Decision Trees for Predictive Modeling

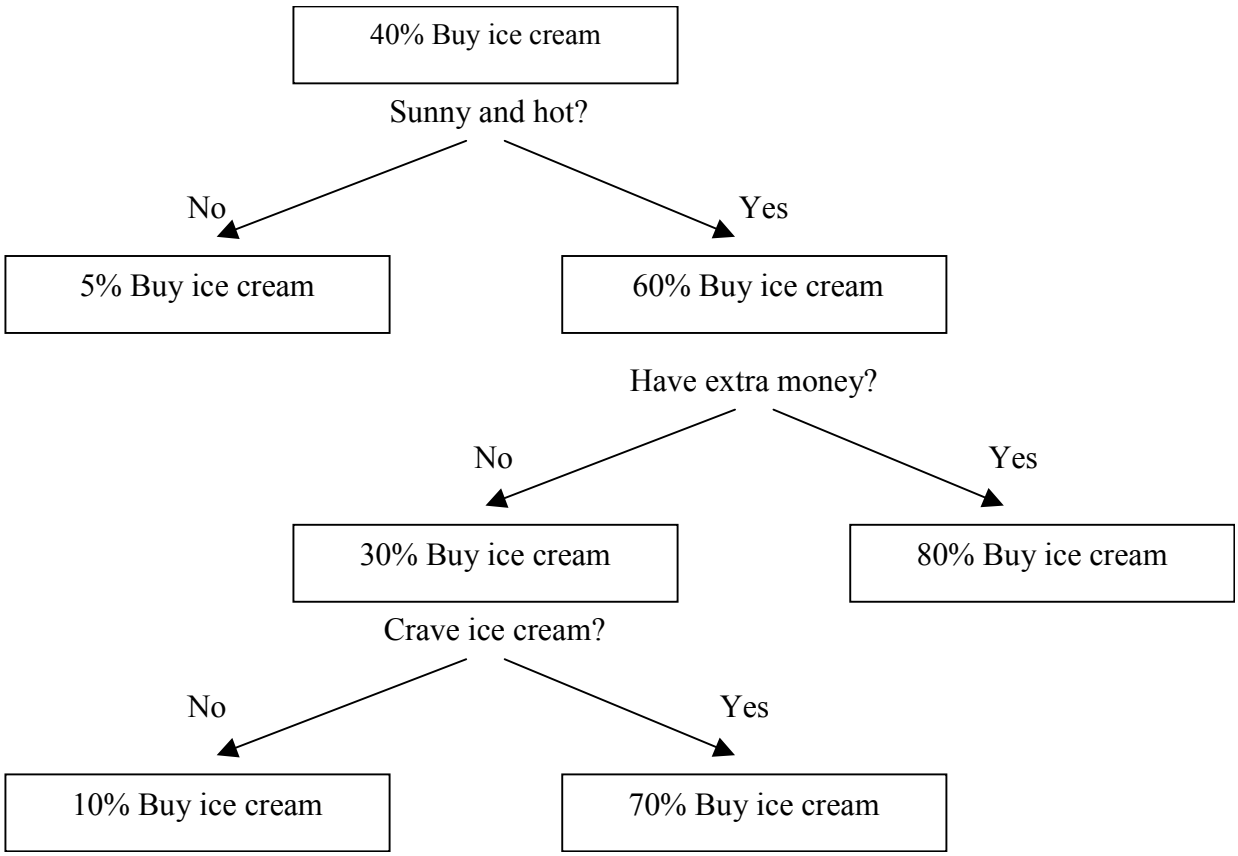
Padraic G. Neville SAS Institute Inc. 4 August 1999

What a Decision Tree Is

A decision tree as discussed here depicts rules for dividing data into groups. The first rule splits the entire data set into some number of pieces, and then another rule may be applied to a piece, different rules to different pieces, forming a second generation of pieces. In general, a piece may be either split or left alone to form a final group.

The tree depicts the first split into pieces as branches emanating from a root and subsequent splits as branches emanating from nodes on older branches. The leaves of the tree are the final groups, the unsplit nodes. For some perverse reason, trees are always drawn upside down, like an organizational chart. For a tree to be useful, the data in a leaf must be similar with respect to some target measure, so that the tree represents the segregation of a mixture of data into purified groups.

Consider an example of data collected on people in a city park in the vicinity of a hotdog and ice cream stand. The owner of the concession stand wants to know what predisposes people to buy ice cream. Among all the people observed, forty percent buy ice cream. This is represented in the root node of the tree at the top of the diagram. The first rule splits the data according to the weather. Unless it is sunny and hot, only five percent buy ice cream. This is represented in the leaf on the left branch. On sunny and hot days, sixty percent buy ice cream. The tree represents this population as an internal node that is further split into two branches, one of which is split again.



The tree displayed has four leaves. The data within a leaf are much more similar with respect to buying ice cream than the overall data with the 40/60 mix. According to the tree, people almost never buy ice cream unless the weather cooperates and either (1) they have some extra money to spend or (2) they crave the ice cream and presumably spend money irresponsibly or figure out another way of buying it. The tree is easy to understand even if the implication of craving ice cream is left to the imagination. Simple decision trees are appealing because of their clear depiction of how a few inputs determine target groups.

Trees are also appealing because they accept several types of variables: nominal, ordinal, and interval. Nominal variables have categorical values with no inherent ordering. Ordinal variables are categorical with ordered values, for example: `cold', `cool', `warm', and `hot'. Interval variables have values that can be averaged. Temperature is an interval variable when its values are expressed in degrees. A variable can have any type regardless of whether it serves as the target or as an input. (The input variables are those available for use in splitting rules.) Other virtues of trees include their robustness to missing values and distribution assumptions about the inputs.

Trees also have their short comings. When the data contain no simple relationship between the inputs and the target, a simple tree is too simplistic. Even when a simple description is accurate, the description may not be the only accurate one. A tree gives an impression that certain inputs uniquely explain the variations in the target. A completely different set of inputs might give a different explanation that is just as good. The issues of descriptive accuracy, uniqueness, and reliability of prediction are extensively discussed in this paper.

The adjective 'decision' in "decision trees" is a curious one and somewhat misleading. In the 1960s, originators of the tree approach described the splitting rules as decision rules. The terminology remains popular. This is unfortunate because it inhibits the use of ideas and terminology from "decision theory". The term "decision tree" is used in decision theory and project management to depict a series of decisions for choosing alternative activities. The analyst creates the tree and specifies probabilities and benefits of outcomes of the activities. The software finds the most beneficial path. The project follows a single path and never performs the unchosen activities.

Decision theory is not about data analysis. The choice of a decision is made without reference to data. The trees in this article are only about data analysis. A tree is fit to a data set to enable interpretation and prediction of data. An apt name would be: *data splitting trees.*

What to Do with a Tree

Prediction is often the main goal of data analysis. Creating a predictive model is not as automatic as one might hope. This section describes some of the practical hurdles involved and how trees are sometimes used to overcome them.

Variable selection

Data often arrive at the analyst's door with lots of variables. The baggage sometimes includes a dictionary that makes uninteresting reading. Yet the analyst must find something interesting in the data. Most of the variables are redundant or irrelevant and just get in the way. A preliminary task is to determine which variables are likely to be predictive.

A common practice is to exclude input (independent) variables with little correlation to the target (dependent) variable. A good start perhaps, but such methods take little notice of redundancy among the variables or of any relationship with the target involving more than one input. Nominal variables are handled awkwardly.

An alternative practice is to use inputs that appear in splitting rules of trees. Trees notice relationships from the interaction of inputs. For example, buying ice cream may not correlate with having extra money unless the weather is sunny and hot. The tree noticed both inputs. Moreover, trees discard redundant inputs. Sunny-and-hot and ozone-warnings might both correlate with buying ice cream, but the tree only needs one of the inputs. Finally, trees treat nominal and ordinal (categorical) inputs on a par with interval (continuous) ones. Instead of the categorical sunny-and-hot input, the split could be at some value of temperature, an interval input.

The analyst would typically use the selected variables as the inputs in a model such as logistic regression. In practice trees often provide far fewer variables than seem appropriate for a regression. The sensible solution is to include some variables from another technique, such as correlation. No single selection technique is capable of fully prophesizing which variables will be effective in another modeling tool.

Hard-core tree users find ways to get more variables out of a tree. They know how to tune tree creation to make large trees. One method uses the variables in 'surrogate' splitting rules. A surrogate splitting rule mimics a regular splitting rule. Surrogates are designed to handle missing values and are discussed more in that context. Using them to select variables is an afterthought. A surrogate variable is typically correlated with the main splitting variable, so the selected variables now have some redundancy.

Variable importance

The analyst may want the variable selection technique to provide a measure of importance for each variable, instead of just listing them. This would provide more information for modifying the selected list to accommodate other considerations of the data analysis. Intuitively, the variables used in a tree have different levels of importance. For example, whether or not a person craves ice cream is not as important as the weather in determining whether people will buy ice cream. While craving is a strong indication for some people, sunny and hot weather is a good indicator for many more people. What makes a variable important is the strength of the influence and the number of cases influenced.

A formulation of variable importance that captures this intuition is as follows: (1) Measure the importance of the model for predicting an individual. Specifically, let the importance for an individual equal the absolute value of the difference between the predicted value (or profit) of the individual with and without the model. (2) Divide the individual importance among the variables used to predict the individual, and then (3) average the variable importance over all individuals.

To divide the importance for an individual among the variables (2), note that only the variables used in splitting rules that direct the individual from the root down the path to the leaf should share in the importance. Apportion the individual importance equally to the rules in the path. If the number of such rules is D, and no two rules use the same variable, then each variable is accredited with 1/D times the individual importance. This formulation ensures that the variable used to split the root node will end up being the most important, and that a variable gets less credit for individuals assigned to leaves deeper down the tree. Both consequences seem appropriate.

Currently no software implements this formulation of importance. Some software packages implement an older formulation that defines the importance of a splitting rule, credits that importance to the variable used in the rule, and then sums over all splitting rules in the tree. For an interval target, the importance of a split is the reduction in the sum of squared errors between the node and the immediate branches. For a categorical target, the importance is the reduction in the Gini index. Refer to Breiman et al. (1984) and Friedman (1999) for discussions about this approach.

Interaction detection

Having selected inputs to a regression, the analyst will typically consider possible interaction effects.

Consider modeling the price of single family homes. Suppose that the prices of most houses in the data set are proportional to a linear combination of the square footage and age of the house, but houses bordering a golf course sell at a premium above what would be expected from their size and age. The best possible model would need a golf course indicator as input. Data rarely come with the most useful variables. However, it seems plausible that the houses bordering the golf course are about the same size and were built about the same time. If none of the other houses of that size were built during that time, then that size and time combination provides an indication of whether the house borders the golf course. The regression should contain three variables: square footage, age, and the golf course indicator. The indicator is constructed from square footage and age and, therefore, represents an interaction between those two inputs.

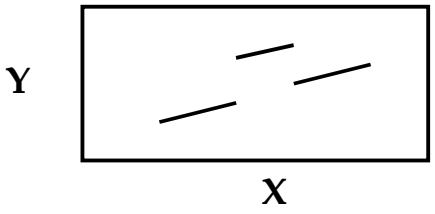
The analyst of course is not told anything about any golf course, much less given any clues as to how to construct the indicator. The seasoned analyst will suspect that key information has to be inferred indirectly, but what and how is guess work. He will try multiplying size and age. No luck. He might then try creating a tree and creating an indicator for each leaf. For a particular observation, the indicator equals one if the observation is in the leaf and equals zero otherwise. The regression will contain square footage, age, and several indicators, one for each leaf in the tree. If the tree creates a leaf with only the houses bordering the golf course, then the analyst will have included the right interaction effects. The indicators for the other leaves would not spoil the fit. Indicators for nonleaf nodes are unnecessary because they equal the sum of indicators of their descendents.

This technique works when the influence of the interaction is strong enough, but it does not work otherwise. Consider a data set containing one target Y and one input X that are related through the equation

$$Y = a X + b W$$

where a and b are constants, and W is like the golf course indicator: it takes values 0 and 1, it is omitted from the data set, but it is 1 if and only if X is in some interval. The analyst must infer the influence of W on Y using only X.

The diagram illustrates the situation in which W equals one when X is near the middle of its range. The tree creation algorithm searches for a split on X that best separates data with small values of Y from large ones. If b is large enough, values of Y with W = 1 will be larger than values of Y with W = 0, and the algorithm will make the first two splits on X so as to isolate data with W = 1. A tree with three leaves appears in which one leaf represents the desired indicator variable.



When b is smaller so that data with extreme values of Y have W = 0, the data with W = 1 have less influence on the algorithm. A split that best separates extreme values of Y is generally different from a split that best detects interactions in linear relationships. A tree creation algorithm specially designed for discovering

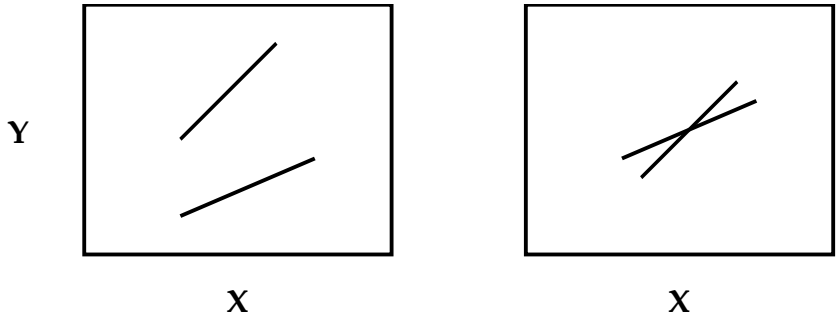
interactions is needed. Otherwise, even if the W = 1 data is eventually separated, the tree will have many superfluous leaves.

Stratified modeling

The analyst preparing for a regression model faces another hidden pitfall when the data represent two populations. A different relationship between an input and the target may exist in the different populations. In symbols, if Y = a + b X + e and Y = a + c X + e express the relationship of Y to X in the first and second population respectively, and b is very different from c, then one regression alone would fit the combined data poorly.

The problem for the analyst is to recognize the need to perform two regressions instead of one. For this purpose, some analysts first create a small decision tree from the data, and then run a separate regression in each leaf. This is called *stratified regression*. Unfortunately, the tree usually will not split data the way the analyst hopes.

Consider a data set with target Y and inputs X and W, where W=0 for one population, and W=1 for the other. Suppose Y = (a + b X) W + (c + d X) (1 - W) for some constants a, b, c, and d. The analyst will apply the tree algorithm to split the data and then fit a regression in each leaf. He hopes that the algorithm will split on W. The predicament is similar to that described for detecting interactions. The tree algorithm will try to separate extreme values of Y. If the values of Y in one population are all larger than those in the other population, then the algorithm will split on W. In the other extreme, if the average of Y is the same in the two populations, the algorithm will split on X. A standard tree creation algorithm separates the populations in the left diagram but not those in the right one. An algorithm designed specifically for stratified regression is needed. Alexander and Grimshaw (1996) and Neville (1999) discuss such algorithms.



Missing value imputation

The analyst may have to contend with missing values among the inputs. Decision trees that split on one input at a time are more tolerant to missing data than models such as regression that combine several inputs. When combining several inputs, an observation missing any input must be discarded. For the simplest of tree algorithms, only observations that need to be excluded are those missing the input currently being considered to split on. They can be included when considering splitting on a different input.

If twenty percent of the data are missing at random, and there are ten inputs, then a regression can use only about ten percent of the observations, while a simple split search algorithm will use about eighty percent. Algorithms that treat missing observations as a special value will use all the observations.

Common tree algorithms that exclude missing data during the split search handle them intelligently when predicting a new observation. The program of Breiman et al. (1984) applies surrogate rules as a backup

when missing data prevent the application of the regular splitting rule. Ross Quinlan's (1993) C4.5 and C5 programs will take a weighted average of the predictions across all branches where missing data prevented a definitive assignment.

An analyst preparing for a regression with missing data might first replace the missing values with guesses. This is called *imputing*. A natural approach is to fit a model to the nonmissing values to predict the missing ones.

Trees may be the best modeling tool for this purpose because of their tolerance to missing data, their acceptance of different data types, and their robustness to assumptions about the input distributions. For each regression input, construct at tree that uses the other inputs to predict the given one. If X, Y and Z represent the inputs, create a tree to predict X from Y and Z, another tree to predict Y from X and Z, and another to predict Z from X and Y.

While trees are probably the best choice in an imputation approach based on prediction, other approaches for handling missing values may be better for a given situation. For example, the variance of estimates based on imputed data may be smaller than it would be if none of the data were missing, depending on how the imputation is done. Imputing with random values drawn from a distribution with the correct variance may be best when the variances matter.

Model interpretation

Trees are sometimes used to help understand the results of other models. An example occurs in market research. A company may offer many products. Different customers are interested in different products. One task of market research is to segregate the potential customers into homogeneous segments and then assign marketing campaigns to the segments. Typically, no response information is available on the customers and so no target variable exists.

Segmentation is based on similarities between input variables. People differ somewhat in their purchasing behavior depending on demographics: their age, family status, and where they live. Demographic information is relatively easy to obtain, and missing data can often be imputed using census information. The segments are intentionally developed using variables for which complete data are available so that everybody can be assigned to a segment.

After the segments are built, the average age, income, and other statistics are available for each segment. However, these demographic statistics are not very suggestive of what products to market. The next step is to select a sample from each segment and ask the people about their life-style and product preferences. Finally, combine the samples from all the segments into one data set and create a tree using the survey questions as inputs and the segment number as the target. Using just a few segments with an equal number of people in each gives the best chance of obtaining a useful tree. With a little luck, the tree will characterize some segments by the type of clothing, cars, or hobbies that suggest what products people in the segment would like to purchase.

Predictive modeling

This section has described how to use trees to overcome some hurdles in predictive modeling. In each example, the tree helps prepare the data or interpret the results of another predictive model. Actually, none of the inventors of tree algorithms were motivated by such supportive roles.

Many individuals have come up with innovative tree creation algorithms. Important ones come from Morgan and Sonquist (1963), Kass (1980), Breiman et al. (1984), and Quinlan (1979). The disparate

approaches rival each other. Yet the originators share the common inspiration that trees by themselves are effective predictive models. Each author can describe studies in which trees simply out perform other predictive techniques.

Consider the circumstances that motivated Morgan and Sonquist. They analyzed data in search of determinants of social conditions. In one example, they tried to untangle the influence of age, education, ethnicity, and profession on a person's income. Their best regression contained 30 terms (including interactions) and accounted for 36 percent of the variance.

As an alternative to regression, they organized the observations into 21 groups. The income of an observation in a group was estimated by the group mean. The groups were defined by values on two or three inputs. Nonwhite high school graduates had a mean income of $5,005. White farmers who did not finish high school had a mean income of $3,950, and so on. This method of prediction accounted for 67 percent of the variance.

The study is interesting because it reveals the inadequacy of regression to discern some relationships in data. Of course every statistician knows this, but Morgan and Sonquist showed how common the problem is in social research and how easily trees get around it. They developed this point in a 1963 article in the *Journal of the American Statistical Association*. They then created the first statistical tree software and called it AID. The program and its successors stayed in service at the Survey Research Center for more than 30 years.

Trees do not supercede other modeling techniques. Different techniques do better with different data and in the hands of different analysts. However, the winning technique is generally not known until all the contenders get a chance. Trees should contend along with the others. Trees are easy.

How to Create a Tree

The simplicity of a decision tree belies the subtleties of creating a good one. A comparable task of fitting a parametric statistical model to data often consists of formulating a log likelihood and solving it numerically. Numerical optimization for trees is infeasible.

On the other hand, tree creation algorithms are easy to invent. Indeed, in the absence of mathematical theorems for judging the superiority of one method over another, many competing algorithms are available. Despite this anarchy, the qualities that characterize a good tree model are the same as those of a parametric one. A parametric model assumes that the data share a common relationship between the inputs and the target, and assumes that the relationship is obscured by idiosyncrasies of individual cases (ineptly called "errors"). A good model reveals a relationship that, depending on the needs of the analyst, either accurately describes the data, accurately predicts the target in similar data, or provides insight. Mathematics forms the basis for trusting the model: if the analyst believes the assumptions of the model, he believes the results. Trees are the same in all respects except one: trust in the model is gained from applying it to a fresh sample.

The following discussion surveys the anarchy of tree creation algorithms and points out what to consider for creating good trees. Murthy (1998) gives a much more extensive survey of algorithmic ideas.

An easy algorithm

Inventing a new algorithm for creating a tree is easy. Here is one.

Find the input variable X that most highly correlates with the target. Tentatively split the values of X into two groups and measure the separation of the target values between the groups. For an interval target, use a Student's *t* statistic as the measure. For a categorical target, use a chi-square measure of association between the target values and the groups. Find a split on X with the largest separation of target values. Use this split to divide that data. Repeat the process in each group that contains at least twenty percent of the original data. Do not stop until every group of adequate size that can be divided is divided.

For many data sets, this algorithm will produce a useful tree. However, its useful scope is limited. Several choices inherent in the algorithm may have more effective alternatives. The most dangerous choice is the stopping rule that may result in predictions that are worse than having no tree at all. The remainder of this section discusses the following topics:

- ◊ Selection of a splitting variable
- ◊ Number of branches from a split
- ◊ Elusive best splits
- ◊ Stepwise, recursive partitioning
- ◊ Stopping or pruning
- ◊ Multivariate splits
- ◊ Missing values

Selection of a splitting variable

The easy algorithm presented in the preceding section first selects a splitting variable and then searches for a good split of its values. Almost all the popular algorithms search for a good split on all the inputs, one at a time, and then select the input with the best split. This seems more reasonable because the goal is to find the best split.

However, Loh and Shih (1997) report that splitting each variable significantly biases the selection towards nominal variables with many categories. Moreover, the algorithms that search for a good split on each variable take longer. Neither approach dominates in terms of classification accuracy, stability of split, or size of tree. Thus, the issue may be of concern if the goal of the tree creation is interpretation or variable selection, and not prediction.

Loh and Shih did not consider the possibility of searching for a split on each input and then penalizing those splits that are prone to bias when comparing the different inputs. The CHAID algorithm due to Gordon Kass (1980) does this by adjusting *p*-values.

Number of branches

The easy algorithm always splits a node into two branches so as to avoid having to decide what an appropriate number of branches would be. But this easy approach poorly communicates structure in the data if the data more naturally split into more branches. For example, if salaries are vastly different in California, Hawaii, and Nebraska, then the algorithm ought to separate the three states all at once when predicting salaries. Gordon Kass seemed to think so when he commented that binary splits are often misleading and inefficient.

On the other hand, several practitioners advocate using binary splits only. A multiway split may always be accomplished with a sequence of binary splits on the same input. An algorithm that proceeds in binary steps has the opportunity to split with more than one input and thus will consider more multistep partitions than an algorithm can consider in a single-step multiway split. Too often the data do not clearly determine the

number of branches appropriate for a multiway split. The extra branches reduce the data available deeper in the tree, degrading the statistics and splits in deeper nodes.

Elusive best splits

The easy algorithm blithely claims it finds the split on the selected variable that maximizes some measure of separation of target values. Sometimes this is impossible. Even when it is possible, many algorithms do not attempt it.

To understand the difficulty, consider searching for the best binary split using a nominal input with three categories. The number of ways of assigning three categories to two branches is two times two times two, or eight. The order of the branches does not matter, so only half of the eight candidate splits are distinct. More generally, the number of distinct binary splits of C nominal categories is two times itself C-1 times, written 2^(C-1). Many more splits are possible on inputs with more nominal categories, and this creates a bias towards selecting such variables.

C	Number of Splits
5	32
10	1024
15	32767
20	1048576
25	33554432
30	1073741824
35	34359738368
40	1099511627776

The table shows the number of distinct binary splits for several values of C. The number of splits is very large for moderate values of C. In practice, the number of splits is often too large for a computer to examine each of them. The number increases exponentially with C, and therefore parallel processing, which increases the computer capacity linearly with the processors, does not help. Either the best split can be found without examining all splits, or it cannot be found without luck.

When the target is binary or interval, the best split can be found without examining all splits: order the categories by the average of the target among all observations with the same input category, then find the best split maintaining the order. This method works only for binary splits with binary or interval targets. For other situations, all the candidate splits must be examined.

Many available algorithms do not attempt to find the best split on a variable. Such methods of searching are called *heuristic* because they do not guarantee optimality. For a nominal input variable, a common theme is first to form a split by assigning a different branch to each input category, and then to derive a sequence of candidate splits by combining two branches of one split to produce the next split in the sequence. Different algorithms differ in the details and variations on this theme. Most heuristic algorithms adopt the best split examined. The original CHAID algorithm may be the only exception. CHAID adopts the last split examined.

Recursive partitioning

The easy algorithm first splits the entire data set, the *root node*, into two nodes. Each node is then split into more nodes, and so on. Each node in its turn is considered in isolation. The search for a split is done without any concern as to how another node is to be split. This process is called *recursive partitioning* and the great majority of tree creation algorithms do it.

Recursive partitioning is myopic: it focuses on optimizing individual splits and pays no attention to the quality of the entire tree. Alternatives to recursive partitioning have not become popular though. They are

hard to implement and take much longer to run. Moreover, recursive partitioning generally gives adequate results.

The first attempt to cure the myopia is to "look ahead one generation" before committing to a split. A candidate split creates two branches. A binary split is then found in each branch. The evaluation of the candidate split is based on the four nodes in the second generation. James Morgan and John Sonquist implemented look-ahead split searches in the AID program in the 1960s. The user's manual by Sonquist et al. (1971) indicates that looking ahead does not help much. A recent, more extensive study by Murthy came to the same conclusion and reported that look-ahead splits sometimes created worse trees.

Chipman, George, and McCullogh (1998) describe an algorithm that searches for an optimal tree. Instead of evaluating candidate splits, their algorithm evaluates candidate trees. One tree is derived from another using a simple operation. The operation is selected at random from a set of possible operations. An example of an operation is selecting a particular variable and forming a random split on that variable. Another operation is to prune a split, transforming an internal node into a leaf. Thus, some operations create larger trees, some create smaller trees. The algorithm is sometimes successful for small trees. Whether the algorithm can examine enough large trees to be useful remains to be seen.

Kristin Bennett (1994) implemented an algorithm for finding globally optimal splits in a tree given the tree structure. The structure consists of the arrangement of nodes and the predicted values assigned to each leaf. The splits are initially undefined, and therefore the nodes are not initially associated with data. The algorithm defines splits that segregates the data into the nodes in a way that is most consistent with the pre-assigned predictions in the leaves. A recursive partitioning algorithm can create an initial structure.

Stopping

The size of a tree may be the most important single determinant of quality, more important, perhaps, than creating good individual splits. Trees that are too small do not describe the data well. Trees that are too large have leaves with too little data to make any reliable predictions about the contents of the leaf when the tree is applied to a new sample. Splits deep in a large tree may be based on too little data to be reliable.

The easy algorithm tries to avoid making trees too small or too large by splitting any node that contains at least twenty percent of the original data. This strategy is easily foiled. Twenty percent of the data will be too few or too many when the original data contain few or many cases. With some data sets, the algorithm will split a node into branches in which one branch has only a couple of cases. Many splits are required to whittle the thick branches down to the limit of twenty percent, so most of the data will end up in tiny leaves.

Certain limitations on partitioning seem necessary:

- the minimum number of cases in a leaf
- the minimum number of cases required in a node being split
- the depth of any leaf from the root node

Appropriate values for these limits depend on the data. For example, a search for a reliable split needs less data the more predictable the data are. A stopping rule that accounts for the predictive reliability of the data seems like a good idea. For this purpose, the CHAID algorithm applies a test of statistical significance to candidate splits. Partitioning stops when no split meets a threshold level of significance. Unfortunately, this

stopping rule turns out to be problematic in three related aspects: choice of statistical test, accommodations for multiple comparisons, and the choice of a threshold.

A test of significance is problematic first because a test must assume some joint distribution of the target and an input. The assumptions become important in the smaller nodes, which are the more numerous down the tree. The appropriate distribution is generally unknown. Worse, the distributions will differ in different nodes when the tree segregates different subpopulations down different branches.

For algorithms that search for a split on every input variable, the test of significance is performed on the maximum value of the splits on the various inputs. Using the distribution of the extreme values of V test statistics, where V equals the number of inputs, is more appropriate than using the distribution for a single test statistic. Such tests seem impractical for computational reasons. Even if an appropriate test were used, the test would be applied thousands of times in the same tree: once for each input in each node. These criticisms of using significance tests persuade some people but not others. Advocates of the tests adjust the threshold significance levels to accommodate multiple tests.

Pruning

Many practitioners contend that stopping rules cannot work. The inherent fallacy is the assumption that an appropriate threshold may be set without much understanding of the data.

The alternative is to grow a tree that is too large, and then prune the tree to the right size. All candidate sub-trees pruned from the large tree are available at the same time for comparison, and this gives retrospective pruning an advantage over a stopping rule that can consider only a single node at one time. The pruning process may evaluate entire subtrees instead of individual splits. An evaluation measure may be chosen that is more relevant to the end use of the tree than the splitting criterion. The proportion of cases misclassified is a common choice.

Stopping or pruning is necessary to avoid over-fitting the data. A model "over-fits" when the good fit to the training data (the data used to create the model) is not replicated when the model is applied to a different sample. Ideally, the pruning process uses a separate "pruning data set" to evaluate sub-trees. Subtrees that over-fit the training data will evaluate poorly on the pruning data. Ross Quinlan (1987) may have published the first algorithm that finds the subtree with the smallest error on pruning data. He calls it *reduced-error pruning.*

The original pruning algorithm is due to Breiman et al. (1984). Evaluating all possible subtrees may have been too time consuming in the 1970s, and so the authors found a fast method of creating a sequence of subtrees of different sizes. Each subtree is a subtree of all the larger ones in the sequence. More importantly, each subtree obtains the optimum assessment among all possible subtrees of its size, based on the training data. Each subtree in the sequence is then applied to a fresh sample, and the larger tree is pruned to the winning subtree. This method is called *cost-complexity pruning*, and results in different trees than reduced-error pruning.

Sometimes tree creation must use all the available data, so no pruning data exist. Ross Quinlan's C4.5 and C5 artificially inflate the error rate of the training data in each node and evaluate subtrees with the inflated rate. This is called *pessimistic pruning*. Breiman et al. use a cross validation method to choose the subtree from their nested sequence of subtrees.

Multivariate splits

The easy algorithm uses a single input to define a splitting rule. Such univariate splits are easy to understand, are tolerant of missing data (because missing values in any other input are of no concern), and implicitly select a small number of inputs to create an entire tree, thereby reducing subsequent demands on data collection and analysis.

Defining splitting rules with many variables also has merit. The OC1 algorithms of Murthy et al. (1994) and the SVM tree algorithms of Bennett (1997) both base splits on linear combinations of variables and produce trees that are often more accurate and smaller than univariate trees. Smaller trees are easier to understand. Smaller trees segregate the data into fewer leaves. Describing the data in a leaf in terms of the variables defining the leaf is cumbersome with many defining variables, whether the leaf is in a large tree with univariate splits or a small tree with multivariate splits. The leaves may be easier to understand by plotting the distribution of variables in the leaves, as done in clustering. Having fewer leaves to profile is helpful.

Missing values

Decision trees accommodate missing values very well compared to other modeling methods. The simplest strategy is to regard a missing value as a special nonmissing value. For a nominal input, missing values simply constitute a new categorical value. For an input whose values are ordered, the missing values constitute a special value that is assigned a place in the ordering that yields the best split. The place is generally different in different nodes of the tree.

The strategy pays off when missing values are indicative of certain target values. For example, people with large incomes might be more reluctant to disclose their income than people with ordinary incomes. If income were predictive of a target, then missing income would be predictive of the target, and the missing values would be regarded as a special large income value. The strategy seems harmless when the distribution of missing values is uncorrelated with the target because no choice of branch for the missing values would help predict the target.

Regarding missing values as a special value is sometimes inappropriate. If a large proportion of values is missing, then they may unduly influence the creation of the split. Evaluating a split using only known information improves credibility. Excluding observations with missing values is feasible with univariate trees because only observations missing on a single input are excluded at any one time. This compares favorably with other modeling techniques that exclude observations missing any input value, which may leave very few usable observations.

When missing values are excluded from the split search, a new policy is needed for assigning missing observations to branches. One policy is to use surrogate splitting rules. A surrogate rule substitutes for the main rule when the main rule cannot handle an observation. A good surrogate rule is one that mimics the main rule very well, even if it does not define a good split in its own right. For example, the main rule might split on county, and the surrogate might split on region. The surrogate applies to observations with an unknown county and a known region. The surrogate might be less effective as a main splitting rule because region represents coarser information than county.

The surrogate policy relies on redundant inputs. Another policy is needed when no good surrogate exists. Assigning all the observations with missing values to a single branch is apt to reduce the purity of the branch, thereby degrading the split. If this is unavoidable, assigning the largest branch results in the least dilution of node purity. Another idea is to split an individual observation and to assign the pieces to all branches in proportion to the size of each branch. For a binary split, the observation is replaced by two

observations with fractional sizes. One fractional observation is assigned to each branch. The prediction for an observation is the weighted average of predictions of the derived fractional observations.

What Can Go Wrong

Trees may deceive. They may fit the data well but then predict new data worse than having no model at all. This is called *over-fitting*. They may fit the data well, predict well, and convey a good story, but then, if some of the original data are replaced with a fresh sample and a new tree is created, a completely different tree may emerge using completely different inputs in the splitting rules and consequently conveying a completely different story. This is called *instability*.

A little education renders all this harmless. Over-fitting is the more serious issue. The remainder of this section discusses how to detect and avoid it. Instability is a simpler issue. It occurs when two inputs have similar predictive ability. The tree can choose only one. The story conveyed is incomplete. The solution is simply to understand that the story is too simplistic when conflicting variables have comparable predictive ability. Instability is actually beneficial for the new prediction techniques based on re-sampling. The phenomenon is discussed fully in the section "The Frontier."

Too good a fit

Over-fitting occurs when the tree fits random variations of the target values in the training data that are not replicated in other samples. The tree fits the training data deceptively well. Over-fitting could be largely due to a single spurious split or to the accumulation of small errors from many splits. Splits generally perform better on the training data than on new data. Even a split representing a `true' prediction performs slightly worse on new data because some randomness exists in the data and the split fits some of that. Smaller nodes are more prone to mistakes.

The surest way to detect over-fitting is to apply the tree to new data. To detect what size of a tree provides a good predictive fit, prune the large tree to subtrees of several sizes, and apply all the subtrees to the new data. Compare the fit to the new data as the tree size increases. Prune to the sub-tree where the fit flattens out. In practice, the choice of subtree is not always clear-cut. The improvement in fit will decrease gradually, sometimes bouncing up and down around a trend. The analyst must decide on the details for choosing the tree size.

Spurious splits due to too many variables

An appealing feature of trees with univariate splits is the resilience against the curse of dimensionality: faced with many inputs, the tree creation algorithm finds a few with predictive ability. Even with more inputs than observations, an algorithm can create a useful tree. In successful examples like these, inputs with predictive ability beat out the inputs that might have only a chance association with the target. The more inputs that are irrelevant, the greater the chance that random association of some irrelevant input will be with the target, and the larger the true association of a predictive input must be to be selected for a splitting rule. When searching for a few good inputs among a thousand or more irrelevant ones, the creation of spurious splits is an important concern.

For a good split, the distribution of new data in the branches will be similar to the distribution of the training data. A spurious split will have almost no worth when evaluated on new data, and therefore the distributions of the new data in the branches will be similar to those in the node being split, and not like the distributions of the training data in the branches. Consequently, spurious splits are easy to detect when enough new data are available.

When spurious splits are detected retrospectively, the only remedy is to prune them. However, the same spurious splits could be avoided by using the new data during the split search. The algorithm would use the training data to select a split for each input and then use the new data to compare the selected splits between inputs.

When significance tests are used to evaluate a split, the test can be modified to account for the number of inputs. A simple way to do this is to multiply the *p*-value of a test statistic by the number of inputs before comparing the statistic to a threshold level of significance. A better method is to base the *p*-value on the distribution of the best V test statistics, where V equals the number of inputs. Both of these methods work by making it harder to accept any individual split. The more irrelevant inputs, the better a truly predictive split must be to be accepted. Modifying a test for more and more inputs eventually prevents accepting any input, no matter how good.

The best strategy for avoiding spurious results from too many inputs may be to reduce the number of inputs to begin with. Although trees are resilient against many inputs, they are not complete cures.

Spurious splits due to too many nominal values

Spurious splits are also common using nominal inputs with many values. Consider an extreme example in which an input contains a different value for each case, as would occur with a customer id number. The algorithm is free to assign a case to any branch. The algorithm strives to assign cases with different target values to different branches. With a binary target and customer id as an input, the algorithm will do so. Of course, the splitting rule is completely unreliable.

Reliability stems ultimately from repetition: if snow has fallen every January in a certain city every year for the past ten years, it is likely to snow there next year also. If many cases occur with a particular nominal input value, and if the target value is the same for most of them, then the input value predicts the target value somewhat reliably. Even if cases with the input value have diverse target values so that the input value poorly predicts the target, no algorithm can use the input to untangle the target, and so no algorithm can fake a good prediction. The key for finding a reliable nominal split is to use only input values that are represented with many cases. Treat input values with too few cases as if they were missing values. The number of cases depends on the distribution of the target values for the given input value.

An alternative strategy to avoid spurious splits with nominal inputs is to use a significance test that takes account of the number of input values. The worth of a split is penalized more for using a nominal input with more values. The consequences are similar to those resulting from adjusting significance tests for many inputs: inputs with many nominal values are effectively eliminated. Other strategies are to combine values so as to reduce their number or just to exclude the input at the outset.

The Frontier

An important new technique is emerging from research in predictive modeling: strategically re-sampling the data and fitting a model to each sample produces an ensemble that is more accurate and stable than any single model. Decision trees have been the preferred model for comprising the ensemble. The instability of trees, so disastrous for interpretation, is an asset for creating ensembles. The rest of this section explains the phenomenon more fully.

Combining models

An average of several measurements is often more accurate than a single measurement. This happens when the errors of individual measurements more often cancel each other than reinforce each other. An average is also more stable than an individual measurement: if different sets of measurements are made on the same object, their averages would be more similar than individual measurements in a single set are.

A similar phenomenon exists for predictive models: a weighted average of predictions is often more accurate and more stable than an individual model prediction. Though similar to what happens with measurements, it is less common and more surprising. A model relates inputs to a target. It seems surprising that a better relationship exists than is obtainable with a single model. Combining the models must produce a relationship not obtainable in any individual model.

An algorithm for training a model assumes some form of the relationship between the inputs and the target. Logistic regression assumes a linear relation. Decision trees assume a constant relation within ranges of the inputs. Neural networks assume a specific nonlinear relationship that depends on the architecture and activation functions chosen for the network.

Combining predictions from two different algorithms may produce a relationship of a different form than either algorithm assumes. If two models specify different relationships and fit the data well, their average is apt to fit the data better. If not, an individual model is apt to be adequate. In practice, the best way to know is to combine some models and compare the results.

Ensembles

An "ensemble" or "committee" is a collection of models regarded as one combined model. The ensemble predicts an observation as an average or a vote of the predictions of the individual model predictions. The different individual models may give different weights to the average or vote.

For an interval target, an ensemble averages the predictions. For a categorical target, an ensemble can average the posterior probabilities of the target values. Alternatively, the ensemble can classify an observation into the class that most of the individual models classify it. The latter method is called *voting* and is not equivalent to the method of averaging posteriors. Voting produces a predicted target value but does not produce posterior probabilities consistent with combining the individual posteriors.

Unstable algorithms

Sometimes applying the same algorithm to slightly different data produces a very different model. Stepwise regression and tree-based models behave this way when two important inputs have comparable predictive ability. When a decision tree creates a splitting rule, only one input is chosen. Changing the data slightly may tip the balance in favor of choosing another input. A split on one input might segregate the data very differently than a split on the other input. In this situation, all descendent splits are apt to be different.

The unstable nature of tree-based models renders the interpretation of trees tricky. A business may continually collect new data, and a tree created in June might look very different than one created the previous January. An analyst who depended on the January tree for understanding the data is apt to become distrustful of the tree in June, unless he investigated the January tree for instability. The analyst should check the competing splitting rules in a node. If two splits are comparably predictive and the input variables suggest different explanations, then neither explanation tells the whole story.

Bagging

The instability of an algorithm makes possible a different strategy for creating an ensemble of models: instead of varying the algorithm, apply the same algorithm to different samples of the original data. If the algorithm is unstable, the different samples yield different models. The average of the predictions of these models might be better than the predictions from any single model. If the algorithm is stable, then the different samples yield similar models, and their average is no better than a single model.

The usual procedure for creating one of the sampled training sets is as follows. Pick a data case at random from the original training data set and copy it to the sampled data set. Repeat this a second time. The same case might be copied twice, but more likely two different cases will be chosen. Keep repeating this process until the sampled data set contains N cases, where N is arbitrary but is usually equal to the number of cases in the original training data set. This process is called *sampling with replacement*.

"Bagging" refers to the creation of an ensemble of models using the same algorithm on data sets sampled with replacement from a single training data set. Bagging stands for "bootstrap aggregation," and it is the invention of Leo Breiman (1996). He uses the voting method for classification.

Arcing

Bagging uses independent samples of the original data. Arcing takes a sample that favors cases that the current ensemble predicts relatively poorly. The first model trains on all of the original data. Cases that the first model incorrectly classify are preferentially sampled for the second training data set. A second model is trained, and the two models are combined. Cases for which the combined model performs poorly are preferentially sampled for the third training data set, and so on.

Leo Breiman (1998) introduced the term "arcing" to stand for "adaptive resampling and combining," and this seemed a better description of the process of "boosting" introduced by Freund and Schapire (1995, 1996).

Unlike bagging, arcing may give different weights to the individual models. Specifying the weights and the probabilities for sampling a case completely specifies an arcing algorithm. No particular specification seems generally superior to others in practice. Arcing applies only to categorical targets.

Boosting

The term "boosting" is much more commonly used than "arcing" and generally means the same thing. However, Friedman, Hastie, and Tibshirani (1998, 1999) have reformulated the concept without reformulating a new name. Boosting, as reformulated, creates an ensemble consisting of a weighted average of models fit to data in which the target values are changed in a special way. In all other respects, the data are the same for all models.

As an example, consider minimizing the sum of squared errors between an interval target and a prediction from an ensemble. The first individual model is fit to the original data. For the training data set for the second model, subtract the predicted value from the original target value, and use the difference as the target value to train the second model. For the third training set, subtract a weighted average of the predictions from the original target value, and use the difference as the target value to train the third model, and so on.

The weights used in averaging the individual models and the prescription for creating a target value depend on the measure of loss associated with the problem. The prescription just described is used when minimizing the sum of squared errors. A different prescription is used when minimizing the sum of absolute errors. Thus, the measure of loss determines the particulars of a boosting algorithm.

Well-Known Algorithms

In conversations about trees, someone is apt to refer to such-and-such algorithm. The following list contains the best known algorithms and describes how they address the main issues in this paper. Each algorithm was invented by a person or group inspired to create something better than what currently existed. Some of their opinions on strategies for creating a good tree are included. The last entry is SAS algorithms. The SAS software lets the user mix some of the better ideas of the well-known programs. It is still being improved.

AID, SEARCH

James Morgan and John Sonquist proposed decision tree methodology in an article in the *Journal of the American Statistical Association* in 1963. They cite similar ideas of Belson (1959) and others. Their motivation has already been discussed under "Predictive modeling" in the section, "What to Do with a Tree." Their original program was called AID, which is an acronym for "automatic interaction detection." AID represents the original tree program in the statistical community. The third version of the program was named SEARCH and was in service from about 1971 to 1996.

The user's manual for SEARCH (1971, 1975) discusses experiments with tree methodology. The most successful idea was to specify an independent variable for a linear regression, and to search for splits that either optimize the regression fit in the branches or most differentiate slopes of the regression lines in the branches. Often an input variable "so dominates the dependent variable that the data are split on little else." The regression technique removes the dominant effect and reveals what else matters.

James Morgan opposes splitting nodes into more than two branches because the data thins out too quickly. He also opposes subjecting splits to tests of statistical significance because the large number of required tests renders them useless. Morgan and Sonquist advocate using test data to validate a tree.

AID finds binary splits on ordinal and nominal inputs that most reduce the sum of squares of an interval target from its mean. The best split is always found, look-ahead split searches are available, and cases with missing values are excluded. AID attempts splits on nodes with larger sum-of-square errors first, so that the program may stop after splitting some number of nodes specified by the user. The program stops when the reduction in sum of squares is less than some constant times the overall sum of squares. The constant is 0.006 by default.

CHAID

AID achieved some popularity in the early 1970s. Some nonstatisticians failed to understand over-fitting and got into trouble. The reputation of AID soured. Gordan Kass (1980), a student of Doug Hawkins in South Africa, wanted to render AID safe and so invented the CHAID algorithm. CHAID is an acronym for "Chi-Squared Automatic Interaction Detection."

CHAID recursively partitions the data with a nominal target using multiway splits on nominal and ordinal inputs. A split must achieve a threshold level of significance in a chi-square test of independence between the nominal target values and the branches, or else the node is not split. CHAID uses a Bonferroni adjustment for the number of categorical values of the input variable, thereby mitigating the bias towards inputs with many values.

CHAID also uses significance testing to determine the number of branches. The search for a split on a given input proceeds as follows. First assign a different branch to each value of the input. For each pair of branches, form a two-way table of counts of cases in each branch by target value. Find the pair of branches

with the smallest chi-square measure of independence. If the significance level is below a threshold, merge the branches and repeat. Otherwise, consider re-splitting branches containing three or more input values: if a binary split is found that exceeds a threshold significance level, split the branch in two and go back to merging branches.

The search ends when no more merges or re-splits are significant. The last split on the input is chosen as the candidate split for the input. Notice that it is generally not the most significant split examined. In the early 1990s, Barry de Ville introduced "exhaustive CHAID," which adopts the most significant split. Exhaustive CHAID produces splits with more branches than the original CHAID.

CHAID treats missing values as special, additional values. Kass cites Morgan and Messenger (1973) for this suggestion.

An excerpt from a 1980 paper shows Kass' intentions:

> Some specific extensions have been made to AID. They include the [treatment of missing data], and the notion of using the "most significant" split rather than the "most explanatory" which does not take into account the type of input nor its number of categories. [The new] criterion for assessing multi-way subdivisions of the data may yield a more effective analysis than the traditional binary splits that are often misleading and inefficient.

Classification and Regression Trees

Leo Breiman and Jerome Friedman, later joined by Richard Olshen and Charles Stone, began work with decision trees when consulting in southern California in the early 1970s. They published a book and commerical software in 1984.

Their program creates binary splits on nominal or interval inputs for a nominal, ordinal, or interval target. An exhaustive search is made for the split that maximizes the splitting measure. The available measures for an interval target are reduction in square error or mean absolute deviation from the median. The measure for an ordinal target is "ordered twoing." The measures for a nominal target are reduction in the Gini index and "twoing." Cases with missing values on an input are excluded during the search of a split on that input. Surrogate rules are applied to such cases to assign them to a branch.

Historically the most significant contribution is the treatment of over-fitting. The authors quickly concluded that an appropriate threshold for a stopping rule is unknowable before analyzing the data; therefore, they invented retrospective cost-complexity pruning: a large tree is created, then a subtree is found, then another sub-tree within the first, and so on forming a sequence of nested sub-trees, the smallest consisting only of the root node. A subtree in the sequence has the smallest overall cost among all sub-trees with the same number of leaves. For categorical targets, the cost is the proportion misclassified, optionally weighted with unequal misclassification costs. The final pruned tree is selected from subtrees in the sequence using the costs estimated from an independent validation data set or cross validation.

The program contains some other important features: prior probabilities that are incorporated into the split search; use of surrogate splits for missing values; a heuristic search for a split on linear combination of variables, the state of the art for about 15 years until the OC1 and SVM tree algorithms; evaluation of categorical trees in terms of purity of class probabilities instead of classification; bagging and arcing. The original commercial program is still available through Salford-Systems.

ID3, C4.5, C5

Ross Quinlan (1979, 1993) wrote his first decision tree program in 1978 while visiting Stanford University. His ID3 (iterative dichotomizer 3rd) and its replacement, C4.5 programs have served as the primary decision tree programs in the Artificial Intelligence and Machine Learning communities. He attributes the original ideas to Earl Hunt's (1996) "Concept Learning Systems," but these are now buried beneath decades of active tinkering.

C5 creates binary splits on interval inputs and multiway splits on nominal inputs for a nominal target. The split is chosen that maximizes the *gain ratio*: Gain ratio = reduction in entropy / entropy of split. (Let P(b) denote the proportion of training cases a split assigns to branch b, b = 1 to B. The entropy of a split is defined as the entropy function applied to {P(b): b=1 to B}.)

For nominal inputs, each category is first assigned to a unique branch, and then, in steps, two branches are merged, until only two branches exist. The split with the maximum gain ratio among splits examined becomes the candidate split for the input. This search method, of course, is heuristic and might not find the nominal split with the largest gain ratio. The search on an interval input will find the best split.

Cases with missing values are excluded from the split search on that input and also from the numerator of the gain ratio. The entropy of the split is computed as if missing values constitute an additional branch. When a missing value prevents applying a splitting rule to a case, the case is replaced by B new ones, each being assigned to a different branch, and each is assigned a weight equal to the proportion of nonmissing training cases assigned to that branch. The posterior probabilities of the original case equals the weighted sum of the probabilities of the generated observations.

Ross Quinlan advocates retrospective pruning instead of stopping rules. If enough data were available, the pruning process would use data withheld from training the tree to compare error rates of candidate sub-trees. The software does not assume data may be withheld from training, so it implements "pessimistic" pruning. In each node, an upper confidence limit of the number of misclassified data is estimated assuming a binomial distribution around the observed number misclassified. The confidence limit serves as an estimate of the error rate on future data. The pruned tree minimizes the sum over leaves of upper confidences.

Some other features of C5 are, unequal misclassification costs, fuzzy splits on interval inputs, and boosting.

QUEST

QUEST was introduced by Wei-Yin Loh and Yu-Shan Shih (1997) and is an acronym for "Quick, Unbiased, Efficient, Statistical Tree." To become quick and unbiased, this algorithm selects an input variable to split on before searching for a split, thereby eliminating the time required for searching on most inputs and eliminating the bias towards nominal inputs inherent when relying on candidate splitting rules to select the input variable.

A simplified version of the QUEST input selection is as follows. For an interval input, perform a J-way ANOVA, where J is the number of target values. For a nominal input with M values, compute a chi-square statistic from the J by M contingency table. Select the input with the smallest Bonferroni adjusted *p*-value. If a nominal input is selected, it is transformed to an interval one. A linear discriminant analysis is then performed on the single selected input, and one of the discriminant boundary points becomes the split point. Splits on linear combinations of inputs are also possible. QUEST searches for binary splits on nominal or interval inputs for a nominal target. Cases whose values are missing an input are excluded in calculations with that input. Surrogate rules assign such cases to branches. Recursive partitioning creates a large tree that is retrospectively pruned using cross validation.

The paper by Loh and Shih is available at http://www.stat.wisc.edu/~loh. The paper states that QUEST is much better than the exhaustive search methods of Breiman et al. (1984) in terms of variable selection bias and speed, but neither approach dominates in terms of classification accuracy, stability of split points, or size of tree. The idea of first selecting the variable using an *F* statistic and then using LDA to chose a split point had already appeared in the FACT tree algorithm by Loh and Vanichsetakul (1988).

OC1

OC1 was introduced by Murthy et al. (1994) and is an acronym for "Oblique Classifier 1." The paper is available at http://www.cs.jhu.edu/~murthy.

OC1 searches for binary splits on linear combinations of interval inputs for a nominal target. A candidate split can be evaluated as a maximum or sum over branches of the count of the rarest target value or as the reduction in the sum of squared errors, treating the categorical target values as numeric. Missing values are not accepted. Recursive partitioning creates a tree that over-fits the data. The tree is then pruned retrospectively using a measure of node impurity.

A synopsis of the split search algorithm is this: Choose an initial linear combination at random. Apply a heuristic hill-climbing algorithm to find a local optimum. Make a random change or a random restart and climb again.

SAS algorithms

SAS algorithms incorporate and extend most of the good ideas discussed for recursive partitioning with univariate splits. The target and inputs can be nominal, ordinal, or interval. The user specifies the maximum number of branches from a split, thus allowing binary trees, bushy trees, or anything in between. Splits can be evaluated as a reduction in impurity (least squares, Gini index, or entropy) or as a test of significance (chi-square test or *F* test). Significance tests allow Bonferroni adjustments, as done in CHAID, to counter the bias of many categorical values and allow more adjustments for the number of inputs and the depth in the tree of the split.

The program always finds the best split on an input unless too many candidate splits would have to be examined. The user specifies this limit. The limited search on an input begins with a split into many branches and then proceeds in steps of merging pairs of branches. This heuristic search also considers switching input values between branches after every merge, thereby examining many more candidates than just merging (as done in C5 for example). "Switching" replaces the "respliting" finesse of CHAID. The best split examined is adopted, as done in "exhaustive CHAID" and not the original CHAID.

Missing values may optionally be treated as a special, acceptable value, as in CHAID. Surrogate rules, if suitable, assign cases with missing values to a branch, as in the algorithms of Breiman et al. (1984).

Many options control the stopping and retrospective pruning of the tree. As in CHAID, a limit on the level of significance may stop tree growth. After the tree stops growing, smaller trees are found that achieve the best assessment value for their size. The user has great latitude in specifying an assessment measure. Unequal misclassification costs of Breiman et al. (1984) are implemented and extended to allow different costs for different cases and to let the user specify new predictable outcomes that do not appear as target values. The assessment may be done on the whole tree or on the half of the tree that predicts the most desired outcomes. Assessing only the better half of the tree gives an indication of the "lift" the tree produces.

The tree algorithms are included in the SAS Enterprise Miner data mining solution. The SAS Enterprise Miner provides a visual programming environment for predictive modeling. Prior probabilities, unequal misclassification costs, case-dependent costs, bagging and arcing apply to all of the modeling tools. The tree can incorporate prior probabilities into the splitting criterion or just use them to adjust the posterior probabilities. The tree can create an indicator variable for each leaf. These variables automatically enter into other models, such as regressions, placed after the tree.

References

Berry, M and Linoff, G. (1997) provide a non-mathematical introduction to the methods of Breiman et al., Kass (CHAID), and Quinlan (C4.5). Ripley (1996) provides a mathematical introduction to trees. Devroye et al. (1996) provide an advanced mathematical introduction to trees.

http://www.recursive-partitioning.com provides an excellent bibliography, links to major tree and rule induction software web sites, and a few comparison studies. The following references are included in the bibliography on the web.

Alexander, W.P. and Grimshaw, S.D. (1996), "Treed Regression," *Journal of Computational and Graphical Statistics* 5 (2), 156-170. Their algorithm searches for splits that minimize squared errors from regression in branches. All regressions on one independent variable are considered for each split candidate.

Belson, W. A. (1959), "Matching and Prediction on the Principle of Biological Classification," *Applied Statistics*, 8, 65-75. Probably the first published use of decision tree ideas.

Bennett, K.P. (1994), "Global Tree Optimization: A Non-Greedy Decision Tree Algorithm," *Computing Science and Statistics*, 26, 156-160.

Bennett, K.P. and Blue, J. (1997), "A Support Vector Machine Approach to Decision Trees." Available from http://postulate.math.rpi.edu/~bennek.

Berry, M. and Linoff, G. (1997), *Data Mining Techniques*, New York: John Wiley and Sons, Inc. Contains a non-mathematical introduction to the methods of Breiman et al, Kass (CHAID), and Quinlan (C4.5).

Breiman, L. (1996), "Bagging Predictors," *Machine Learning*, 24, No. 2, 123-140.

Breiman, L. (1998), "Arcing Classifiers," *Annals of Statistics*, 26, 801-824. Includes a discussion by Fruend and Shapire. Explore ftp.stat.Berkeley.edu/pub/users/breiman.

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984), *Classification and Regression Trees*, Belmont, California: Wadsworth, Inc.

Carson, R.T. (1984), "Compensating for Missing Data and Invalid Responses in Contingent Valuation Surveys," *Proceedings of the Survey Research Section of the American Statistical Association*. Very obscure paper that compares imputing missing values using trees with expectation-maximization methods.

Chipman, H., George, E.I., and McCullogh, R.E. (1998), "Bayesian CART Model Search (with discussion)," *Journal of the American Statistical Association*, 93, 935-960. Randomly selects splitting and pruning operation to search for a good tree.

Devroye, L., Gyorfi, L., and Gabor, L. (1996), *A Probabilistic Theory of Pattern Recognition*, New York: Springer-Verlag Inc. Excellent text for those comfortable with a measure theoretic formulation of probability theory. Argues (page 338) that impurity measures as used by Breiman et al and Quinlan are too easily fooled, and should therefore be avoided. Applies Vapnik-Chervonenkis theory to construct trees that, with enough data, approach the limiting error rate (Bayes risk) inherent in the data.

Esposito, F., Malerba, D., and Semeraro, G. (1997), "A Comparative Analysis of Methods for Pruning Decision Trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19 (5), 476−491.

Freund, Y. (1995), "Boosting a Weak Learning Algorithm by Majority," *Information and Computation*, 121, 256-285. Introduces an early AdaBoost algorithm in the context of "probably approximately correct" concept learning models.

Freund, Y. and Schapiro, R. (1996), "Experiments with a New Boosting Algorithm," in *Machine Learning: Proceedings of the Thirteenth International Conference*, 148−156. Presents AdaBoost.

Friedman, J. H., "Greedy Function Approximation: A Gradient Boosting Machine," (1999). Postscript technical report trebst.ps available from http://www.stat.stanford.edu/~jhf/ftp. Complete mathematical reformulation of boosting as a series of models on data whose target values are modified.

Friedman, H.J., Hastie, T., and Tibshirani, R. (1998), "Additive Logistic Regression: a Statistical View of Boosting." Technical report available from ftp://stat.stanford.edu/pub/friedman/boost.ps.Z. Mathematical reformulation of boosting techniques containing some of the basic ideas of the 1999 Friedman technical report.

Hunt, E.B. and Hovland, C.I. (1961), "Programming a Model of Human Concept Formation," *Proceedings of the Western Joint Computer Conference,* 145−155. First published implementation of a concept-learning program. The program constructs binary trees from data with a binary target and nominal inputs. The information is also in the book by Hunt, Marin, and Stone (1966).

Hunt, E.B., Marin, J., and Stone, P.J. (1966), *Experiments in Induction*, San Diego, CA: Academic Press, Inc. Seminal text on Concept Learning Systems (CLS). A CLS tries to mimic the human process of learning a concept, starting with examples from two classes and then inducing a rule to distinguish the two classes based on other attributes. The book deals with automatically constructing binary decision trees for a binary target. Hunt was Ross Quinlan's Ph.D. advisor. Ross Quinlan attributes his work on trees (ID3, C5) as originating in Hunt's ideas about CLS.

Jensen, D. D. and Cohen, P. R (1999), "Multiple Comparisons in Induction Algorithms," *Machine Learning* (to appear). Excellent discussion of bias inherent in selecting an input. Explore http://www.cs.umass.edu/~jensen/papers.

Kass, G.V. (1975), "Significance Testing in Automatic Interaction Detection (A.I.D.)," *Applied Statistics*, 24: 178−189. Computes a distribution appropriate for selecting the best split on several inputs.

Kass, G.V. (1980), "An Exploratory Technique for Investigating Large Quantities of Categorical Data," *Applied Statistics*, 29, 119−127. Standard reference for the CHAID algorithm Kass described in his 1975 Ph.D. thesis.

Loh, W. and Shih, Y. (1997), "Split Selection Methods for Classification Trees," *Statistica Sinica*, 7, 815-840. Introduces the QUEST algorithm. Refer to http://www.stat.wisc.edu/~loh.

Loh, W. and Vanichsetakul N. (1988), " Tree-Structured Classification Via Generalized Discriminant Analysis," *Journal of the American Statistical Association*, 83, 715−728.

Morgan, J.N. and Messenger, R.C. (1973), "THAID- a Sequential Analysis Program for the Analysis of Nominal Scale Dependent Variables," Survey Research Center, Institute for Social Research, University of Michigan. Cited by Kass (1980) as suggesting treating missing values as a special, acceptable value while searching for a split.

Morgan, J.N. and Sonquist, J.A (1963), "Problems in the Analysis of Survey Data, and a Proposal," *Journal of the Americal Statistical Association*, 58, 415−35. Thoughtful discussion of the limitations of additive models, and a seminal proposal for sometimes using decision trees in their stead. Still worth reading.

Murthy, S.K. (1998), "Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey," *Data Mining and Knowledge Discovery*, 2, 345−389. Extensive survey of ideas for constructing decision trees.

Murthy, S.K. and Salzberg, S., (1995), "Lookahead and Pathology in Decision Tree Induction," *Proceedings of the Fourtenth International Joint Conference on Artificial Intellegence*.

Murthy, S., Simon, K., Salzberg, S., and Beigel, R. (1994), "OC1: Randomized Induction of Oblique Decision Trees," *Journal of Artificial Intelligence Research*. Refer to http://www.cs.jhu.edu/~murthy/ for more OC1 links.

Neville, P. (1999), "Growing Trees for Stratified Modeling," *Computing Science and Statistics*, 30. Extends the ideas of Sonquist et al. and Alexander and Grimshaw for more independent variables and more types of models.

Quinlan, R.J. (1979), "Discovering Rules from Large Collections of Examples: A Case Study," *Expert Systems in the Micro Electronic Age*, Edinburgh University Press, Michie, D. editor. Introduces the ID3 (Iterative Dichotomizing 3rd) algorithm.

Quinlan, R.J. (1987), “Simplifying Decision Trees,” *International Journal of Man-Machine Studies*, 27, 221-234. Introduces reduced-error pruning.

Quinlan, R.J. (1993), *C4.5: Programs for Machine Learning*, San Mateo, California: Morgan Kaufmann Publishers, Inc. His web site is: http://www.rulequest.com.

Ripley, B.D. (1996), *Pattern Recognition and Neural Networks*, Cambridge University Press.

Sonquist, J.A., Baker, E.L., and Morgan, J.N. (1971, 1975), *Searching for Structure*, Ann Arbor: Institute for Social Research, The University of Michigan. Out of print and hard to locate. User's manual for SEARCH (alias AID-III) tree construction OSIRIS program. An initial section relates their experience with experimental strategies.