Critical Features of High performance Decision Trees

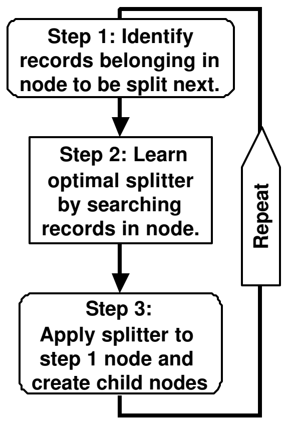
Decision trees have justifiably become one of the most popular data mining tools. They are relatively easy to use, the results can usually be displayed in an easy to read flow chart, and their predictive accuracy can be good to excellent across a broad range of database types and structures. Decision tree products can differ markedly, however, in their flexibility, predictive accuracy, diagnostic feedback, and inherent ability to handle essential data mining tasks. If you choose the wrong decision tree you could easily find yourself unable to conduct certain analyses, or developing models far inferior to those produced by a state-of-the-art package. You could also end up learning much less about your data and waiting longer than necessary for your results. This document reviews some of key features of decision trees that any informed analyst should be thinking about when choosing this kind of data mining tool for important data analyses.

Enforced Versus Optional Sampling

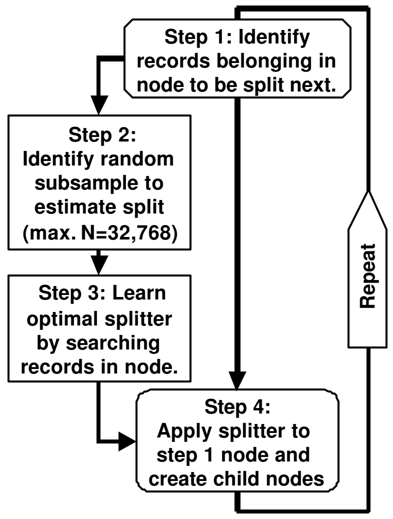
This topic is likely to be an unexpected one for most analysts. You would think that if you ask a data mining product to conduct an analysis of a training database with 100,000 records that the tool will actually train on this data. Unfortunately, this is not necessarily the case. Indeed, at least one major "enterprise class" product will extract a rather small sample from your training data for analysis purposes, with no clear warning that this is happening. The end result is that your core analysis will actually be conducted on a tiny fraction of the data you were intending to analyze. Here is how enforced sub-sampling works:

First, the training data base is identified and readied for analysis by the decision tree. At this stage the training data may be of any size, although you might be required to have sufficient RAM to store it in its entirety. Second, a random sample is quietly selected from this training database without any indication to the analyst that this is happening. For the major product we are describing the default sample size is 5,000 records. The analyst has the option of going into advanced control settings and changing this mandatory sample size but the maximum allowable sample is 32,768. Third, the small sample is searched to identify the best splitting variable and split point. Finally, the entire training database is partitioned using the splitter found. It is this last step that creates the illusion that all the data is being used for the analysis, because once a split has been discovered from the small subsample it is applied to the entire training database. The entire process is repeated at each node in the tree until node sample sizes become very small. Every large node at the top of the tree will be split on the basis of a small subsample and only the

Process *without* subsampling


Step 1: Identify records belonging in node to be split next.
Step 2: Learn optimal splitter by searching records in node.
Step 3: Apply splitter to step 1 node and create child nodes
Repeat

Process *with* subsampling


Step 1: Identify records belonging in node to be split next.
Step 2: Identify random subsample to estimate split (max. N=32,768)
Step 3: Learn optimal splitter by searching records in node.
Step 4: Apply splitter to step 1 node and create child nodes
Repeat

nodes at the bottom of tree will use all the available training data.

The key issues here are not whether you should be sampling from the training data set when searching for a splitter. The issues are: who decides (you or the data mining tool) and are you kept properly informed. In the CART® decision tree you always have full control and you will always know the operational control settings. By default, CART will use all of your training data to search for splitters. If you wish, you can opt to use node sub-sampling to run faster analyses or to generate different trees based on different random node sub-samples. When you use CART the choice will always be yours to make and the log will clearly report the control settings; with some other tools you will not have the option of using all of your data.

In our opinion enforced node sub-sampling can seriously impair the quality of your analyses. Suppose, for example, that you have a 1 million record data base and you wish to study a binary response target with outcomes “Yes” and “No”. If your response is relatively rare, as it might very well be in cases of fraud or drug discovery you could easily have as few as 5,000 “responders”. If you work with a package which enforces random node sub-sampling you would be basing the top of your tree on a random sample of at most 32,768 records containing about 168 responses. Clearly, your ability to learn what is important from this data would be radically compromised.

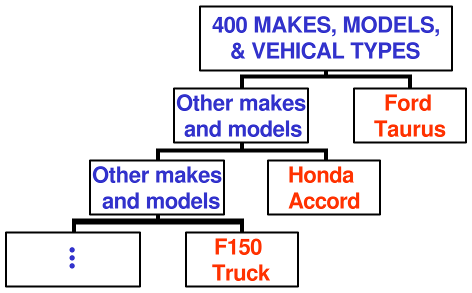
Handling A Large Number of Columns

This decision tree capability appears to be so basic that you would not expect it to appear in a discussion of high performance tools. Nevertheless, this is a dimension you should test explicitly if you expect to need to work with many columns. The CART decision tree is tuned to handle 8,000 columns readily and this number can be increased substantially in special versions of the software. Competing products may well claim that they can handle as many columns as you have in your database but the key question is performance. We know of real world data analysis problems where other decision tree products essentially collapsed under the weight of as few as 1,500 columns. You won’t know how a tool will perform under these heavy loads unless you conduct real world tests. The size of the vendor is not a reliable guide to the capability of the software and neither are descriptive terms like ‘enterprise’ or ‘scalable’. In our experience, the bolder the descriptive terminology the more underpowered the software actually is. If database size is an issue for you, you should conduct serious capacity tests and make sure that the software is actually analyzing all the data you provide.

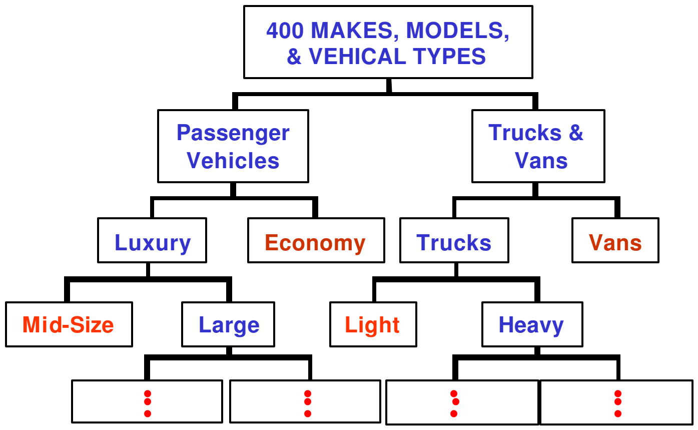
Set of Splitting Rules Available

Several decision tree packages offer a collection of methods or splitting rules, but a few offer only one tree growing method such as entropy or a Chi-squared criterion. In our view it is essential to have a wider menu of options. We would go further though. Rules such as gini and entropy are quite similar to each other and share essentially the same strengths and weaknesses. To offer real variety a decision tree should offer the “twoing” split criterion introduced by Breiman, Friedman, Olshen, and Stone in their classic monograph, Classification and Regression trees (1984). The twoing criterion will grow trees that are quite different from their gini and entropy (or chi-squared-based) counterparts. In particular, the twoing tree tends to be far more balanced in the share of data going to any part of the tree. In contrast, the gini and entropy criteria tend to produce trees that break off a series of (sometimes small) subsets of the data. There will be many circumstances when the twoing trees are far easier to understand and they will appear to tell an intelligible and well-organized story about the data. A useful example is predicting the specific make of new vehicle a consumer will buy from the full range of more than 400 choices available today. A gini tree could well begin in the root node and separate out Ford Taurus buyers whereas a twoing tree would be more likely to separate cars from trucks in the root node. Going further, the twoing tree might continue by separating large vehicles from small ones within both cars and trucks whereas the gini tree might continue by separating out Honda Accords. Which tree is better? From a predictive accuracy perspective they might have very similar performance. But most market researchers would clearly prefer the twoing tree.

A typical gini style tree



A typical towing style tree



How many tree growing rules do you need? Certainly more than one and certainly you need the twoing rule in the tool kit. We offer seven tree growing rules for classification trees: (1) gini (2) symmetric gini (3) twoing (4) power modified twoing (5) ordered twoing (6) entropy and (7) class probability. For regression trees we also offer the least squares and least absolute deviation rules that are discussed further below. The operating characteristics and their strengths and weaknesses are discussed in our extensive 300 page technical manual and briefly described in the on-line help.

You may have observed that we do not offer the CHAID (chi-square automatic interaction detection) splitting rule in spite of the fact that it is one of the simplest trees to implement. We believe that CHAID is a flawed technology that should be approached with great caution. Its biggest flaw is that it is much too capable of growing "false positive trees" – trees that report non-existent data structure. To confirm this for yourself, add some random data columns to a data set you know well and then use one of these random columns as your target variable. You will likely see a disturbingly frequent set of trees generated. By contrast, CART will reject the data and report that it was not possible to grow a defensible tree. This topic will be further discussed in another white paper devoted exclusively to CHAID.

Handling Categorical Variables With Many Levels

Categorical variables (also called nominal variables) are ubiquitous in databases and they are increasingly found with very large numbers of levels. There are over 32,000 distinct zip codes in the United States, there are more than 400 different makes and models of new motor vehicles for sale in the US, and there are over one million web sites live on the internet. It is becoming far more common for researchers to want to make use of such data in data mining models but for the most part decision trees are unable to accommodate the demand. Major decision trees routinely limit the number of levels a categorical variable may take. Some packages limit you to about 36 levels; one "enterprise" class product allows only 128 levels. At Salford Systems we have found such limits to be far too constraining so we have relaxed them entirely. There are no fixed limits on the number of levels a target or predictor variable may take in the CART decision tree. Our own consulting practice has required us to use predictors with 4,000 levels and target variables with more than 400. If you need to work with such data, then to the best of our knowledge, Salford Systems is the only provider of data mining software that can help you.

Active Versus Passive Cost Matrices

Major commercial decision trees allow the user to specify varying costs of misclassification. These costs are used to represent the seriousness of different mistakes. For example, if failing to detect a fraudulent transaction is 10 times more costly than stopping a good transaction we want to reflect this in our analyses. Growing a standard decision tree and then reporting results incorporating costs is quite easy; you could even cut and paste a table of classification results into a spreadsheet, multiply each mistake by its cost and produce a total cost for the tree. Indeed, this is just what the major packages automate for you, and the results are quite informative. But what differentiates a good decision tree from a mediocre one is the ability to use the costs actively while growing the tree. When costs are used actively the decision tree adapts to avoiding the most serious costs in every split. A tree grown with an active cost matrix is likely to be completely different from a tree grown without the costs. When costs are employed passively they do not have any impact on the tree structure at all. Passive costs come into play only after the tree is grown and simply reflect the consequences of the standard tree; they do not influence the structure of the tree in any way.

A tree that does not use cost information while it is being grown is at a serious disadvantage compared to a tree that does use cost information actively. By attempting to avoid serious mistakes throughout the tree growing process the end result will be a better performing tree. So why aren't all decision trees capable of using active cost matrices? There appear to be several reasons. First, working out the mathematical details can be difficult to impossible, depending on the splitting rule involved. Second, it appears that some data mining developers have simply failed to appreciate this subtlety.

How can you tell whether the tree you are evaluating uses active or passive costs? The best way is to set up a multi-class target with 3 or more levels and grow a tree with and without costs. If the splits in the tree are the same in both sets of runs it is incapable of using cost information in the tree growing process.

A major automobile company in the US has been making effective use of active cost matrices by trying to predict the crashworthiness of potential new cars. The company collected all the crash test data produced by the National Highway Traffic Safety Administration (NHTSA) which rates each vehicle with between 1 and 5 stars (5 is best). They augmented the data with detailed engineering specifications describing each car and then developed a CART model to predict the outcome (1,2,3,4, or 5 stars). The company wanted a model that would only make innocuous mistakes and never mistake an unsafe car for a safe one. They first built quite accurate trees without using cost matrices but were troubled by the fact that occasionally a 3-star vehicle would be misclassified as being 4-star. Other mistakes, such as misclassifying a 5-star vehicle as 4-star, or misclassifying a 3-star vehicle as 2-star were considered innocuous. They then rebuilt their trees using a cost matrix to reflect the seriousness of every possible mistake. The end result was a tree that never made unacceptable mistakes and the model is now in use to evaluate the crash worthiness of every potential new car before any engineering work is undertaken. This kind of result is only possible with an active cost matrix technology such as found in the Salford Systems CART decision tree.

A plausible cost matrix for this problem is displayed below. It reflects the fact that if an unsafe car (3 stars or fewer) is chosen for development the downstream costs of reengineering the vehicle once this is discovered can be very substantial. If a safe vehicle is misclassified as unsafe it might not be considered further for development or it might receive additional unnecessary safety engineering. The most important point is that the tree grown with this matrix will quite different than a tree grown without the matrix.

		Model Classification (Safety Rating)				
		1-Star ★	2-Stars ★★	3-Stars ★★★	4-Stars ★★★★	5-Stars ★★★★★
Actual Classification (Safety Rating)	1-Star ★		½	1	***5***	***10***
	2-Stars ★★	½		1	***4***	***9***
	3-Stars ★★★	1	½		***4***	***8***
	4-Stars ★★★★	2	2	2		½
	5-Stars ★★★★★	2	2	2	½	

Class Weights and Prior Probabilities

Priors, also known as class weights, are a key control parameter in sophisticated decision tree products. They are central to any decision tree analysis and you will be using priors whether or not you are aware of it. In brief, priors are used to specify the overall weight placed on a level of the target variable. In the simplest decision tree products, which do not offer explicit priors controls or case weights, the weight of a target variable class is proportional to the number of records in that class. Thus, if you have a file containing 100,000 records, breaking down into 4,000 responders and 96,000 non-responders, most decision trees will automatically apply priors (class weights) of 4% to the responders and 96% to the non-responders. In the CART decision tree terminology these class weights are called "data priors" because they are taken directly from the data. Since a naïve decision tree is trying to maximize the number of cases it classifies correctly such a tree will devote most of its effort to classifying the non-responders. Because the responders make up only 4% of the data they might not receive very much attention. After all, even if you misclassify every responder you would only be wrong 4% of the time! Indeed, a model that is correct 96% of the time can be guaranteed by simply classifying every record as a non-responder. Such a model would be literally accurate but practically worthless.

Various strategies have been taken by decision tree developers to deal with such situations. The most primitive decision trees are unable to do anything internally and the analyst is asked to add clone copies of the responder records to the training data. This is just a low technology way of up-weighting the rare class (or classes). For example, if you copied each responder record 23 times and added these copies to the training data you would have 96,000 records for each class (responders and nonresponders). Some packages may instead recommend that you throw away 92,000 of your nonresponder records so that you would be left with a balanced sample of 4,000 responders and 4,000 non-responders. Neither of these primitive strategies is very appealing. The first requires added data preparation for every analysis and may be infeasible for very large training data bases. The second requires throwing away potentially valuable and informative data.

Very few decision trees follow the CART practice of balancing your data automatically and allowing you to use all of your training data in its original proportions. In the default (but optional) approach pioneered by CART each class, regardless of size, is treated as if it were of the same size as every other class. This "equal weights/equal priors" option is a considerable convenience, allowing any data to be profitably analyzed without special handling, regardless of how unbalanced the target variable might be. Equal priors ensures that a serious attempt to classify all classes correctly will be made.

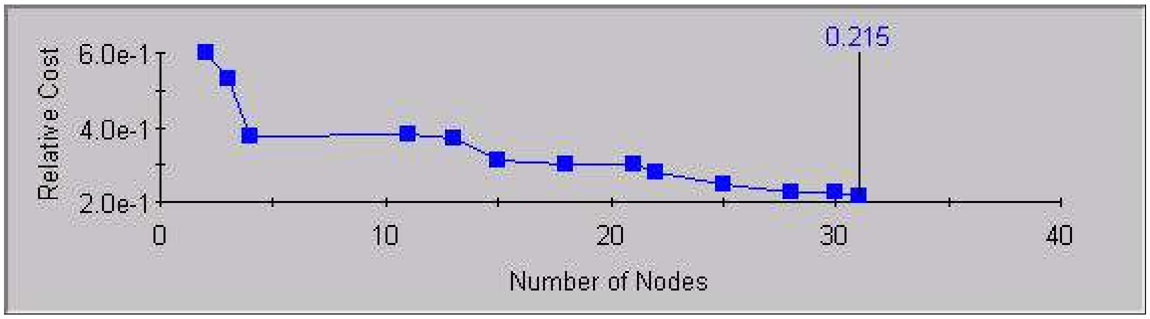
Every decision tree we are aware of allows you to select "data priors" and for several trees this is your only choice. A few more sophisticated trees also offer the 'equal priors" option, which is an important feature. But a high performance tree should also allow you to specify any prior class weights you choose. For example, in our example with 4% of the data being fraud, you should be able to give the fraud class a weight of 8% or 22% or 3% or 55%. Experienced CART modelers know that varying the prior class weights systematically can be used to uncover hot spots consisting of very high concentrations of fraud. Varying the priors can also yield new insight by generating alternative trees with a different slant on the data. But even decision trees that offer equal and data priors may not allow you to specify explicit priors and thus do not allow you to exploit the full power of decision tree technology. It is important to test the software you are considering to ensure that it has actually all the technology that it appears to be delivering. If you cannot set priors freely you will be missing an important decision tree control. The size of the vendor, and the use of 'scalable' and 'enterprise' in the product literature are especially poor guides to the true capabilities of the products.

Growing, pruning, and stopping rules

In the late 1970's and early 1980's CART researchers Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone spent several years testing stopping rules for decision trees. They were finally able to establish that it is impossible to specify a reliable stopping rule; there is always the risk that important data structure might be left undiscovered due to premature termination of the analysis. They suggested instead a revolutionary new two-stage approach to finding the optimal sized tree. In the first stage a very large tree is grown containing hundreds or even thousands of nodes. In the second stage the tree is pruned back and the portions of the tree that are not supported by test data are eliminated. This grow and prune strategy is the only reliable method for arriving at the right sized tree. A decision tree product that does not offer automatic growing and pruning is seriously deficient and is unlikely to consistently deliver high performance results. Unfortunately, some decision tree products, and particularly those based on the CHAID algorithm, are inherently dependent on a stopping rule, and are thus perpetually at risk of missing important data structure.

The critical feature here is the automated growing of a too large tree followed by automated pruning to find the 'right-sized tree". This disciplined process should not be confused with the option to manually prune splits, which is a post-final tree tweaking option found in many decision trees. The rationale for the growing/pruning process is illustrated in the error curve graph below.

CART Decision Tree Error Curve



Note the flat portion of the curve between four and twelve nodes for example. A stopping rule might have concluded that the tree was not capable of making further progress and thus stop too early. Another flat portion of the error curve is observed between fifteen and twenty one nodes and again a stopping rule could be fooled into stopping the tree growth too soon. The CART strategy of growing the tree to a large size and then pruning back is the only way to determine the right sized tree.

Testing And Self-Testing

Every data mining tool should self-test automatically. By this we mean that at the beginning of an analysis you should be specifying both train and test samples and your performance summary should automatically display test results. Modern data mining tools are capable of generating models that perform brilliantly on training data while failing miserably on test data. (Data miners call such models overfit or overtrained and say that the models fail to generalize). At the very least, if this is happening with your models your tool needs to let you know loudly and clearly. You should not have to go through any cumbersome process to honestly evaluate your models.

Fortunately, the CART decision tree has a self-testing procedure built into its core and every one of our performance summaries provides both train and test data results.

Penalties on Variables

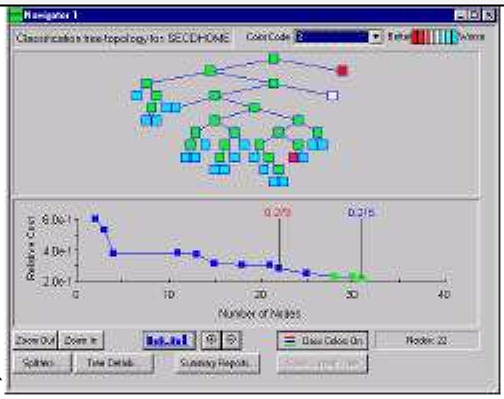
Penalties on specific types of variables were pioneered by Jerome Friedman in MARS and Salford Systems extended and broadened the concept in the CART decision tree. A penalty works by reducing the goodness-of-split score credited to a variable thus making that variable less likely to appear as the splitter in any node. An analyst may wish to impose a penalty on a variable to reflect the fact that the variable is costly to acquire or because of a preference to allow other variables to take precedence in the evolution of a tree. Recent releases of CART also allow penalties to be placed on variables with a high percentage of missing values and categorical variables with many levels. Penalties offer the analyst an added lever use to shape a tree without having to manually dictate the specifics of certain splits. Although penalties are new to decision trees we feel that they should be considered a key feature when comparing different algorithms and implementations of trees.

Case Weights

Case weights are record specific weights and can be unique to each record. Case weights are separate form class weights and priors and both kinds of weights may be active in an analysis. There will be circumstances in which having the flexibility of case specific weights can be very helpful in an analysis. If the distribution of the predictors is expected to be different between the training and scoring samples case specific weights can allow the analyst to tune the tree to the future data. For example, if we knew that the age, education, and income mix of the prospects we intended to target were somewhat dfferent than that found in the available training data then case weights could be used to adapt the tree to the expected target population. Here the important point is that quite a few major decision trees do not support case weights thereby limiting the analyst.

See the next pages for a summary table of the points made here

CART®	Other Major Trees	Notes
Data use (rows)		
♦ Uses ALL data to split every node unless sampling is explicitly requested	♦ Some data mining packages will not use all of your data regardless of the settings you choose ♦ In one major package a sample of no more than 32,768 records is used to split node without warning the user ♦ There is no option to use more data regardless of the data set size	♦ Some major decision tree packages are unable to use all the rows of data available, regardless of what the vendor claims. The major limitation occurs when a tree always uses sub-sampling when searching for the best splitter of a node. ♦ With node subsampling, a sample is used to determine the best split and then ALL data is partitioned using this split. This creates the illusion that all data is being used but splits are potentially being determined by small fractions of the data
Data use (columns)		
♦ CART is capable of handling very large numbers of columns	♦ Some major packages are very limited in practice. Limits appear as low as 500 and 1,500 columns depending on the package.	♦ Some decision tree packages are unable to effectively handle problems with more than 1,500 columns. This is manifested in exceedingly long run times and crashes. ♦ CART easily handles problems with up to 8,000 columns and custom adaptations to 1 million columns can be obtained.
Splitting Rules		
♦ CART has a rich set of splitting rules available including: ❖ Gini ❖ Entropy ❖ Twoing ❖ Symgini ❖ Ordered twoing ❖ Power-modified ❖ Linear combinations	♦ Chi-square (CHAID) ♦ Entropy ♦ Gini	♦ The CART splitting rules include a broad range of options and span very different tree growing strategies. This allows an analyst to conduct substantially different analyses and select the tree that is truly best for the problem. In our extensive consulting experience we have found the broad range of rules to be essential in getting the best results. ♦ Most other decision tree packages cover a very limited number of splitting rules and several major packages include only one splitting method. No single splitting method is best for all problems even within the same subject matter and database.
Categorical or Nominal Predictors		
♦ Allows thousands of levels	Limited to very few levels (examples:100, 128)	♦ Web mining and bioinformatics require handling thousands of levels ♦
Priors		
♦ Priors allow automatic re-weighting of the data to balance very unequal sample sizes ♦ Priors in CART can also be used to focus on a class, or to down-weight its significance	♦ Most data mining packages do not permit any automatic re-weighting of class sizes. Instead they suggest that you manually ensure that the number of non-responders be kept similar to the number of responders. ♦ A select few decision trees allow automatic balancing but do not allow intentional re-weighting to favor or disfavor a class	♦ Sophisticated analysis is impossible without having complete control over PRIORS. By varying priors one can improve tree performance and conduct HotSpot analyses to identify highly concentrated groups of records (responders, bankrupts, effective compounds).
Weights		
❖ Individual records can be given case specific weights	♦ Many data mining trees are incapable of using case weights.	♦ Case weights are especially helpful if the distribution of predictors is expected to differ between the training and scoring data sets.

Costs		
♦ If some mistakes are worse than others this information can be actively used by CART to tune the tree and ensure that inevitable errors are as innocuous as possible ❖ Costs used to grow and prune tree ❖ Option to grow without costs but prune using costs	♦ Most data mining packages are incapable of making effective use of costs of misclassification during the building of the tree. ♦ Instead costs are used passively after tree is grown to report results	♦ Incorporating costs of misclassification into the growing of a tree is vital to properly reflect the costs and generate an optimal tree. A tree actively grown using costs will almost always be quite different than a tree which ignores those costs during growing ♦ Data mining packages that are not capable of using the costs to grow the tree are able only to report costs of a standard tree.
Testing		
♦ CART is based on a mandatory self-test methodology and offers several means of testing. CART trees automatically report performance on test data ❖ Cross-validation ❖ Separate test data identified by flag, randomly selected on the fly or resident in a separate file	♦ Many decision tree packages do not offer cross-validation as a test method ♦ Some decision trees do not offer any automated self-testing and are very difficult to test manually	♦ Data mining methods are often capable of developing models that are nearly perfect on the training data. Such models are entirely useless as their performance on new or test data can be dreadful. Reliable data mining requires testing to certify models built and CART provides such test results automatically. ♦ Cross-validation for decision trees is a major CART innovation making it possible to conduct reliable tests even in the face of sparse data. Cross-validation is vital when working with small data sets or small numbers of records of interest. If there only a few hundred cases of interest (e.g. examples of fraud) cross-validation may be the only way to test regardless of the overall database size
Visualization: Tree Display		
♦ CART uses an innovative display to give access to the tree at several levels ❖ A high level overview of the tree displays its size, shape, accuracy, and location of interesting nodes ❖ The first level drill down displays all splitters on the tree diagram	♦ Other data mining packages have less effective decision tree visualization	
Overall tree summary reports		
♦ CART offers several overall summaries of the tree including: ♦ Gains Charts ♦ Variable Importance Lists ♦ Summaries of all trees developed in an analysis session, including accuracy, size of tree, growing method, etc.	♦ Variable importance is a unique CART innovation which does not appear in several major tree packages ♦ Some packages do not include a gains chart ♦ Some packages do not summarize the analysis session	♦ Summary reports are essential for assessing a tree ♦ Session summaries help the analyst organize work