We have seen with previous methods that the modeling job is not completed by fitting a model to training data; we need out-of-sample data to assess and tune the model. This is particularly true with classification and regression trees, for two reasons:

- Tree structure can be quite unstable, shifting substantially depending on the sample chosen.
- A fully-fit tree will invariably lead to overfitting.

To visualize the first challenge, potential instability, imagine that we partition the data randomly into two samples, A and B, and we build a tree with each. If there are several predictors of roughly equal predictive power, you can see that it would be easy for samples A and B to select different predictors for the top level split, just based on which records ended up in which sample. And a different split at the top level would likely cascade down and yield completely different sets of rules. So we should view the results of a single tree with some caution.

To illustrate the second challenge, overfitting, let's examine another example.

Example 2: Acceptance of Personal Loan

Universal Bank is a relatively young bank that is growing rapidly in terms of overall customer acquisition. The majority of these customers are liability customers with varying sizes of relationship with the bank. The customer base of asset customers is quite small, and the bank is interested in growing this base rapidly to bring in more loan business. In particular, it wants to explore ways of converting its liability (deposit) customers to personal loan customers.

A campaign the bank ran for liability customers showed a healthy conversion rate of over 9% successes. This has encouraged the retail marketing department to devise smarter campaigns with better target marketing. The goal of our analysis is to model the previous campaign's customer behavior to analyze what combination of factors make a customer more likely to accept a personal loan. This will serve as the basis for the design of a new campaign.

Our predictive model will be a classification tree. To assess the accuracy of the tree in classifying new records, we start with the tools and criteria discussed in Chapter 5—partitioning the data into training and validation sets, and later introduce the idea of *cross-validation*.

The bank's dataset includes data on 5000 customers. The data include customer demographic information (age, income, etc.), customer response to the last personal loan campaign (*Personal Loan*), and the customer's relationship with the bank (mortgage, securities account, etc.). Table 9.2 shows a sample of the bank's customer database for 20 customers, to illustrate the structure of the data. Among these 5000 customers, only 480 (= 9.6%) accepted the personal loan that was offered to them in the earlier campaign.

Table 9.2 Sample of Data for 20 Customers of Universal Bank

ID	Age	Professional Experience	Income	Family Size	CC Avg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online Banking	Credit Card
1	25	1	49	4	1.60	UG	0	No	Yes	No	No	No
2	45	19	34	3	1.50	UG	0	No	Yes	No	No	No
3	39	15	11	1	1.00	UG	0	No	No	No	No	No
4	35	9	100	1	2.70	Grad	0	No	No	No	No	No
5	35	8	45	4	1.00	Grad	0	No	No	No	No	Yes
6	37	13	29	4	0.40	Grad	155	No	No	No	Yes	No
7	53	27	72	2	1.50	Grad	0	No	No	No	Yes	No
8	50	24	22	1	0.30	Prof	0	No	No	No	No	Yes
9	35	10	81	3	0.60	Grad	104	No	No	No	Yes	No
10	34	9	180	1	8.90	Prof	0	Yes	No	No	No	No
11	65	39	105	4	2.40	Prof	0	No	No	No	No	No
12	29	5	45	3	0.10	Grad	0	No	No	No	Yes	No
13	48	23	114	2	3.80	Prof	0	No	Yes	No	No	No
14	59	32	40	4	2.50	Grad	0	No	No	No	Yes	No
15	67	41	112	1	2.00	UG	0	No	Yes	No	No	No
16	60	30	22	1	1.50	Prof	0	No	No	No	Yes	Yes
17	38	14	130	4	4.70	Prof	134	Yes	No	No	No	No
18	42	18	81	4	2.40	UG	0	No	No	No	No	No
19	46	21	193	2	8.10	Prof	0	Yes	No	No	No	No
20	55	28	21	1	0.50	Grad	0	No	Yes	No	No	Yes

After randomly partitioning the data into training (3000 records) and validation (2000 records), we use the training

After randomly partitioning the data into training (3000 records) and validation (2000 records), we use the training data to construct a tree. A tree with 7 splits is shown in Figure 9.9 (this is the default tree produced by *rpart()* for this data). The top node refers to all the records in the training set, of which 2709 customers did not accept the loan and 291 customers accepted the loan. The "0" in the top node's rectangle represents the majority class ("did not accept" = 0). The first split, which is on the income variable, generates left and right child nodes. To the left is the child node with customers who have income less than 114. Customers with income greater than or equal to 114 go to the right. The splitting process continues; where it stops depends on the parameter settings of the algorithm. The eventual classification of customer appears in the terminal nodes. Of the eight terminal nodes, four lead to classification of "did not accept" and four lead to classification of "accept."

code for creating a default classification tree

```
library(rpart)
library(rpart.plot)

bank.df <- read.csv("UniversalBank.csv")
bank.df <- bank.df[ , -c(1, 5)]  # Drop ID and zip code columns.

# partition
set.seed(1)
train.index <- sample(c(1:dim(bank.df)[1]), dim(bank.df)[1]*0.6)
train.df <- bank.df[train.index, ]
valid.df <- bank.df[-train.index, ]

# classification tree
default.ct <- rpart(Personal.Loan ~ ., data = train.df, method = "class")
# plot tree
prp(default.ct, type = 1, extra = 1, under = TRUE, split.font = 1, varlen = -10)
```

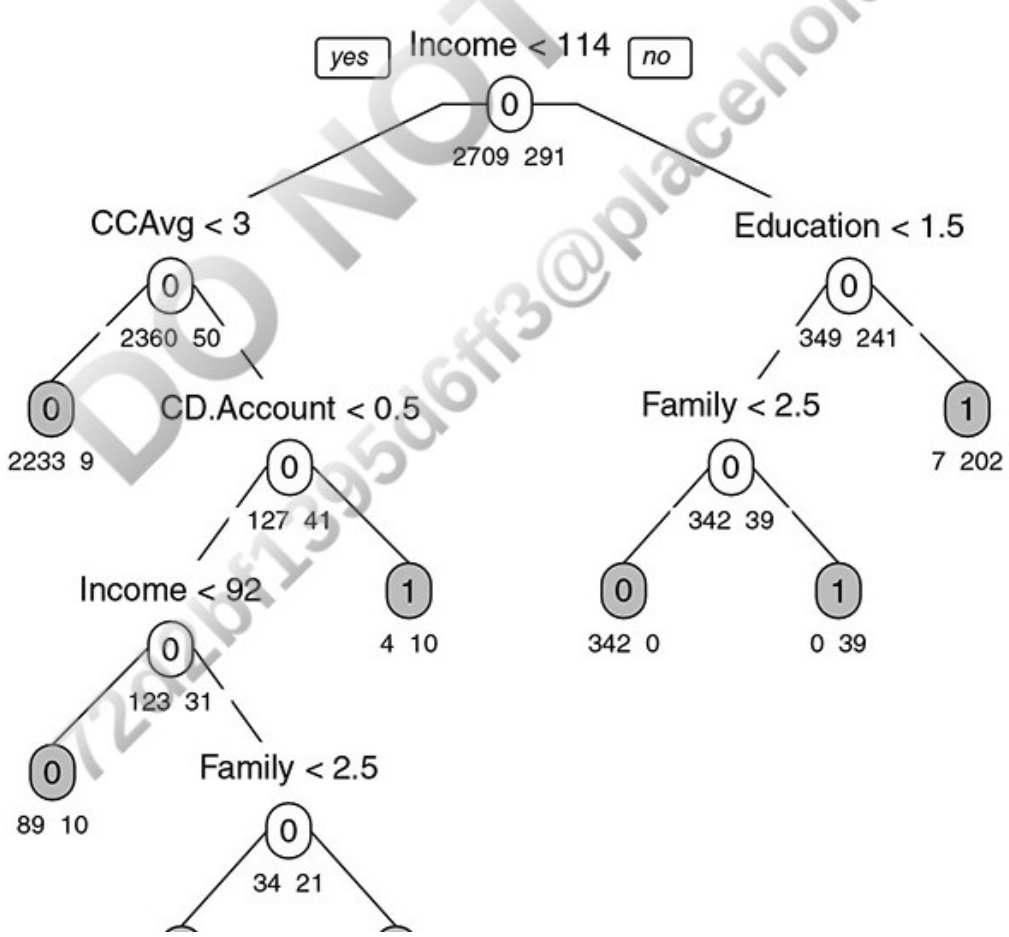

Figure 9.9 Default classification tree for the loan acceptance data using the training set (3000 records)

A full-grown tree is shown in Figure 9.10. Although it is difficult to see the exact splits, let us assess the performance of this tree with the validation data and compare it to the smaller default tree. Each record in the validation data is "dropped down" the tree and classified according to the terminal node it reaches. These predicted classes can then be compared to the actual memberships via a confusion matrix. When a particular class is of interest, a lift chart is useful for assessing the model's ability to capture those members.

code for creating a deeper classification tree

```
deeper.ct <- rpart(Personal.Loan ~ ., data = train.df, method = "class", cp = 0, minsplit = 1)
# count number of leaves
length(deeper.ct$frame$var[deeper.ct$frame$var == ""])
# plot tree
prp(deeper.ct, type = 1, extra = 1, under = TRUE, split.font = 1, varlen = -10,
    box.col=ifelse(deeper.ct$frame$var == "", 'gray', 'white'))
```

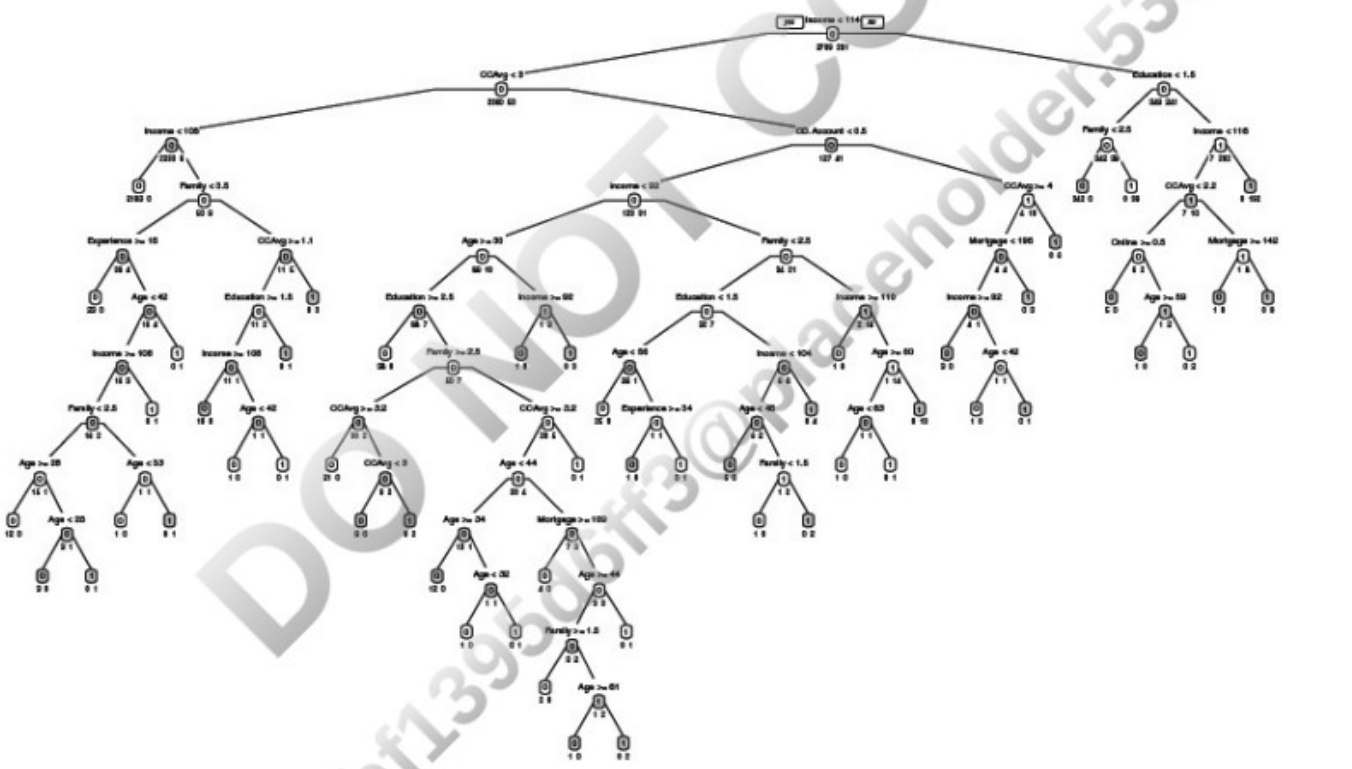

Figure 9.10 A full tree for the loan acceptance data using the training set (3000 records)

Table 9.3 displays the confusion matrices for the training and validation sets of the small default tree (top) and for the full tree (bottom). Comparing the two training matrices, we see that the full tree has higher accuracy: it is 100% accurate in classifying the training data, which means it has completely pure terminal nodes. In contrast, the confusion matrices for the validation data show that the smaller tree is more accurate. The main reason is that the full-grown tree overfits the training data (to perfect accuracy!). This motivates the next section, where we describe ways to avoid overfitting by either stopping the growth of the tree before it is fully grown or by pruning the full-grown tree.

Table 9.3 Confusion matrices and accuracy for the default (small) and deeper (full) classification trees, on the training and validation sets of the personal loan data

training and validation sets of the personal loan data

 code for classifying the validation data using a tree and computing the confusion matrices and accuracy for the training and validation data

```
# classify records in the validation data.
# set argument type = "class" in predict() to generate predicted class membership.
default.ct.point.pred.train <- predict(default.ct,train.df,type = "class")
# generate confusion matrix for training data
confusionMatrix(default.ct.point.pred.train, train.df$PersonalLoan)
### repeat the code for the validation set, and the deeper tree
Output
> # default tree: training
> confusionMatrix(default.ct.point.pred.train, train.df$Personal.Loan)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2696   26
         1   13  265

               Accuracy : 0.987

> # default tree: validation
> confusionMatrix(default.ct.point.pred.valid, valid.df$Personal.Loan)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1792   18
         1   19  171

               Accuracy : 0.9815

> # deeper tree: training
> confusionMatrix(deeper.ct.point.pred.train, train.df$Personal.Loan)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2709    0
         1    0  291

               Accuracy : 1

> # deeper tree: validation
> confusionMatrix(deeper.ct.point.pred.valid, valid.df$Personal.Loan)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1788   19
         1   23  170

               Accuracy : 0.979
```

9.4 Avoiding Overfitting

One danger in growing deeper trees on the training data is overfitting. As discussed in Chapter 5, overfitting will lead to poor performance on new data. If we look at the overall error at the various sizes of the tree, it is expected to decrease as the number of terminal nodes grows until the point of overfitting. Of course, for the training data the overall error decreases more and more until it is zero