CHAPTER

7

Decision Trees

Decision trees are one of the most powerful directed data mining techniques, because you can use them on such a wide range of problems and they produce models that explain how they work. Decision trees are related to table lookup models. In the simple table lookup model described in Chapter 6, such as RFM cubes, the cells are defined in advance by splitting each dimension into an arbitrary number of evenly spaced partitions. Then, something of interest — a response rate or average order size, for instance — is measured in each cell. New records are scored by determining which cell they belong to.

Decision trees extend this idea in two ways. First, decision trees recursively split data into smaller and smaller cells which are increasingly "pure" in the sense of having similar values of the target. The decision tree algorithm treats each cell independently. To find a new split, the algorithm tests splits based on all available variables. In doing so, decision trees choose the most important variables for the directed data mining task. This means that you can use decision trees for variable selection as well as for building models.

Second, the decision tree uses the target variable to determine how each input should be partitioned. In the end, the decision tree breaks the data into segments, defined by the splitting rules at each step. Taken together, the rules for all the segments form the decision tree model.

A model that can be expressed as a collection of rules is very attractive. Rules are readily expressed in English so that we can understand them. Rules, of the kind that make up a decision tree, can also be expressed in SQL, the database

access language, to retrieve or score matching records. As models, decision trees can be used for classification, estimation, and prediction. Decision trees are also useful for data exploration and variable selection even when you plan to use a different technique to create the final model. The variables chosen for inclusion in the tree are also likely to be useful for other data mining techniques.

This chapter opens with an example of a decision tree for a simple prediction task. This example highlights the way decision trees can provide insight into a business problem, and how easy decision trees are to understand. It also illustrates how you can use a tree for selecting variables, making classifications, and estimating real numbers.

The chapter continues with more technical detail on how to create decision trees. In particular, several different criteria for evaluating decision tree splitting rules are introduced and compared. There are many different decision tree algorithms, some bearing names such as CART, CHAID, and C5.0. However, these variants are all based on the same building blocks.

This chapter also presents a discussion of the difference between local and global models using a comparison of decision trees with linear regression models as a case in point. A technical aside compares decision trees with support vector machines, another technique for partitioning classes. Interesting applications of decision trees are scattered throughout the chapter.

What Is a Decision Tree and How Is It Used?

A *decision tree* is a hierarchical collection of rules that describes how to divide a large collection of records into successively smaller groups of records. With each successive division, the members of the resulting segments become more and more similar to one another with respect to the target.

This section presents examples of trees used for various purposes, including gaining insight into a business problem, exploring data (in a directed fashion), making predictions, classifying records, and estimating values.

A Typical Decision Tree

The decision tree in Figure 7-1 was created from a model set describing postpaid phone subscribers; these are subscribers who talk first and pay later. The model set is set up for a predictive model. So, the input variables are recorded for all active customers on a given date, and the target is assigned, based on the customer status 100 days later. The model set is balanced, containing equal numbers of customers who are active 100 days later, who stopped involuntarily (by not paying) and who stopped voluntarily. These three possibilities are represented by the target variable, which takes on one of the three values, **A**, **V**, or **I**.

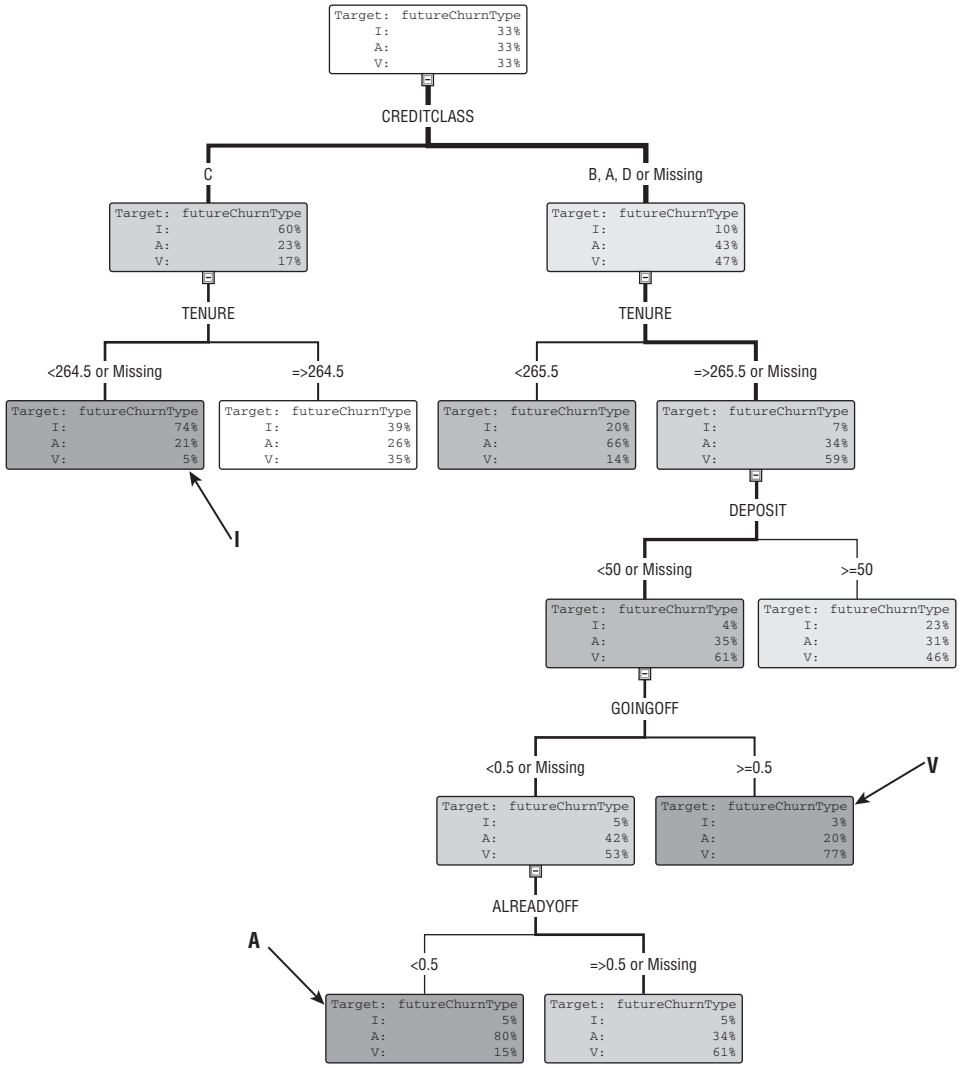


Figure 7-1: A decision tree.

The box at the top of the diagram is the *root node*, which contains all the training data used to grow the tree. In this node, all three classes are represented equally. The root node has two children, and a rule that specifies which records go to which child. The rule at the top of the tree is based on credit class: Credit class "C" goes to the left child and credit classes "A," "B," and "D" go to the right child. The point of the tree is to split these records into nodes dominated by a single class. The nodes that ultimately get used are at the ends of their branches, with no children. These are the *leaves* of the tree.

The path from the root node to a leaf describes a rule for the records in that leaf. In Figure 7-1, nodes with distributions similar to the training data are lightly shaded; nodes with distributions quite different from the training data are darker. The arrows point to three of the darkest leaves. Each of these leaves has a clear majority class.

Decision trees assign scores to new records, simply by letting each record flow through the tree to arrive at its appropriate leaf. For instance, the tree in Figure 7-1 can be used to assign an **A** score, **V** score, and I score to any currently active subscriber. Each leaf has a rule, which is based on the path through the tree. The rules are used to assign subscribers in need of scoring to the appropriate leaf. The proportion of records in each class provides the scores.

Using the Tree to Learn About Churn

In mature markets, nearly all mobile service providers are concerned about *churn*, the industry term for subscribers switching from one provider to another. In markets where telephone penetration is already high, the easiest way to acquire new subscribers is to lure them away from a competitor. The decision tree in Figure 7-1 describes who is doing the churning and which of two variants of churn is more common in particular segments. *Voluntary* churn is when the customer decides to leave. *Involuntary* churn is when the company tells customers to leave, usually because they have not been paying their bills. To create the model set, subscribers active on a particular date were observed and various attributes of each captured in a customer signature.

The first split in the tree is on credit class. Subscribers with credit class C take one path whereas those with any other credit class take another. The credit class is "A," "B," "C," or "D," with "A" meaning excellent credit and "D" the lowest credit rating. The fact that this variable is chosen first means that credit class is the most important variable for splitting the data.

This split drastically changes the distribution of the target in each of the children. Sixty percent of subscribers with credit class "C" experience involuntary churn compared to only 10 percent for all other credit classes. Subsequent splits continue to concentrate the classes. Notice that different variables are used in different parts of the tree. However, any variable can be used anywhere in the tree, and a variable can be used more than once.

In the full tree, most leaves are dominated by a single class. Each node is annotated with the percent of subscribers in each of the three classes.

Look first at the leaf marked **I**. These subscribers are credit class "C" and have tenure of 264 days or less. Seventy-four percent of them were cancelled for non-payment. The rate of voluntary cancellations is quite low because most subscribers are on a one-year contract that includes a hefty cancellation fee. Rather than paying the fee, dissatisfied subscribers whose attitude toward debt repayment has earned them credit class "C" simply walk away.

Now consider the node marked **V**. These subscribers pay no deposit (the smallest deposit is $100) and have been around for at least 265 days. Although

they were on contract at the time the inputs were recorded, they were known to be going off contract before the date when the target was recorded. The split on `deposit>=50` is exactly equivalent to a split on `credit class='D'` because everyone with credit class D pays a deposit ranging from $100 to $600, whereas people with credit class "A," "B," or "C" pay no deposit.

Finally, look at the leaf marked **A**. Like those in the node marked **V**, they have no deposit and have been around for more than 265 days. But these subscribers are still on contract and not about to go off contract. Perhaps they signed two-year contracts to start with, or perhaps they were enticed to renew a contract after the first year. In any case, 80% are still active.

Judging by this tree, contracts do a good job of retaining subscribers who are careful with their credit scores, and large deposits do a good job of retaining customers who are not. Both of these groups wait until they can leave voluntarily and without punishment. The worst attrition is among subscribers with credit class "C." They are not forced to pay a deposit, but unlike others who have no deposit, customers with credit class "C" are willing to walk away from a contract. Perhaps these customers should be asked to pay a deposit.

Using the Tree to Learn About Data and Select Variables

The decision tree in Figure 7-1 uses five variables from among the many available in the model set. The decision tree algorithm picked these five because, together, they do a good job of explaining voluntary and involuntary churn. The very first split uses `credit class`, because `credit class` does a better job of separating the target variable classes than any other available field. When faced with dozens or hundreds of unfamiliar variables, you can use a decision tree to direct your attention to a useful subset. In fact, decision trees are often used as a tool for selecting variables for use with another modeling technique. In general, decision trees do a reasonable job of selecting a small number of fairly independent variables, but because each splitting decision is made independently, it is possible for different nodes to choose correlated or even synonymous variables. An example is the inclusion of both `credit class` and `deposit` seen here.

Different choices of target variable create different decision trees containing different variables. For example, using the same data used for the tree in Figure 7-1, but changing the target variable to the binary choice of active or not active (by combining **V** and **I**) changes the tree. The new tree no longer has credit class at the top. Instead, `handset churn rate`, a variable not even in the first tree, rises to the top. This variable is consistent with domain knowledge: Customers who are dissatisfied with their mobile phone (handset) are more likely to leave. One measure of dissatisfaction is the historical rate of churn for handsets. This rate can (and should) be recalculated often because handset preferences change with the speed of fashion. People who have handsets associated with high rates of attrition in the recent past are more likely to leave.

PICKING VARIABLES FOR A HOUSEHOLD PENETRATION MODEL AT THE BOSTON GLOBE

During the data exploration phase of a directed data mining project, decision trees are a useful tool for choosing variables that are likely to be important for predicting particular targets. One of the authors' newspaper clients, the *Boston Globe*, was interested in estimating a town's expected home delivery circulation level based on various demographic and geographic characteristics. Armed with such estimates, it would be possible to spot towns with untapped potential where the actual circulation was lower than the expected circulation. The final model would be a regression equation based on a handful of variables. But which variables? The U.S. Census Bureau makes hundreds of variables available. Before building the regression model, we used decision trees to explore the possibilities.

Although the newspaper was ultimately interested in predicting the actual number of subscribing households in a given city or town, that number does not make a good target for a regression model because towns and cities vary so much in size. Wasting modeling power on discovering that there are more subscribers in large towns than in small ones is not useful. A better target is *penetration* — the proportion of households that subscribe to the paper. This number yields an estimate of the total number of subscribing households simply by multiplying it by the number of households in a town. Factoring out town size yields a target variable with values that range from 0 to somewhat less than 1.

The next step was to figure out which factors, from among the hundreds in the town signature, separate towns with high penetration (the "good" towns) from those with low penetration (the "bad" towns). Our approach was to build a decision tree with a binary good/bad target variable. This involved sorting the towns by home delivery penetration and labeling the top one-third "good" and the bottom one-third "bad." Towns in the middle third — those that are not clearly good or bad — were left out of the training set.

TIP When trying to model the difference between two groups, removing examples that are not clearly in one group or the other can be helpful.

The resulting tree used median home value as the first split. In a region with some of the most expensive housing in the country, towns where the median home value is less than $226,000 dollars are poor prospects for this paper (all census variables are from the 2000 Census). The next split was on one of a family of derived variables comparing the subscriber base in the town to the town population as a whole. Towns where the subscribers are similar to the general population are better, in terms of home delivery penetration, than towns where the subscribers are further from average. Other variables that were important for distinguishing good from bad towns included the average

years of school completed, the percentage of the population in blue collar occupations, and the percentage of the population in high-status occupations.

Some variables picked by the decision tree were less suitable for the regression model. One example is distance from Boston. The problem is that at first, as one drives out into the suburbs, home penetration goes up with distance from Boston. After a while, however, distance from Boston becomes negatively correlated with penetration as people far from Boston do not care as much about what goes on there. A decision tree easily finds the right distance to split on, but a regression model expects the relationship between distance and penetration to be the same for all distances. Home price is a better predictor because its distribution resembles that of the target variable, increasing in the first few miles and then declining. The decision tree provides guidance about which variables to think about as well as which variables to use.

Using the Tree to Produce Rankings

Decision trees score new records by looking at the input variables in each new record and following the appropriate path to the leaf. For many applications, the ordering of the scores is more important than the actual scores themselves. That is, knowing that Customer A has higher or lower churn than Customer B is more important than having an actual estimate of the churn risk for each customer. Such applications include selecting a fixed number of customers for a specific marketing campaign, such as a retention campaign. If the campaign is being designed for 10,000 customers, the purpose of the model is to find the 10,000 customers most likely to churn; determining the actual churn rate is not important.

Using the Tree to Estimate Class Probabilities

For many purposes, rankings are not sufficient, and probabilities of class membership are needed. The class probabilities are obtained from the leaves. For example, the distribution of classes in the node labeled **I** in Figure 7-1 comes from applying the rule `credit class='C' and tenure<264.5` to the balanced data at the root node. Saying that any record arriving at node **I** has probability 0.6 of churning involuntarily in the next 100 days might seem reasonable; however, the distribution of values in the original data is quite different from the distribution in the model set used to build the tree. After six months, 89.30 percent of subscribers are still active, 4.39 percent have left involuntarily, and 6.32 percent have left voluntarily.

Chapter 5 explains one way to convert scores to probability estimates. Another way to estimate the true probabilities is to apply the decision tree rules to the original, unbalanced preclassified data and observe the resulting distribution.

For this particular dataset, selecting all subscribers with `credit class='C'` and `tenure<264.5` yields a sample in which 84.14% are still active, 14.44% have left involuntarily, and 1.42% have left voluntarily. So the correct probability estimate for involuntary churn at this leaf is 14 percent rather than 60 percent. The percentage of involuntary churn in this leaf is well over three times the level in the subscriber population, but even here, "active" is still the most probable outcome by far.

Using the Tree to Classify Records

To use the tree as a classifier, all that is required is to estimate the class probabilities as described earlier and label each leaf with its most probable class. This is a use of decision trees that is often presented as primary in the academic literature. In the marketing world, the class probability estimates are usually more useful than the classification because classifiers quite commonly produce only one outcome. Classifying everyone as nonresponders is not helpful because the point of creating models is to differentiate among records.

A model that puts everyone in the same class is neither surprising nor uncommon in marketing applications where the behaviors of interest (response, fraud, attrition, and so on) tend to be rare. No matter how the segments for a marketing campaign are defined, the most likely outcome in any segment is no response. Fortunately, some segments are more likely to respond than others and that is enough to be useful. A charity does not send you an appeal for donations because they think you will respond; they reach out to you because they think the chance of your responding, while low, is high enough to justify the postage.

Using the Tree to Estimate Numeric Values

A tree used to estimate the value of a numeric target variable (as opposed to the probability of class membership) is referred to as a *regression tree*. The tree is built so that records in any given leaf all have target values close to the average for that leaf — or in the language of statistics, the goal is to minimize the variance of the target values in each leaf. The leaf average is the score assigned to any new record that matches the rule for the leaf.

A regression tree can only generate as many distinct values as there are leaves in the tree. Using a discrete model such as a regression tree to estimate a continuous value may seem odd while using a continuous function to estimate continuous values seems more natural; however, regression trees can be used in other ways, such as selecting the variables for a regression model. Also, regression trees do a good job of breaking up the original data into local segments. Building a different model in each segment using a technique such as regression can also produce effective models. The following section discusses this idea of finding local phenomena in the data.

Decision Trees Are Local Models

The discussion of regression models in the previous chapter points out that regression models are global and, as a consequence, regression models do not do a good job of fitting data that has local characteristics. Trying to fit local phenomena in the input space changes the values of the model everywhere. Decision trees, on the other hand, are local models; they carve the input space into segments and produce a separate estimate for each one.

Figure 7-2 shows a tree that uses only two input variables, `days since last purchase` and `lifetime orders` to predict `order size`, a numeric variable. This tree has 12 leaves and a depth of 4. Figure 7-3 shows the same model as a rectangular box divided into more boxes. Each sub-box represents a leaf in the tree. Notice that the boxes fill the entire rectangle. Just as every record reaches some leaf, every record lands in a box. The shading of the boxes represents the average order size for records landing there; darker boxes have higher average order sizes.

The splits themselves are the vertical and horizontal lines in this box. Each split on `lifetime orders` corresponds to one of the vertical line segments. Each split on `days since last purchase` corresponds to one of the horizontal line segments. The longest vertical line, right down the middle of the figure, is the highest split in the tree.

This example only uses two dimensions, because this is easier to show in a diagram. Introducing another variable would introduce another dimension. Then, the data would be represented as a brick shape (technically, a rectangular polyhedron), and it would be broken up into small bricks, by planes cutting it into pieces.

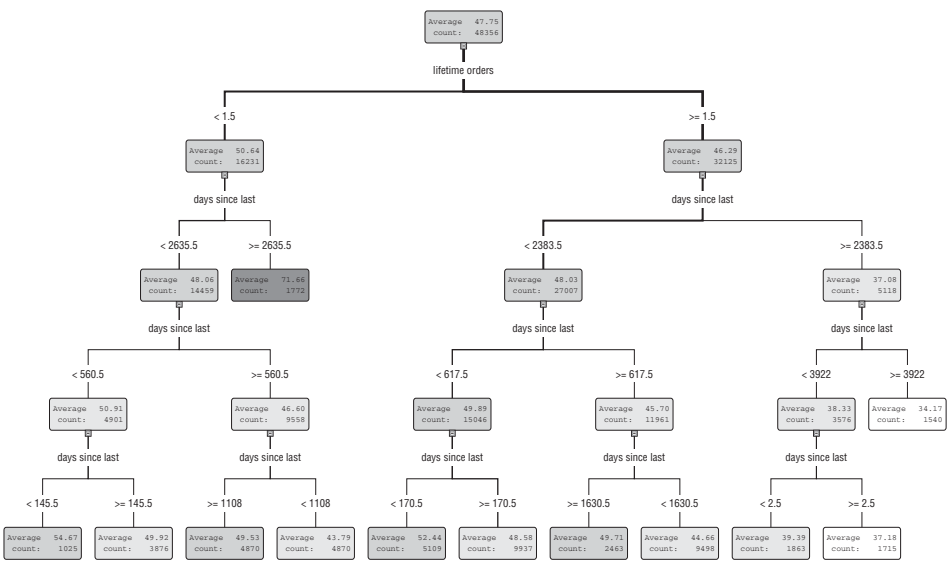


Figure 7-2: A regression tree for average order size as a function of recency and frequency.

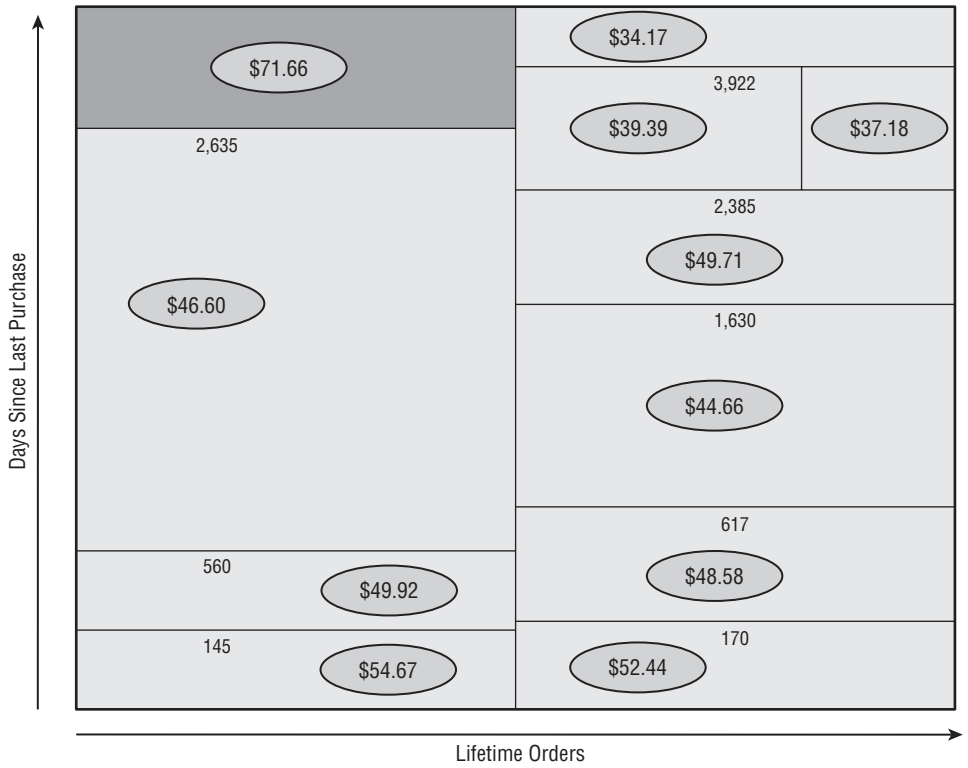


Figure 7-3: The tree puts the records into rectangular boxes.

The box diagram makes it easier to see an interesting pattern that is hard to spot in the tree. For the most part, customers who have made an order recently have larger order sizes. However, a point exists where the correlation between recency and order size changes direction. Over most of the range, average order size increases with recency. In other words, it is inversely correlated with `days since last purchase`. But surprisingly, the very highest average order size is in the box in the upper-left corner — customers whose sole purchase was more than seven years ago. (As an aside, it seems a bit odd that the company does not purge "customers" who have not made a purchase in seven years from its house file. Perhaps they do purge most such customers, but spare those who have spent more than $50, which could explain why the long-lost customers still on file were all such big spenders.) Whatever its cause, the change in sign of the correlation means that a regression model might not make good use of the recency variable. Regression models assume the relationship between an input and the target is the same everywhere.

Because the decision tree model is local, it is fine for the relationship between recency and order size to be quite different in different leaves. If new customers were to come in and make very large purchases, they would have no effect on the box in the upper left, which only contains customers whose most recent purchase was long ago. Conversely, one could alter the training data to give customers in

the upper-left box an average purchase of $100 or even $1,000 without affecting the average order size in any other box. Another strength of the regression tree model is that, because the estimated order sizes associated with each box are averages of actual observed values, they can never be too unreasonable. This is in contrast with a regression model, which may predict negative order sizes or other values outside the range of what has ever been seen.

The traditional tree diagram is a very effective way of representing the actual structure of a decision tree, but for some purposes, box diagrams like the one in Figure 7-3 can be more expressive. A box diagram brings all leaves, no matter how many levels down in the tree, to the surface where they are easy to compare. For example, Figure 7-3 shows at a glance that the top-left corner contains the biggest spenders. One way of thinking about decision trees is that they are a way of drawing boxes around groups of similar records. All the records within a particular box are classified the same way because they all meet the rule defining that box. This differs from global classification methods such as logistic regression, and more recent inventions such as support vector machines, all of which attempt to partition data into classes by drawing a single line or curve or hyperplane through the data space. This is a fundamental distinction: Global models are weak when there are several very different ways for a record to become part of the target class.

In the credit card industry, for instance, there are several ways for customers to be profitable. Some profitable customers, called *revolvers*, have low transaction rates, but keep high revolving balances without defaulting. *Transactors*, on the other hand, pay off their balance in full each month, but are profitable due to the high transaction volume they generate. Yet others, called *convenience users*, have few transactions, but occasionally make a large purchase and take several months to pay it off. Two very dissimilar customers may be equally profitable. A decision tree can find each separate group, label it, and by providing a description of each box, suggest the reason for each group's profitability.

Growing Decision Trees

Although there are many variations on the core decision tree algorithm, all of them share the same basic method for creating the tree: Click on the decision tree icon in your tool of choice or call the appropriate decision tree procedure. But what happens inside? This section dives into more detail, because understanding the detail helps you use decision trees more effectively and makes understanding their results easier.

The decision tree algorithm repeatedly splits the data into smaller and smaller groups in such a way that each new set of nodes has greater purity than its ancestors with respect to the target variable. For the most part, this discussion assumes a binary, categorical target variable, such as responder/nonresponder. This assumption simplifies the explanations without much loss of generality.

Finding the Initial Split

At the start of the process, there is a model set consisting of preclassified records — that is, the value of the target variable is known for all cases. The goal is to build a tree that uses the values of the input fields to create rules that result in leaves that do a good job of assigning a target value to each record. For a binary target, this value is the probability of membership in each class. Remember that each record in the model set starts with a known target, and this guides the construction of the tree.

The tree starts with all the records in a subset of the model set — the training set — at the root node. The first task is to split the records into children by creating a rule on the input variables. What are the best children? The answer is the ones that are purest in one of the target values, because the goal is to separate the values of the target as much as possible.

To perform the split, the algorithm considers all possible splits on all input variables. For instance, for `days since last purchase`, the tree considers splitting at 100 days, with customers having 0–100 days going into one child and the rest going to the other child. It considers splits at 1 day and at 1,000 days, and at all other distinct values found in the training set. The algorithm evaluates the splits, and chooses the best split value for each variable. The best variable is the one that produces the best split.

The measure used to evaluate a potential split is *purity* of the target variable in the children. Low purity means that the distribution of the target in the children is similar to that of the parent node, whereas high purity means that members of a single class predominate. The best split is the one that increases purity in the children by the greatest amount. A good split also creates nodes of similar size, or at least does not create nodes containing very few records.

These ideas are easy to see visually. Figure 7-4 illustrates some good and bad splits. In this case, the original data consists of nine circles and nine triangles and the goal is to separate these two groups. The first split is a poor one because no increase in purity exists. The initial population contains equal numbers of the two shapes; after the split, so does each child. The second split is also poor, because although purity is increased slightly, the pure node has few members and the purity of the larger child is only marginally better than that of the parent. The final split is a good one because it leads to children of roughly the same size and with much higher purity than the parent. Visualizing purity is perhaps easier than formalizing the concept so it can be calculated by a computer; the next section discusses several different ways to calculate purity, including measures for both categorical and numeric targets.

Splits are evaluated based on how pure the resulting children are in the target variable. This means that the choice of an appropriate splitting criterion depends on the type of the target variable, not on the type of the input variable. Numeric targets have different definitions of purity from categorical and binary targets.

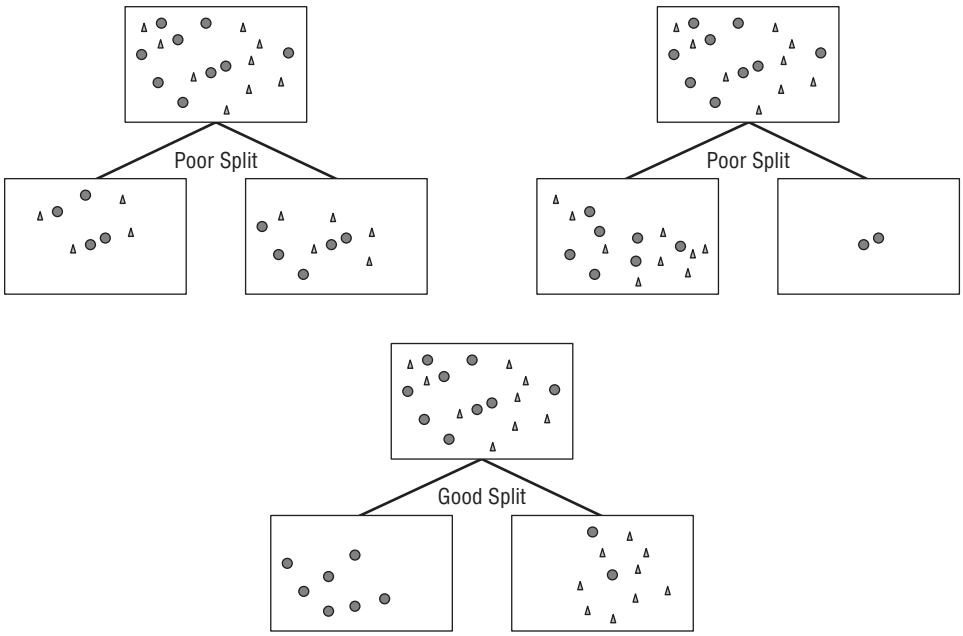


Figure 7-4: A good split increases purity for all the children.

Splitting on a Numeric Input Variable

When searching for a binary split on a numeric input variable, distinct each value that the variable takes on in the training set is treated as a candidate value for the split. Splits on a numeric variable take the form $X<N$. All records where the value of X (the splitting variable) is less than some constant N are sent to one child and all records where the value of X is greater than or equal to N are sent to the other. After each trial split, the increase in purity due to the split is measured. In the interests of efficiency, some implementations of decision trees use a representative sample of the values instead of evaluating every one.

When the decision tree is scored, the only use it makes of numeric inputs is to compare their values with the split points. They are never multiplied by weights or added together as they are in many other types of models. This has the important consequence that decision trees are not sensitive to outliers or skewed distributions of numeric variables.

Splitting on a Categorical Input Variable

The simplest algorithm for splitting on a categorical input variable is to create a new branch for each class that the categorical variable can take on. So, a split on the contract start month would yield twelve children, one for each calendar

month. This approach has been used in some software implementations, but it often yields poor results. High branching factors quickly reduce the population of training records available at each child node, making further splitting less likely and less reliable.

A better and more common approach is to group together classes that, taken individually, predict similar outcomes. For a binary target, the simplest approach is to determine the proportion of the target for each value of the input variable. Then, all values that have a smaller proportion of one target value than the parent node go to one child, and the rest go to the other child.

A more sophisticated approach looks at the distributions of the target within each value of the input variable, and combines values whose distributions are very similar. The usual test for whether the distributions differ significantly is the chi-square test explained in Chapter 4.

Splitting in the Presence of Missing Values

One of the nicest things about decision trees is their ability to handle missing values in input fields by using null as an allowable value. This approach is preferable to throwing out records with missing values or trying to impute missing values. Throwing out records is likely to create a biased training set because the records with missing values are probably not a random sample of the population. Replacing missing values with imputed values runs the risk that important information provided by the fact that a value is missing will be ignored in the model.

The authors have seen many cases where the fact that a particular value is null has predictive value. In one such case, the count of non-null values in appended household-level demographic data was predictive of the response to an offer of term life insurance. Apparently, people who leave many traces in Acxiom's household database (by buying houses, getting married, having babies, registering products, subscribing to magazines, and so on) are more likely to be interested in life insurance than those whose lifestyles leave more fields null.

WARNING **Decision trees can produce splits based on missing values of an input variable. The fact that a value is null can often have predictive value so do not be hasty to filter out records with missing values or to try to replace them with imputed values.**

An alternative approach to missing values, which is part of the CART algorithm and available in several software implementations, keeps several splitting rules for each node. These *surrogate splits* use different fields to produce similar results. When a null value is encountered in the field that yields the best split, the next best rule can be used. The use of surrogate splits is more interesting

in theory than in practice because often, when the first variable is missing the surrogates are missing as well. For instance, if the first variable is from census data and describes something about the neighborhood, the surrogates probably also describe the neighborhood. And, if the first variable is missing, it is because the census information is not available for the customer's address, so similar variables are also missing.

Growing the Full Tree

The initial split produces two or more children, each of which is then split in the same manner as the root node. This is called a *recursive* algorithm, because the same splitting method is used on the subsets of data in each child. Once again, all input fields are considered as candidate splitters, even fields already used for splits. Eventually, the tree building stops, for one of three reasons:

- No split can be found that significantly increases the purity of any node's children.
- The number of records per node reaches some preset lower bound.
- The depth of the tree reaches some preset limit. At this point, the full decision tree has been grown.

If a completely deterministic relationship existed between the input variables and the target, this recursive splitting would eventually yield a tree with completely pure leaves. Manufacturing examples of this sort is easy, but they do not occur very often in marketing or CRM applications. Customer behavior data almost never contains clear, deterministic relationships between inputs and outputs. The fact that two customers have the exact same description in terms of the available input variables does not ensure that they will exhibit the same behavior. A decision tree for a catalog response model might include a leaf representing females with age greater than 50, three or more purchases within the last year, and total lifetime spending of more than $145. The customers reaching this leaf will typically be a mix of responders and nonresponders. If the leaf in question is labeled "responder," then the proportion of nonresponders is the *misclassification rate* for this leaf.

One circumstance where deterministic rules *are* likely to be discovered is when patterns in data reflect business rules. The authors had this fact driven home to them while analyzing warranty claims at Caterpillar, a manufacturer of diesel engines. We built a decision tree model to predict which claims would be approved. At the time, the company had a policy of paying certain claims automatically. The results were startling: The decision tree had some leaves that were 100 percent accurate on unseen test data. In other words, the tree had discovered the exact rules used to classify the claims. Of course, discovering known business rules may not be particularly useful; it does, however, underline

the effectiveness of decision trees on rule-oriented problems. On this same problem, a neural network also produced good results, but it could not explain the patterns that it found. The decision tree could identify a business rule that explained why the model was doing so well.

Finding the Best Split

Many different criteria may be used to evaluate potential splits. Alternate splitting criteria often lead to trees that look quite different from one another, but have similar performance. That is because there are usually many candidate splits with very similar performance. Different purity measures select different splits, but because all the measures are trying to capture the same idea, the resulting models tend to behave similarly.

Figure 7-5 shows a good split. The parent node contains 9 circles and 9 triangles. The left child contains 7 circles and 1 triangle. The right child contains 8 triangles and 2 circles. Clearly, the purity has increased, but how can the increase be quantified? And how can this split be compared to others? That requires a formal definition of purity, several of which are listed next.

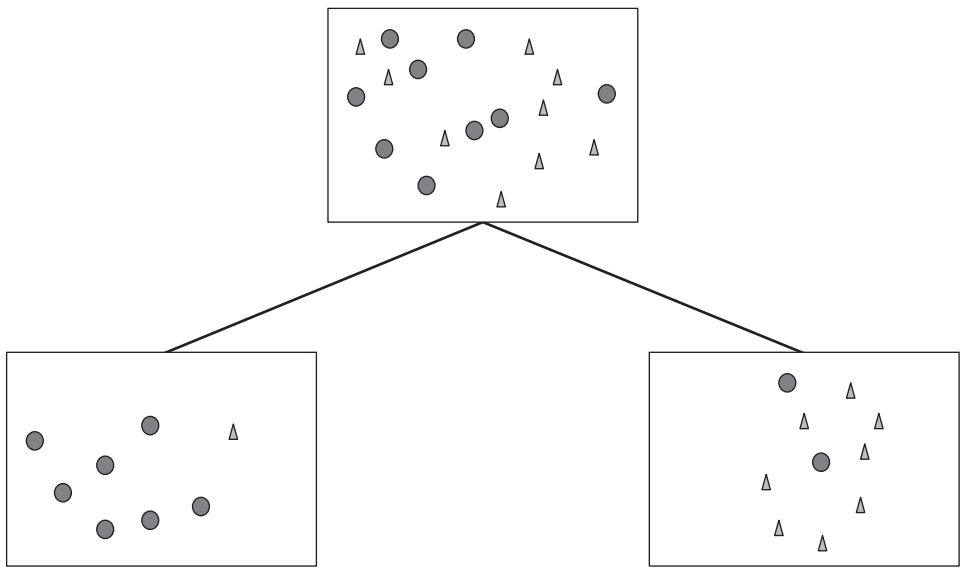


Figure 7-5: A good split on a binary categorical variable increases purity.

Purity measures for evaluating splits for categorical target variables include:

- Gini (also called population diversity)
- Entropy (also called information gain)

- Chi-square test
- Incremental response

When the target variable is numeric, one approach is to bin the value and use one of the preceding measures. However, two measures are in common use for numeric targets:

- Reduction in variance
- F test

Note that the choice of an appropriate purity measure depends on whether the *target* variable is categorical or numeric. The type of the input variable does not matter. The split illustrated in Figure 7-5 could just as easily be provided by a numeric input variable (`AGE > 46`) or by a categorical variable (`STATE` is a member of CT, MA, ME, NH, RI, VT).

Gini (Population Diversity) as a Splitting Criterion

One popular splitting criterion is named Gini, after the 20th century Italian statistician and economist, Corrado Gini. This measure, which is also used by biologists and ecologists studying population diversity, gives the probability that two items chosen at random from the same population are in the same class.

As an example, consider an ecosystem that has exactly two animals — wily coyotes and roadrunners. The question is: How pure is this ecosystem? The approach for answering this question is the following: Two ecologists go into the ecosystem and each takes a picture of an animal. Purity is then the probability that these two pictures are of the same type of animal. For a pure population, the probability is 1, because the pictures will always be of that animal. For a population that is half wily coyotes and half roadrunners, the probability is 0.5. This probability is the Gini score.

For the Gini measure, a score of 0.5 means that two classes are represented equally. When a node has only one class, its score is 1. Because purer nodes have higher scores, the goal of decision tree algorithms that use this measure is to maximize the Gini score of the split.

The Gini measure of a node is easy to calculate. It is simply the sum of the squares of the proportions of the classes in the node. For the split shown in Figure 7-5, the parent population has an equal number of circles and triangles. A node with equal numbers of each of two classes has a score of $P(\text{circle})^2+P(\text{triangle})^2 = 0.5^2 + 0.5^2 = 0.5$, which is expected because the chance of picking the same class twice by random selection with replacement is 1 out of 2. The Gini score for the left child is $0.125^2 + 0.875^2 = 0.781$. The Gini score for the right child is $0.200^2 + 0.800^2 = 0.680$.

To calculate the impact of a split, take the average of the Gini scores of the children, weighted by the size of each child. In this case, 0.444*0.875 + 0.556*0.680 = 0.725.

As shown in Figure 7-6, the Gini score varies between 0.5 and 1. A perfectly pure node has a Gini score of 1. A node that is evenly balanced has a Gini score of 0.5. Sometimes, the score is manipulated so it is in the range from 0 to 1 (doubling the score and subtracting the result from 1). However, such a manipulation makes no difference when comparing different scores to optimize purity.

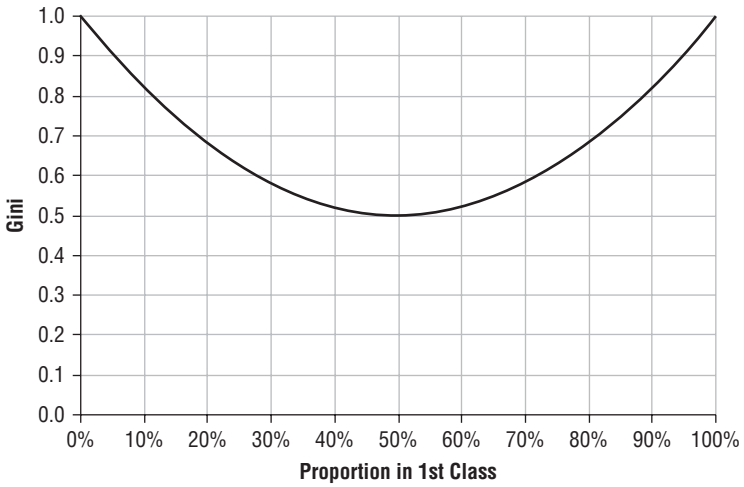


Figure 7-6: For a binary target, the Gini score varies from 0.5 when there is an equal number of each class to 1 when all records are in the same class.

Entropy Reduction or Information Gain as a Splitting Criterion

Information gain uses a clever idea for defining purity, borrowed from the world of machine learning. If a leaf is entirely pure, then the classes in the leaf can be described very easily — there is only one. On the other hand, if a leaf is highly impure, then describing it is much more complicated. Information theory has a measure for this called *entropy*, which measures how disorganized a system is. A comprehensive introduction to information theory is far beyond the scope of this book. For this book's purposes, the intuitive notion is that the number of bits required to describe a particular outcome depends on the number of possible outcomes. You can think of entropy as a measure of the number of yes/no questions it would take to determine the state of the system. If there are 16 possible states, it takes $\log_2(16)$, or four bits, to enumerate them or identify a particular one. Additional information reduces the number of questions needed

to determine the state of the system, so information gain means the same thing as entropy reduction. Both terms are used to describe decision tree algorithms.

The entropy of a particular decision tree node can be readily calculated using a formula. The entropy for a node is the sum, for all the target values in the node, of the proportion of records with a particular value multiplied by the base two logarithm of that proportion. (In practice, this sum is usually multiplied by –1 in order to obtain a positive number, because logarithms of probabilities are negative.) Despite the logarithms, this formula is quite similar to the formula for the Gini score; the Gini score multiplies each target value's proportion by itself while the entropy score multiplies each target value's proportion by its logarithm.

As shown in Figure 7-7, an entropy score of 1 means that two classes are represented equally. When a node has only one class, its score is 0. So, purer nodes have lower scores, and the goal is to minimize the entropy score of the split.

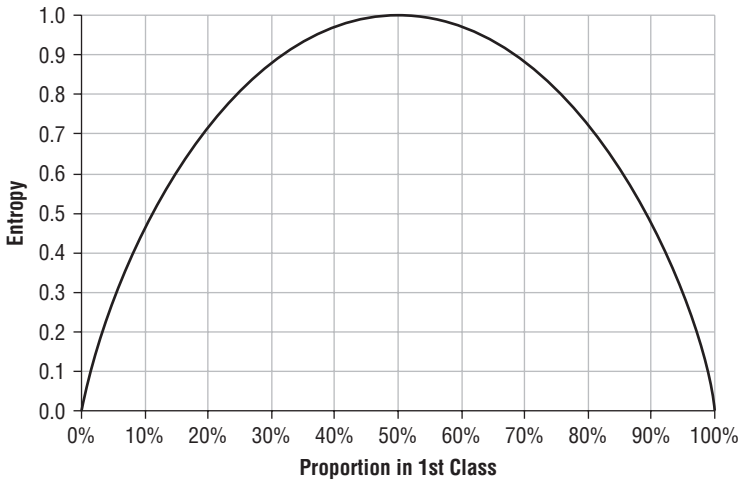


Figure 7-7: Entropy goes from 0 for a pure population to 1 when there is an equal number of each class.

The entropy of a split is calculated in the same way as the Gini score: It is simply the weighted average of the entropies of all the children. When entropy reduction is the splitting criterion, the decision tree algorithm selects the split that reduces entropy by the greatest amount.

For a binary target variable such as the one shown in Figure 7-5, the formula for the entropy of a single node is:

$$-1 * (P(\text{circle})\log_2 P(\text{circle}) + P(\text{triangle})\log_2 P(\text{triangle}))$$

In this example, for the left child, *P*(circle) is 7 out of 8 and *P*(triangle) is 1 out of 8. Plugging these numbers into the entropy formula yields:

$$-1 * (0.875 \log_2(0.875) + 0.125 \log_2(0.125)) = 0.544$$

The first term is for the circles and the second term is for the triangles. For the right child, *P*(circle) is 2 out of 10 and *P*(triangle) is 8 out of 10. Plugging these into the entropy formula yields:

$$-1 * (0.200 \log_2(0.200) + 0.800 \log_2(0.800)) = 0.722$$

To calculate the total entropy of the system after the split, multiply the entropy of each node by the proportion of records that reach that node and add them up. In this example, 8 of 18 records are in the left child, and 10 of 18 in the right. The total entropy reduction or information gain due to the split works out to 0.643. This is the figure that would be used to compare this split with other candidates.

Information Gain Ratio

Measures of purity can run into trouble when the splitting methodology allows more than two splits. This was the case for ID3, a decision tree tool developed by Australian researcher J. Ross Quinlan in the 1980s that became part of several commercial data mining software packages. And, it was a particularly severe problem for ID3, because it placed each category into a separate child, resulting in nodes with many children. However, the same problem arises whenever a tree considers splits with different numbers of children. Just by breaking the larger data set into many small subsets, the number of classes represented in each node tends to go down, so each child increases in purity, even for a random split.

Professor Quinlan used entropy for ID3. The decrease in entropy due solely to the number of branches is called the *intrinsic information* of a split. For a random n-way split, the probability of each branch is `1/n`. Therefore, the entropy due solely to splitting from an n-way split is `n * 1/n log (1/n)` or `log(1/n)` (and a similar calculation would apply for Gini). Because of the intrinsic information of many-way splits, decision trees built using the entropy reduction splitting criterion without any correction for the intrinsic information due to the split tend to prefer many splits at a node. Bushy trees with many multiway splits are undesirable because these splits lead to small numbers of records in each node — a recipe for unstable models.

In reaction to this problem, C5.0 and other descendants of ID3 that once used information gain now use the ratio of the total information gain due to a proposed split to the intrinsic information attributable solely to the number of branches created as the criterion for evaluating proposed splits. This test reduces the tendency towards very bushy trees that was a problem in earlier decision tree software packages.

Chi-Square Test as a Splitting Criterion

As described in Chapter 4, the chi-square test is a test of statistical significance developed by the English statistician Karl Pearson in 1900. The chi-square value

measures how likely or unlikely a split is. The higher the chi-square value, the less likely the split is due to chance — and not being due to chance means that the split is important.

Calculating the chi-square value relies on a simple formula. For a child node, the chi-square value is the sum of the squares of the differences between the *expected* and *observed* frequencies of each value of the target, divided by the expected frequency. The chi-square value of a split is simply the sum of the chi-square values of all the children — not the weighted average as with Gini and entropy. In common with other significance tests, it is a measure of the probability that an observed difference between samples could occur just by chance. When used to measure the purity of decision tree splits, higher values of chi-square mean that the variation is more significant, and not due merely to chance.

For example, suppose the target variable is a binary flag indicating whether or not customers continued their subscriptions at the end of the introductory offer period and the proposed split is on `acquisition channel`, a categorical variable with three classes: direct mail, outbound call, and e-mail. If the acquisition channel had no effect on renewal rate, the expected number of renewals in each class would be proportional to the number of customers acquired through that channel.

Each proposed split can be evaluated according to the following table:

Table 7-1: Contingency Table for Split Evaluation

	RESPONSE = 0	RESPONSE = 1
Left Child	# of 0s on left	# of 1s on left
Right Child	# of 0s on right	# of 1 on right

In a contingency table, such as this, any given record is counted exactly once. The chi-square value measures the probability that the contingency table could be due to chance. The idea is that a split due to chance is not interesting — some other split is more useful. This is measured by looking at the proportions of the target in the children. When they have similar proportions to their parent, then the split is probably due to chance and hence uninteresting. On the other hand, if the distribution of the response in the children differs from that of the parents, a very low probability exists that the split is due to chance and the split is likely to be useful.

Calculating the chi-square value only requires a bit of arithmetic. For each cell in the table, the chi-square test calculates the expected number of 0s and 1s. The chi-square value for each cell is calculated by subtracting the expected value from the observed value, squaring the result, and dividing by the expected

number. The overall chi-square is the sum of all the cell chi-square contributions. As described in Chapter 4, the chi-square distribution provides a way to translate this chi-square score to a probability, although this is not necessary when used for decision trees. To measure the purity of a split in a decision tree, the score is sufficient. A high chi-square score means that the proposed split successfully splits the population into subpopulations with significantly different distributions.

Unlike the Gini and entropy measures, the chi-square value does not have a restricted range such as 0 to 1; it grows larger as the amount of data grows. Figure 7-8 graphs the chi-square value for a sample of 100 records, taken from a parent population that has equal numbers in each of two classes, as the number of elements in the first class ranges from 0 to 100.

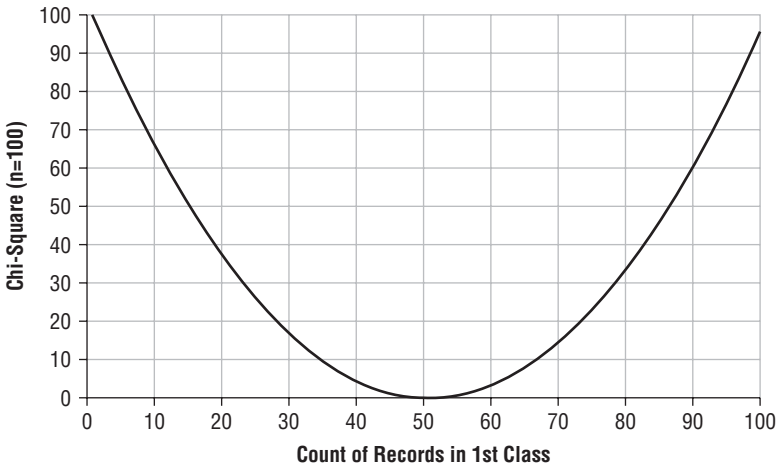


Figure 7-8: Chi-square is 0 when the sample distribution is the same as the population's.

The chi-square test gives its name to CHAID, a well-known decision tree algorithm first published by John A. Hartigan in 1975 and improved upon by Kass in 1980. The full acronym stands for Chi-square Automatic Interaction Detector. CHAID makes use of the chi-square test in several ways — first to merge classes that do not have significantly different effects on the target variable, then to choose a best split, and finally to decide whether it is worth performing any additional splits on a node.

Incremental Response as a Splitting Criterion

The splitting criteria described so far depend only on the target variable. This section discusses a different measure based on the idea of incremental response introduced in Chapter 5. When working with incremental response, there is

a test group and a control group as well as responders and non-responders. Incremental response models attempt to isolate *persuadables* — the people most likely to be persuaded by an offer, rather than those who would respond anyway.

Modeling incremental response is different from modeling response. Response can be measured at the level of individuals, but incremental response cannot. A person either responds to an offer or not, but there is no way to measure how someone who was included in a marketing campaign would have behaved had they been left out.

In Chapter 5, the approach is to build two different response models — one for probability of response given treatment (the test group) and one for probability of response given no treatment (the control group). The incremental response is the difference between these two scores. Portrait Software has a clever alternative approach to modeling incremental response. The Portrait Software Uplift Optimizer™ builds a decision tree using the difference in response between the treated group and the control group as the splitting criterion using training records that have both a target variable, such as response or lack of response to an offer, and a field indicating whether the record is from the treatment group or the control group. The best split is the one that maximizes the difference in response between the two groups. The leaves of the resulting tree identify segments that are highly persuadable and segments that are not. There may even be segments that respond better when left untreated. The sidebar describes an application of this approach.

Reduction in Variance as a Splitting Criterion for Numeric Targets

The four previous splitting criteria all apply to categorical targets. When the target variable is numeric, one way to measure a good split is that such a split should reduce the variance of the target variable. Recall that variance is a measure of the extent to which values in a sample stay close to the average value. In a sample with low variance, most values are quite close to the average; in a sample with high variance, many values are quite far from the average.

Although the reduction in variance purity measure is meant for numeric targets, the circles and triangles shown earlier in Figure 7-5 can still be used to illustrate it by considering the circles to be 1 and the triangles to be 0. The average value in the parent node is clearly 0.5. Every one of the 18 observations differs from the mean by 0.5, so the variance is $(18 * 0.5^2) / 18 = 0.25$. After the split, the left child has seven circles and one triangle, so the node mean is 0.875. Seven of the observations differ from the mean value by 0.125 and one observation differs from the mean value by 0.875. In the right child, the two circles and 8 triangles have an average value of 0.2. The eight triangles differ from the average by 0.2 and the two triangles differ from the average by 0.8. So, the variance across both children is $(0.875^2 + 7 * 0.125^2 + 8*0.2^2 + 2*0.8^2) / 18 = 0.138$. The reduction in variance due to the split is $0.25 - 0.138 = 0.112$.

U.S. BANK IMPROVES INCREMENTAL RESPONSE

Headquartered in Minneapolis, U.S. Bank is one of the ten largest banks in the United States with 2,850 branches serving 15.8 million customers. They are true believers in the value of incremental response modeling to capture the true return on marketing dollars; consequently, their product managers' bonuses are based on it.

Back in the 1990s, they did what many of their competitors still do today — they built response models and hoped that high lift for the response model would yield incremental lift as well. Often, it did not. After a campaign, the people most likely to respond were not any *more* likely to respond than they were already. In the years since then, U.S. Bank has tried several approaches to modeling incremental response.

The Difference Model

The first approach taken by U.S. Bank was the one described in Chapter 5. Customers were scored using one model trained on response given inclusion in the campaign and another trained on a control group not included in the campaign. Customers were ranked on the difference between these scores. The difficulty with this approach is that the standard error in the difference of two scores is higher than the standard error of either of the scores alone, and the individual scores may themselves have high standard errors when based on small samples. If constructing good models were the only goal of the campaign, the solution would be to use large, equal-sized treatment and control groups. Unfortunately for modelers, the campaign's primary goal is to reach persuadable customers. Even in an enlightened company such as U.S. Bank, the modelers have to negotiate with marketing managers to get any control group at all for the top deciles and to be allowed to have any members of the lower deciles included in the campaign. As a result, the model for response given no treatment is starved for data in the top deciles and the model for response given treatment is starved for data in the low deciles.

The Matrix Model

U.S. Bank tried a cell-based approach. In cell-based models, customers are placed into segments according to which decile they fall in for each of the variables defining the cells. In a cell-based incremental response model, randomly selected members of each cell are included in a test campaign. The difference in response between those included and those excluded is recorded for each cell, and the cells with the greatest difference in response rate are targeted for the full campaign. U.S. Bank calls this approach the "matrix model." It served as the baseline "champion" for comparison with newer "challenger" models.

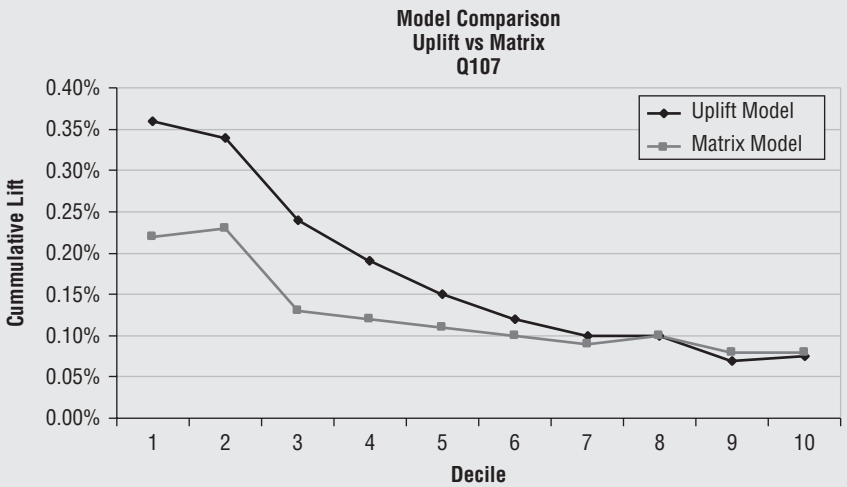
One challenge in constructing a cell-based model is deciding which variables should be used to define the cells. U.S. Bank's approach to that problem

was essentially a manual decision tree. They tried binning each candidate variable in turn to find the one that produced the greatest difference in response between the treatment group and the control group. The two or three best variables by this measure were used to create the cells.

The Portrait Uplift Optimizer

U.S. Bank eventually settled on a software package that creates decision trees with splits based on incremental lift. They made a head-to-head comparison of the new tool with their home-grown matrix model on a campaign to cross-sell home equity lines of credit to existing customers. It was a bit like John Henry against the steam drill — the matrix model and manual cells were heroic efforts, but in the end they were beaten by modern technology. The Uplift Optimizer from Portrait Software uses a decision tree to choose the best variables based on incremental response. Like any decision tree, it finds the optimal points to split numeric variables rather than using arbitrary boundaries, such as deciles. It deals with the small sample problem, caused by the desire to not miss many persuadables, by using *bagging* and *boosting*. Bagging means combining the votes of several different models trained on the same data. Boosting takes this idea further, by building a second model trained on the examples that were misclassified by the first, and perhaps a third model trained on the mistakes of the second.

The following chart shows the challenger bested the champion, especially on the first three deciles — the ones slated to receive the mailing.



Champion-Challenger comparison.

F Test

Another split criterion that can be used for numeric target variables is the F test, named for another famous Englishman — the statistician, astronomer, and geneticist, Ronald. A. Fisher. Fisher and Pearson reportedly did not get along despite, or perhaps because of, the large overlap in their areas of interest. Fisher's test does for continuous variables what Pearson's chi-square test does for categorical variables. It provides a measure of the probability that samples with different means and variances are actually drawn from the same population.

A well-understood relationship exists between the variance of a sample and the variance of the population from which it was drawn. (In fact, as long as the samples are of reasonable size and randomly drawn from the population, sample variance is a good estimate of population variance; very small samples — with fewer than 30 or so observations — usually have higher variance than their corresponding populations.) The F test looks at the relationship between two estimates of the population variance — one derived by pooling all the samples and calculating the variance of the combined sample, and one derived from the between-sample variance calculated as the variance of the sample means. If the various samples are randomly drawn from the same population, these two estimates should agree closely.

The F score is the ratio of the two estimates. It is calculated by dividing the between-sample estimate by the pooled sample estimate. The larger the score, the less likely it is that the samples are all randomly drawn from the same population. In the decision tree context, a large F score indicates that a proposed split has successfully split the population into subpopulations with significantly different distributions.

Pruning

The basic algorithm for decision trees keeps growing the tree by splitting nodes as long as new splits create children that increase purity. Such a tree has been optimized for the training set, so eliminating any leaves would only increase the error rate of the tree on the training set. Does this imply that the full tree also does the best job on new data? Certainly not!

A decision tree algorithm makes its best split first, at the root node where there is a large number of records. As the nodes get smaller, idiosyncrasies of the particular training records at a node come to dominate the process. The smaller the nodes become, the greater the danger of overfitting. One way to avoid overfitting is to set a large minimum leaf size. Another approach is to allow the tree to grow as long as there are splits that appear to be significant on the training data and then eliminate the splits that prove to be unstable

by cutting away leaves through a process called *pruning*. Three approaches to pruning are discussed next. These are not the only possible pruning strategies, but the first two covered here are commonly implemented, and the third one ought to be.

The CART Pruning Algorithm

CART (Classification and Regression Trees) is a popular decision tree algorithm first published by Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone in 1984. The CART algorithm grows binary trees and continues splitting as long as new splits can be found that increase purity. As illustrated in Figure 7-9, inside a complex tree are many simpler subtrees, each of which represents a different trade-off between model complexity and accuracy. The CART algorithm identifies a set of such subtrees as candidate models. These candidate subtrees are applied to the validation set, and the tree with the lowest validation set misclassification rate (or average squared error for a numeric target) is selected as the final model.

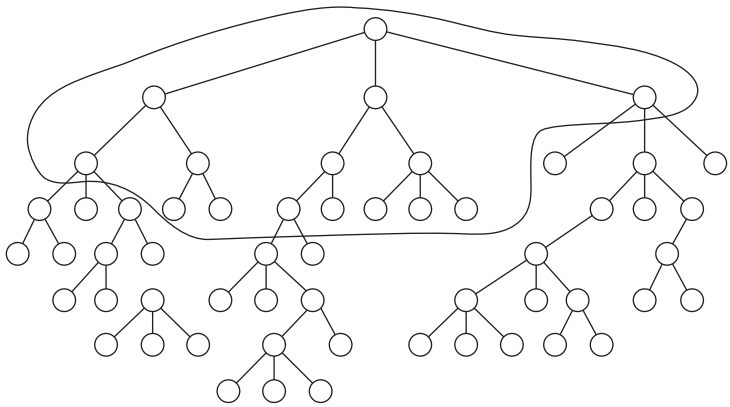


Figure 7-9: Inside a complex tree are simpler, more stable trees.

Creating Candidate Subtrees

The CART algorithm identifies candidate subtrees through a process of repeated pruning. The goal is to prune first those branches providing the least additional predictive power per leaf. To identify these least useful branches, CART relies on a concept called the *adjusted error rate*. This is a measure that increases each node's misclassification rate or mean squared error on the training set by imposing a complexity penalty based on the number of leaves in the tree. The adjusted error is used to identify weak branches (those whose error is not low enough to overcome the penalty) and mark them for pruning.

The formula for the adjusted error rate is:

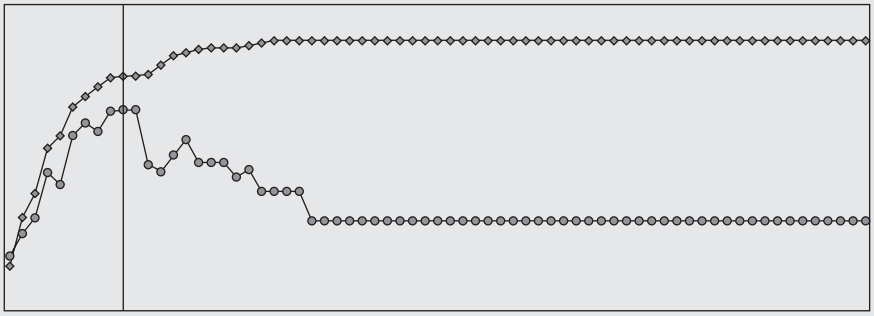
$$AE(T) = E(T) + \alpha leaf\_count(T)$$

Where α is an adjustment factor that is increased in gradual steps to create new subtrees. When α is 0, the adjusted error rate equals the error rate. The algorithm continues to find trees by adjusting α and pruning back one node at a time, creating a sequence of trees, α_1, α_2, and so on, each with fewer and fewer leaves. The process ends when the tree has been pruned all the way down to the root node. Each of the resulting subtrees (sometimes called the *alphas*) is a candidate to be the final model. Notice that all the candidates contain the root node and the largest candidate is the entire tree.

COMPARING MISCLASSIFICATION RATES ON TRAINING AND VALIDATION SETS

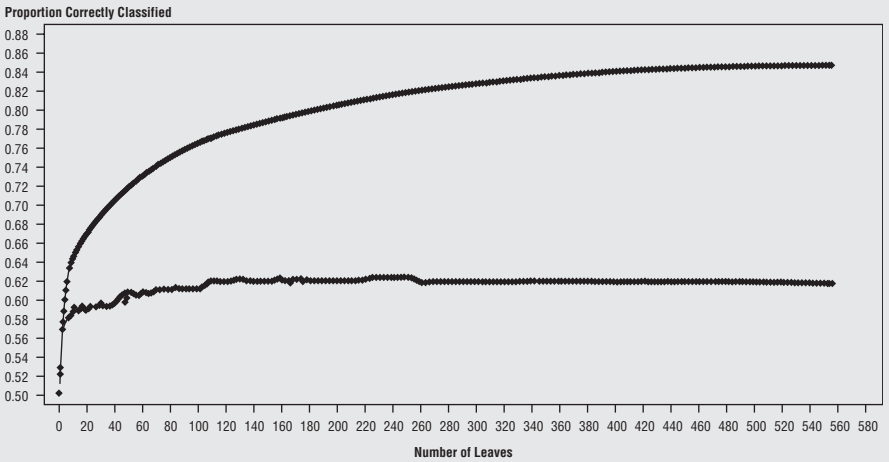
The error rate on the validation set should be larger than the error rate on the training set, because the training set was used to build the rules in the model. A large difference in the misclassification error rate, however, is a symptom of an unstable model. This difference can show up in several ways as shown by the following three graphs. The graphs represent the percent of records correctly classified by the candidate models in a decision tree. Candidate subtrees with fewer nodes are on the left; those with more nodes are on the right.

As expected, the first chart shows the candidate trees performing better and better on the training set as the trees have more and more nodes — the training process stops when the performance no longer improves. On the validation set, however, the candidate trees reach a peak and then the performance starts to decline as the trees get larger. The optimal tree is the one that works best on the validation set, and the choice is easy because the peak is well-defined.



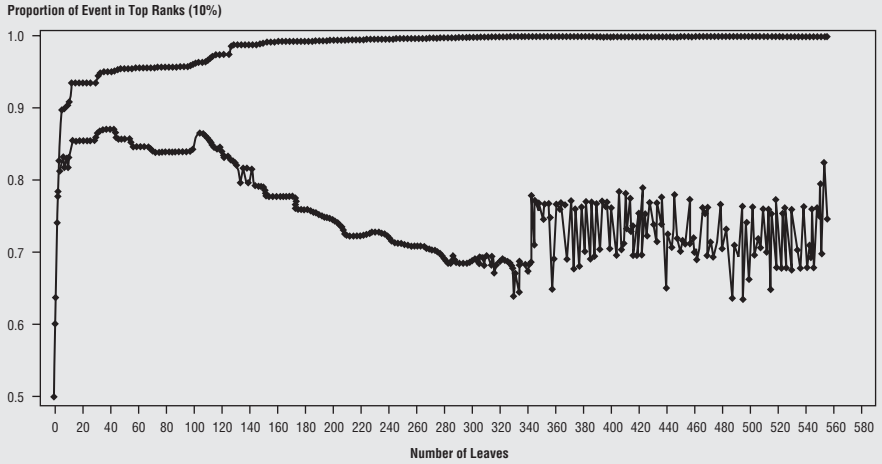
This chart shows a clear inflection point in the graph of the percent correctly classified in the validation set.

Sometimes, though, there is no clear demarcation point. That is, the performance of the candidate models on the validation set never quite reaches a maximum as the trees get larger. In this case, the pruning algorithm chooses the entire tree (the largest possible subtree), as shown.



In this chart, the percent correctly classified in the validation set levels off early and remains far below the percent correctly classified in the training set.

The final example is perhaps the most interesting, because the results on the validation set become unstable as the candidate trees become larger. The cause of the instability is that the leaves are too small. In this tree, there is an example of a leaf that has three records from the training set and all three have a target value of 1 — a perfect leaf. However, in the validation set, the one record that falls there has the value 0. The leaf is 100 percent wrong. As the tree grows more complex, more of these too-small leaves are included, resulting in the instability shown:



In this chart, the percent correctly classified on the validation set decreases with the complexity of the tree and eventually becomes chaotic.

The last two figures are examples of unstable models. The simplest way to avoid instability of this sort is to ensure that leaves are not allowed to become too small.

Picking the Best Subtree

The next step is to select, from the pool of candidate subtrees, the one that works best on new data. That, of course, is the purpose of the validation set. Each of the candidate subtrees is used to classify the records or estimate values in the validation set. The tree that performs this task with the lowest overall error is declared the winner. The winning subtree has been pruned sufficiently to remove the effects of overtraining, but not so much as to lose valuable information. The graph in Figure 7-10 illustrates the effect of pruning on classification accuracy.

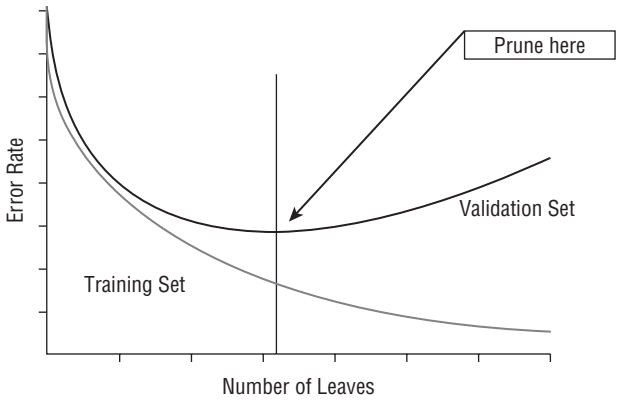


Figure 7-10: Pruning chooses the tree whose miscalculation rate is minimized on the validation set.

The winning subtree is selected on the basis of its overall error when applied to the validation set. But, while one expects that the selected subtree will continue to be the best model when applied to other datasets, the error rate that caused it to be selected may slightly overstate its effectiveness. There may be many subtrees that all perform about as well as the one selected. To a certain extent, the one of these that delivered the lowest error rate on the validation set may simply have "gotten lucky" with that particular collection of records. For that reason, as explained in Chapter 5, the selected subtree is applied to a third preclassified dataset, the test set. The error obtained on the test set is used to predict expected performance of the model when applied to unclassified data.

WARNING **Do not evaluate the performance of a model by its lift or error rate on the validation set or the training set. Both have had a hand in creating the model and so overstate the model's accuracy. Instead, measure the model's accuracy on a test set drawn from the same population as the training and validation sets, but not used in any way to create the model.**

Because this pruning algorithm is based solely on the misclassification rate, without taking the probability of each classification into account, it replaces any subtree whose leaves all make the same classification with a common parent that also makes that classification. In applications where the goal is to select a small proportion of the records (the top 1 percent or 10 percent, for example), this pruning algorithm may hurt the performance of the tree, because it may remove leaves that contain a very high proportion of the target class.

WARNING **Pruning sometimes removes leaves that should be kept or fails to remove leaves that should be trimmed. The CART pruning algorithm removes children that result in the same classification even when one child is much purer than the other. For some applications, such as when good lift in the highest decile is more important than the overall error rate, retaining such splits to preserve the rule associated with the purer child is preferable. Other pruning algorithms leave some nodes where the distribution of the target differs significantly from the training set to the validation set. It is therefore sometimes advisable to prune by hand.**

Pessimistic Pruning: The C5.0 Pruning Algorithm

C5.0 is a more recent version of the decision-tree algorithm that J. Ross Quinlan has been evolving and refining for many years. An earlier version, ID3, published in 1986, was very influential in the field of machine learning and its successors are used in several commercial data mining products.

The trees grown by C5.0 are similar to those grown by CART (although unlike CART, C5.0 makes multiway splits on categorical variables). Like CART, the C5.0 algorithm first grows an overfit tree and then prunes it back to create a more stable model. The pruning strategy is quite different because C5.0 does not make use of a validation set to choose from among candidate subtrees. The same data used to grow the tree is also used to decide how the tree should be pruned. This may reflect the algorithm's origins in the academic world, where in the past, university researchers had a hard time getting their hands on substantial quantities of real data to use for training sets. Consequently, they spent much time and effort trying to coax the last few drops of information from their impoverished datasets — a problem that data miners in the business world do not face.

C5.0 prunes the tree by measuring each node's error on the training data and assuming that the error on unseen data would be substantially worse. The algorithm treats the data as if it resulted from a series of trials, each of which can have one of two possible results. (Heads or tails is the usual example.) As it happens,

mathematicians have been studying this particular situation since at least 1713, the year that Jacob Bernoulli's famous binomial formula was posthumously published. So, well-known formulas exist for determining what it means to have observed *E* occurrences of some event, such as the number of errors, in *N* trials.

In particular, a formula exists which, for a given confidence level and size of a node, gives the confidence interval — the range of expected numbers of errors. C5.0 assumes that the observed number of errors on the training data is the low end of this range; it then calculates the corresponding number of errors at the high end of the range. When the high-end estimate of the error at a node is less than the estimate for the error of its children, the children are pruned.

Stability-Based Pruning

The pruning algorithms used by CART and C5.0 (and indeed by all the commercial decision tree tools that the authors have used) have a problem. They fail to prune some nodes that are clearly unstable. The split highlighted in Figure 7-11 is a good example. The numbers on the left side of each node show what is happening on the training set. The numbers on the right side of each node show what is happening on the validation set. This particular tree is meant to identify churners. When only the training data is taken into consideration, the highlighted branch seems to do very well; the concentration of churners rises from 58.0 percent to 70.9 percent. Unfortunately, when the very same rule is applied to the validation set, the concentration of churners actually *decreases* from 56.6 percent to 52 percent.

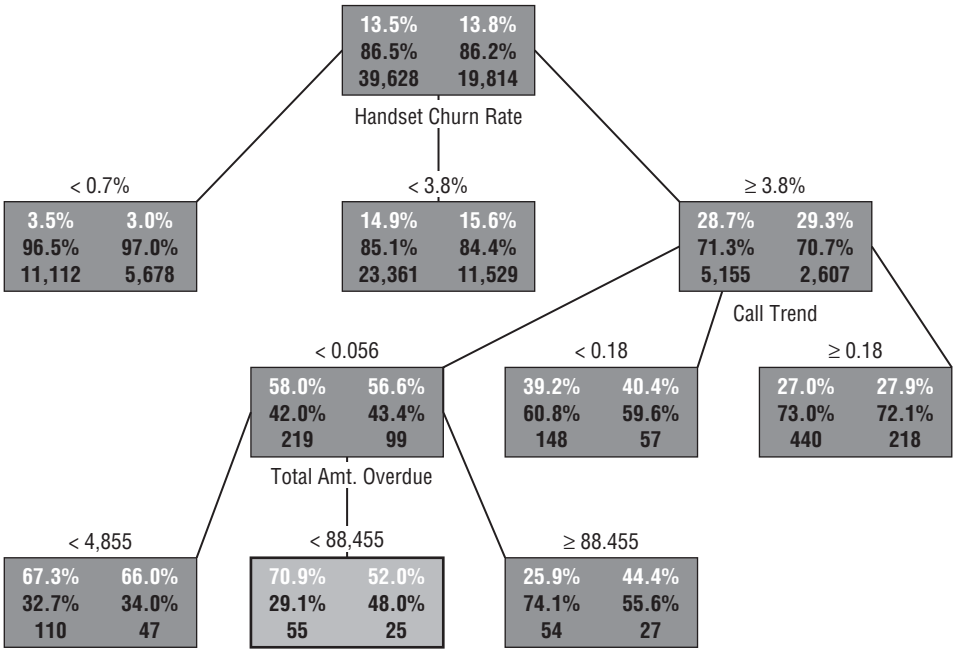


Figure 7-11: An unstable split produces very different distributions on the training and validation sets.

Stable models make consistent predictions on previously unseen records. Any rule that cannot achieve that goal should be eliminated from the model. Many data mining tools allow the user to prune a decision tree manually. This facility is useful, but the authors look forward to data mining software that provides automatic stability-based pruning as an option. Such software would need to have a less subjective criterion for rejecting a split than "the distribution of the validation set results looks different from the distribution of the training set results." A test of statistical significance, such as the chi-square test would be used. The split would be pruned when the confidence level is less than some user-defined threshold, so only splits that are, say, 95 percent confident on the validation set would remain.

WARNING **Small nodes cause big problems. A common cause of unstable decision tree models is allowing nodes with too few records. Most decision tree tools allow the user to set a minimum node size. As a general rule, nodes that receive fewer than about 100 training set records are likely to be unstable.**

Extracting Rules from Trees

When a decision tree is used primarily to generate scores, one can easily forget that a decision tree is actually a collection of rules. If one of the purposes of the data mining effort is to gain understanding of the problem domain, reducing the huge tangle of rules in a decision tree to a smaller, more comprehensible collection can be useful.

Other situations exist where the desired output is a set of rules. In *Mastering Data Mining* (Wiley 2000), the authors describe the application of decision trees to an industrial process improvement problem, namely the prevention of a certain type of printing defect. In that case, the end product of the data mining project was a small collection of simple rules that could be posted on the wall next to each press.

When a decision tree is used for producing scores, having a large number of leaves is advantageous because each leaf generates a different score. When the object is to generate rules, fewer rules may be better. Fortunately, it is often possible to collapse a complex tree into a smaller set of rules.

As a first step, any subtree where all leaves have the same label can be replaced by its parent node without changing the way anything is classified. C5.0 includes a rule generator that goes further; it is willing to sacrifice some classification accuracy to reduce the number of rules. It does so by removing clauses, then comparing the predicted error rate of the new, briefer rule to that of the original using the same pessimistic error rate assumption described earlier in the pessimistic pruning section. Often, the rules for several different leaves generalize to the same rule, so this process results in fewer rules than the decision tree had leaves.

In the decision tree, every record ends up at exactly one leaf, so every record has a definitive classification. After the rule-generalization process, however, there may be rules that are not mutually exclusive and records that are not covered by any rule. Simply picking one rule when more than one is applicable can solve the first problem. The second problem requires the introduction of a default class assigned to any record not covered by any of the rules. Typically, the most frequently occurring class is chosen as the default.

After it has created a set of generalized rules, C5.0 groups the rules for each class and eliminates those that do not contribute much to the overall accuracy of the rule set. The end result is a small number of hopefully easy-to-understand rules.

Decision Tree Variations

Oak, ash, maple, birch, pine, spruce — real trees come in many varieties and so do decision trees. In addition to named algorithms such as CART, CHAID, and C5.0, there are countless other ways of combining split evaluation criteria, pruning strategies, and other algorithmic choices. This section introduces a few of the many variations.

Multiway Splits

So far, all the trees in the chapter have nodes with exactly two children. In such trees, each node represents a yes-or-no question, whose answer determines by which of two paths a record proceeds to the next level of the tree. Because any multiway split can be expressed as a series of binary splits, there is no real need for trees with higher branching factors. Nevertheless, many data mining tools are capable of producing trees with more than two branches. For example, some decision tree algorithms split on categorical variables by creating a branch for each class, leading to trees with differing numbers of branches at different nodes. Figure 7-12 shows a tree that includes a five-way split on `tenure` and both two-way and three-way splits on `credit class`. This tree was built on the same data and the same table as used earlier in Figure 7-1. The tree with two-way splits performed better than the tree with five-way splits when applied to a test data set. This is probably because after the five-way split on tenure, no further splits were found for most tenure ranges.

WARNING **No relationship exists between the number of branches allowed at a node and the number of classes in the target variable. A binary tree (that is, one with two-way splits) can be used to classify records into any number of categories, and a tree with multiway splits can be used to classify a binary target variable.**

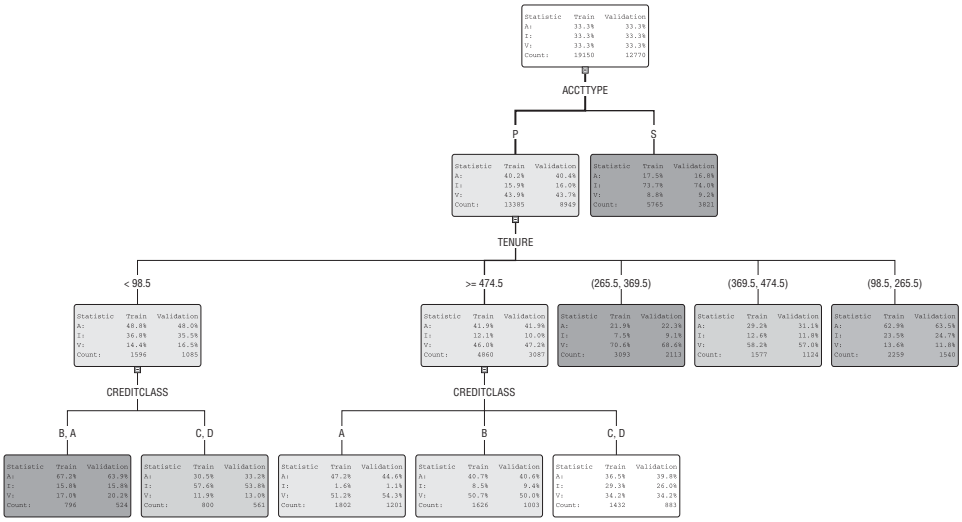


Figure 7-12: This tree with multiway splits does not perform as well as the binary tree in Figure 7-1.

Splitting on More Than One Field at a Time

Most decision tree algorithms test a single variable to perform each split. This approach can be problematic for several reasons, not least of which is that it can lead to trees with more nodes than necessary. Extra nodes are cause for concern because only the training records that arrive at a given node are available for inducing the subtree below it. The fewer training examples per node, the less stable the resulting model.

Suppose that you are interested in a condition for which both age and gender are important indicators. If the root node split is on age, then each child node contains only about half the women. If the initial split is on gender, then each child node contains only about half the old folks.

Several algorithms have been developed to allow multiple attributes to be used in combination to form the splitter. One technique forms Boolean conjunctions of features to reduce the complexity of the tree. After finding the feature that forms the best split, the algorithm looks for the feature which, when combined with the feature chosen first, does the best job of improving the split. Features continue to be added as long as they cause a statistically significant improvement in the resulting split.

Creating Nonrectangular Boxes

Classification problems are sometimes presented in geometric terms. This way of thinking is especially natural for datasets having continuous variables for all

fields. In this interpretation, each record is a point in a multidimensional space. Each field represents the position of the record along one axis of the space. Decision trees are a way of carving the space into regions, each of which can be labeled with a class. Any new record that falls into one of the regions is classified accordingly.

Traditional decision trees, which test the value of a single field at each node, can only form rectangular regions. In a two-dimensional space, a test of the form Y less than some constant forms a region bounded by a line perpendicular to the Y axis and parallel to the X axis. Different values for the constant cause the line to move up and down, but the line remains horizontal. Similarly, in a space of higher dimensionality, a test on a single field defines a hyperplane that is perpendicular to the axis represented by the field used in the test and parallel to all the other axes. In a two-dimensional space, with only horizontal and vertical lines to work with, the resulting regions are rectangular. In three-dimensional space, the corresponding shapes are rectangular solids, and in any multidimensional space, they are hyper-rectangles.

The problem is that some things don't fit neatly into rectangular boxes. Figure 7-13 illustrates the problem: The two regions are really divided by a diagonal line; it takes a deep tree to generate enough rectangles to approximate it adequately.

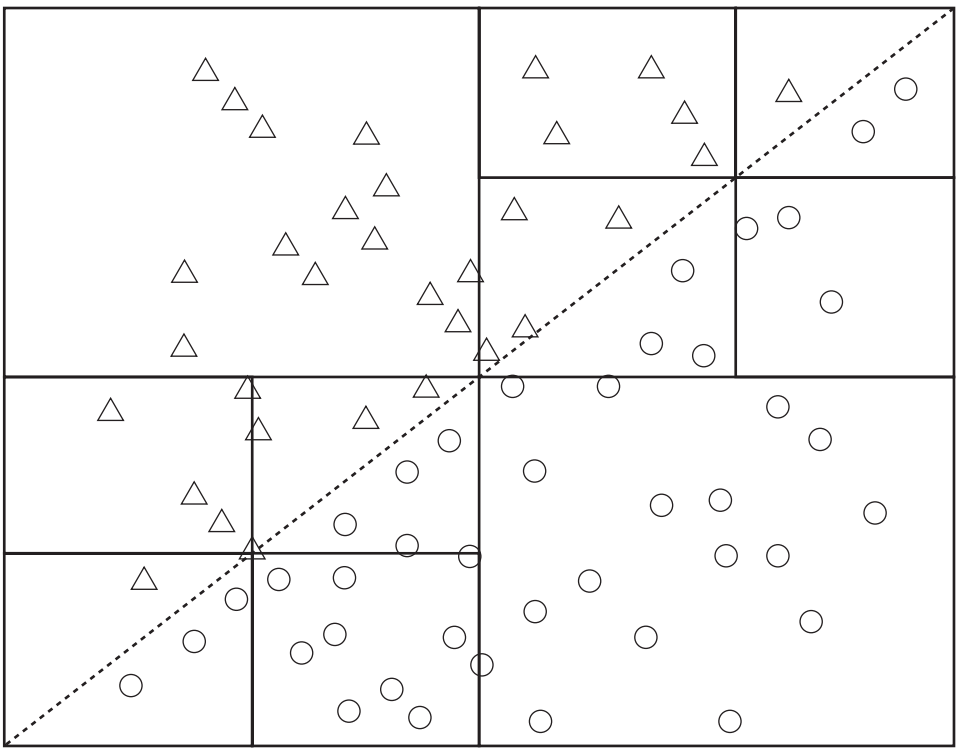
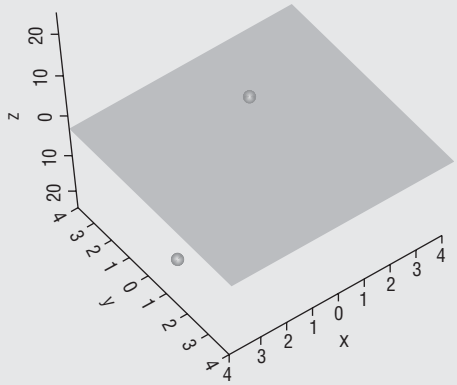


Figure 7-13: The upper-left and lower-right quadrants are easily classified, whereas the other two quadrants must be carved into many small boxes to approximate the boundary between regions.

In this case, the true solution can be found easily by allowing linear combinations of the attributes to be considered. Some software packages attempt to tilt the hyperplanes by basing their splits on a weighted sum of the values of the fields. A variety of hill-climbing approaches exist for optimizing the weights. This process of searching for a hyperplane that separates two classes is shared with another data mining technique that has generated a lot of excitement in academic circles, *support vector machines*. Although support vector machines have been around since 1995, they have been slow to catch on with practitioners in the business world. The sidebar provides a brief introduction.

SUPPORT VECTOR MACHINES

Support vector machines are a geometric method of separating two classes (responders and nonresponders, for example) by finding the best hyperplane that puts one class above it and the other below. In the very unlikely case that the two classes happen to be entirely separable by a line, this is easy. The following figure shows such a case in three dimensions. A hyperplane that separates the classes is called a *decision surface*. The figure shows a two-dimensional decision surface separating points in three dimensions.



A two-dimensional plane separating points in three dimensional space.

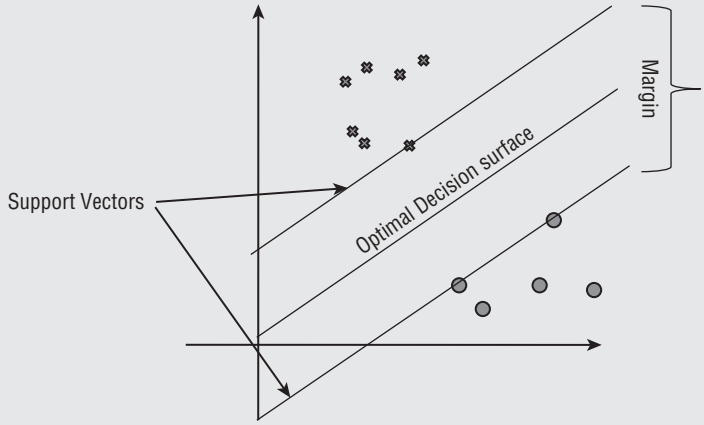
The decision surface has one fewer dimensions than the data it separates. The split at a decision tree node is an example of the simplest case. Because decision trees split on only one field at a time, the decision "surface" is a single point. Every value on one side of the point goes one way, and every value on the other side goes the other way.

The next figure shows two classes (represented by the noughts and crosses of tic-tac-toe) on a two-dimensional plane. The line separating them is the optimal decision surface. It is optimal because it maximizes the distance from the decision surface to the boundaries of the two classes. The boundary lines are called *supporting hyperplanes* and the data points on these boundaries

Continued

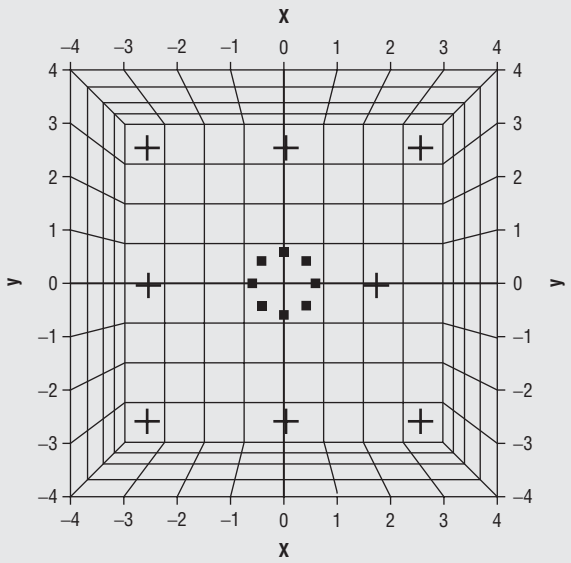
SUPPORT VECTOR MACHINES (*continued*)

are the *support vectors* that give the technique its name. The distance from one supporting hyperplane to the other is called the *margin*. The support vector machine algorithm finds the decision surface that maximizes the margin.



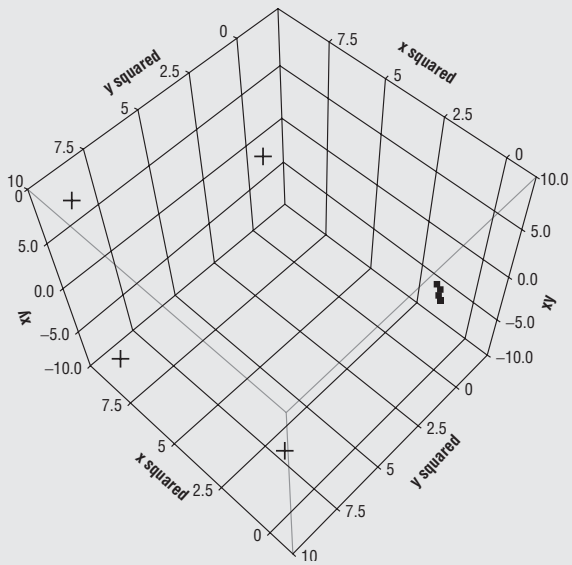
A one-dimensional line separating points on a two-dimensional plane.

The data points shown in the figure above were carefully chosen to make the idea seem easy. Real-life data is unlikely to be so kind. The following figure shows a more usual and difficult situation in two dimensions. Here, even though visually a clear boundary exists between the classes, there is obviously no straight line that can separate the points on this two-dimensional plane.



On the plane, boundary between the two classes is not a straight line.

The central insight of support vector machines is to think of the observed two-dimensional data as a projection onto the two-dimensional plane of points that really exist in three dimensions. Suppose you could come up with a transformation on the input variables from two dimensions to three that caused all the crosses to float up above the noughts. Finding a decision surface to separate the classes along the new z axis would then be easy. Such a function is called a *kernel function*. In the following figure, a kernel function has been applied that maps every point (x,y) to (x^2,y^2,xy). After this transformation, finding a decision surface is easy. The hard part is choosing the kernel function and coming up with the right parameters for it. This is an optimization task much like finding the right weights for a neural network.



After application of the kernel function, the two classes are easily separated.

Assessing the Quality of a Decision Tree

The effectiveness of a decision tree, taken as a whole, is determined by applying it to the test set — a collection of records not used to build the tree — and observing the percentage classified correctly by a classification tree, or a measure such as the average squared error for a regression tree. This provides an error measure for the tree as a whole, but it is also important to pay attention to the quality of the individual branches of the tree.

At each node, you can measure:

- The number of records entering the node
- The proportion of records in each class or the average value of the target variable
- How those records would be scored if this were a leaf node
- The percentage of records classified correctly or the average squared error
- The difference in distribution between the training set and the test set

Each path through the tree represents a rule, and some rules are better than others.

In the discussion so far, error has been the sole measure for evaluating the fitness of rules and subtrees. In many applications, however, the costs of misclassification vary from class to class. Certainly, in a medical diagnosis, a false negative can be more harmful than a false positive; a scary Pap smear result that, on further investigation, proves to have been a false positive, is much preferable to an undetected cancer. A cost function multiplies the probability of misclassification by a weight indicating the cost of that misclassification. Several tools allow the use of such a cost function instead of a purity measure for building decision trees.

When Are Decision Trees Appropriate?

There is often a trade-off between model accuracy and model transparency. In some applications, the accuracy of a classification or prediction is the only thing that matters; if a direct mail firm obtains a model that can accurately predict which members of a prospect pool are most likely to respond to a certain solicitation, the firm may not care how or why the model works. In other situations, the ability to explain the reason for a decision is crucial. In insurance underwriting, for example, there are legal prohibitions against discrimination based on certain variables. An insurance company could find itself in the position of having to demonstrate to a regulator or court of law that it has not used illegal discriminatory practices in granting or denying coverage. Similarly, hearing that an application for credit has been denied on the basis of a computer-generated rule (such as income below some threshold and number of existing revolving accounts greater than some other threshold) is more acceptable to both the loan officer and the credit applicant than hearing that the application has been rejected for unexplained reasons.

Decision trees have been used in some very imaginative ways. The final case study in this chapter describes how decision trees were used to simulate the operation of an industrial coffee roasting plant. The case study is based on

discussions with Marc Goodman and on his 1995 doctoral dissertation. The simulation can be run to project the values of all variables into the future, to be sure that the roasting process stays within acceptable bounds to ensure quality. One of the most interesting things about the case study is that a simulator requires building a separate model for the next value of each input, so variables are both inputs and targets, albeit for different models.

Case Study: Process Control in a Coffee Roasting Plant

Nestlé, one of the largest food and beverages companies in the world, uses a number of continuous-feed coffee roasters to produce a variety of coffee products. Each of these products has a "recipe" that specifies target values for a plethora of roaster variables such as the temperature of the air at various exhaust points, the speed of various fans, the rate that gas is burned, the amount of water introduced to quench the beans, and the positions of various flaps and valves. There are a lot of ways for things to go wrong when roasting coffee, ranging from a roast coming out too light in color to a costly and damaging roaster fire. A bad batch of roasted coffee wastes the beans and incurs a cost; damage to equipment is even more expensive.

To help operators keep the roaster running properly, data is collected from about 60 sensors. Every 30 seconds, this data, along with control information, is written to a log and made available to operators in the form of graphs. The project described here took place at a Nestlé research laboratory in York, England. Nestlé built a coffee roaster simulation based on the sensor logs.

Goals for the Simulator

Nestlé saw several ways that a coffee roaster simulator could improve its processes:

- By using the simulator to try out new recipes, a large number of new recipes could be evaluated without interrupting production. Furthermore, recipes that might lead to roaster fires or other damage could be eliminated in advance.
- The simulator could be used to train new operators and expose them to routine problems and their solutions. Using the simulator, operators could try out different approaches to resolving a problem.
- The simulator could track the operation of the actual roaster and project it several minutes into the future. When the simulation ran into a problem, an alert could be generated while the operators still had time to avert trouble.

Fortunately, Nestlé was already collecting data at half-minute intervals, which could be used to build the simulator.

Building a Roaster Simulation

A model set of 34,000 cases was created from the historical log data. Each case consisted of a set of measurements on the roaster along with the same measurements 30 seconds later. Notice that the same data might be used as targets for one case, and then, for the next case, might be the inputs (where the targets come 30 seconds later).

This training set is more complicated than the training sets we have been working with, because multiple targets exist — all the measurements 30 seconds later. The solution is to build a separate model for each measurement. Each model takes the input from the earlier part of the case, and the target from the later period, as shown in Figure 7-14:

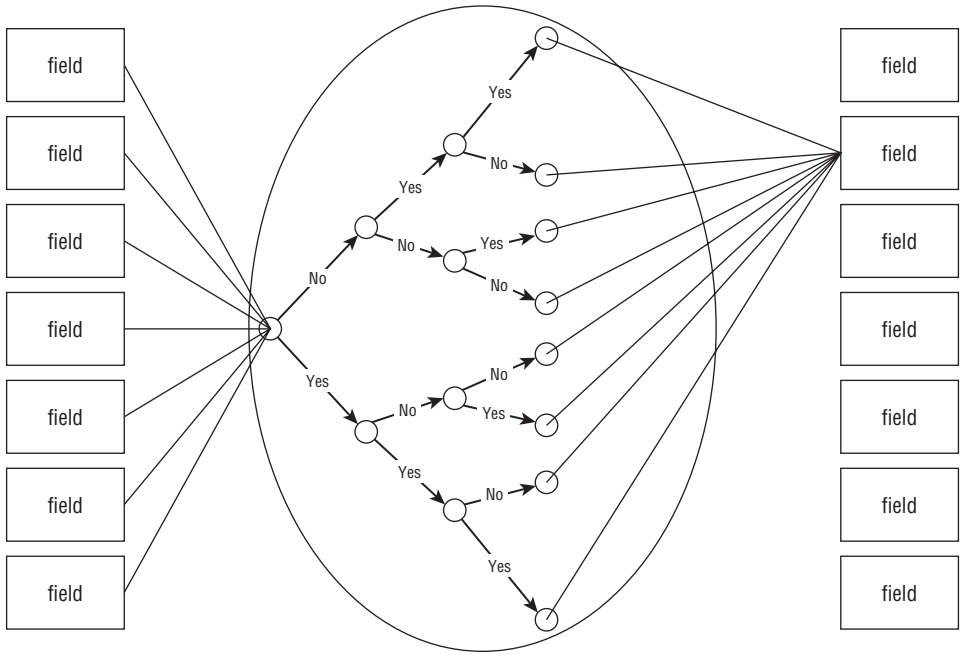


Figure 7-14: A decision tree uses values from one snapshot to create the next snapshot in time.

The entire set of models was trained, resulting in a set of models that takes the input measurements for the roaster and produces estimates of what happens 30 seconds later.

Evaluation of the Roaster Simulation

The simulation was then evaluated using a test set of around 40,000 additional cases that had not been part of the training set. For each case in the test set, the simulator generated projected snapshots 60 steps into the future (that is, 30 minutes into the future). At each step the projected values of all variables

were compared against the actual values. As expected, the size of the error increases with time. For example, the error rate for product temperature turned out to be 2/3°C per minute of projection, but even 30 minutes into the future the simulator was considerably better than random guessing.

The roaster simulator turned out to be more accurate than all but the most experienced operators at projecting trends, and even the most experienced operators were able to do a better job with the aid of the simulator. Operators enjoyed using the simulator and reported that it gave them new insight into corrective actions.

Lessons Learned

Decision-tree methods have wide applicability for data exploration, classification, and selecting important variables. They can also be used for estimating continuous values although they are rarely the first choice because decision trees generate "lumpy" estimates — all records reaching the same leaf are assigned the same estimated value. They are a good choice when the data mining task is classification of records or prediction of discrete outcomes. Use decision trees when your goal is to assign each record to one of a few broad categories.

Decision trees are also a natural choice when the goal is to generate understandable and explainable rules. The ability of decision trees to generate rules that can be translated into comprehensible natural language or SQL is one of the greatest strengths of the technique. Even in complex decision trees, following any one path through the tree to a particular leaf is generally fairly easy, so the explanation for any particular classification or prediction is relatively straightforward.

Decision trees are grown using a recursive algorithm that evaluates all values of all inputs to find the split that causes the greatest increase in purity in the children. The same thing happens again inside each child. The process continues until no more splits can be found or some other limit is reached. The tree is then pruned to remove unstable branches. Several tests are used as splitting criteria, including the chi-square test for categorical targets and the F test for numeric targets.

Decision trees require less data preparation than many other techniques because they are equally adept at handling continuous and categorical variables. Categorical variables, which pose problems for neural networks and statistical techniques, are split by forming groups of classes. Continuous variables are split by dividing their range of values. Because decision trees do not make use of the actual values of numeric variables, they are not sensitive to outliers and skewed distributions. Missing values, which cannot be handled by many data mining techniques, cause no problems for decision trees and may even appear in splitting rules.

This robustness comes at the cost of throwing away some of the information that is available in the training data, so a well-tuned neural network or regression model often makes better use of the same fields than a decision tree. For that reason, decision trees are often used to pick a good set of variables to be used as inputs to another modeling technique. Time-oriented data does require

a lot of data preparation. Time series data must be enhanced so that trends and sequential patterns are made visible.

Decision trees reveal so much about the data to which they are applied that the authors often make use of them in the early phases of a data mining project even when the final models are to be created using some other technique.