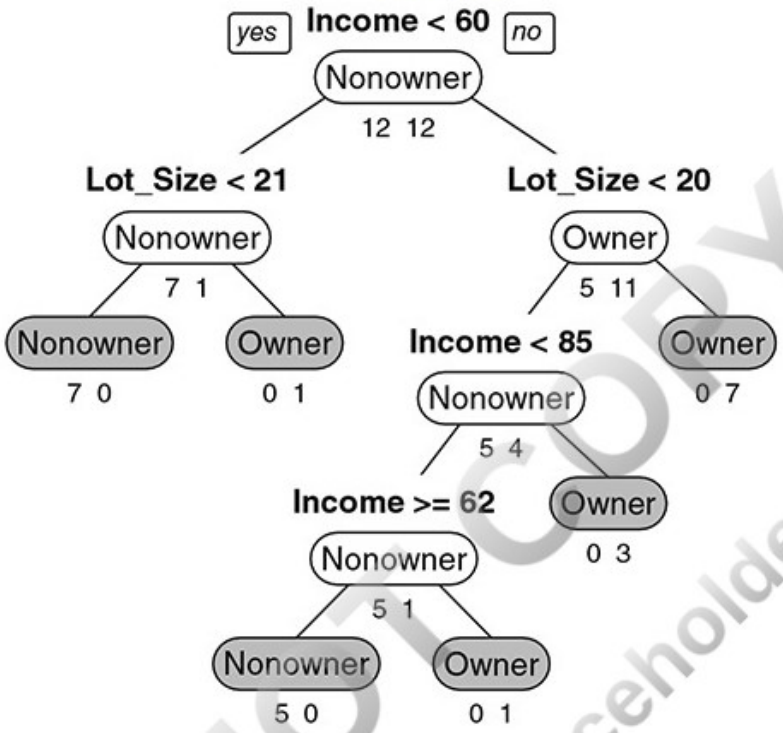


Figure 9.8 Tree representation after all splits (corresponds to Figure 9.6). This is the full grown tree

Tree Structure

We have two types of nodes in a tree: decision nodes and terminal nodes. Nodes that have successors are called *decision nodes* because if we were to use a tree to classify a new record for which we knew only the values of the predictor variables, we would "drop" the record down the tree so that at each decision node, the appropriate branch is taken until we get to a node that has no successors. Such nodes are called the *terminal nodes* (or *leaves* of the tree), and represent the partitioning of the data by predictors.

It is useful to note that the type of trees grown by R's *rpart()* function, also known as CART or *binary trees*, have the property that the number of terminal nodes is exactly one more than the number of decision nodes.

When using the R function *prp()* for plotting a tree, decision nodes are depicted by white ovals, and terminal nodes by gray ovals.

The name of the variable chosen for splitting and its splitting value are above each decision node. The numbers below a node are the number of records in that node that had values lesser than (left side) or larger or equal to (right side) the splitting value. ("yes" and "no" tags at the top node clarify on which side we find the lesser values).

Classifying a New Record

To classify a new record, it is "dropped" down the tree. When it has dropped all the way down to a terminal node, we can assign its class simply by taking a "vote" of all the training data that belonged to the terminal node when the tree was grown. The class with the highest vote is assigned to the new record. For instance, a new record reaching the rightmost terminal node in Figure 9.8, which has a majority of records that belong to the owner class, would be classified as "owner." Alternatively, if a single class is of interest, the algorithm counts the number of "votes" for this class, converts it to a proportion (propensity), then compares it to a user-specified cutoff value. See Chapter 5 for

class, convert it to a proportion (propensity), then compare it to a user-specified cutoff value. See Example 5 for further discussion of the use of a cutoff value in classification, for cases where a single class is of interest.

In a binary classification situation (typically, with a success class that is relatively rare and of particular interest), we can also establish a lower cutoff to better capture those rare successes (at the cost of lumping in more failures as successes). With a lower cutoff, the votes for the *success* class only need attain that lower cutoff level for the entire terminal node to be classified as a *success*. The cutoff therefore determines the proportion of votes needed for determining the terminal node class.