Lecture 3

Classification Trees

Classification and Regression Trees

If one had to choose a classification technique that performs well across a wide range of situations without requiring much effort from the application developer while being readily understandable by the end-user a strong contender would be the tree methodology developed by Brieman, Friedman, Olshen and Stone (1984). We will discuss this classification procedure first, then in later sections we will show how the procedure can be extended to prediction of a continuous dependent variable. The program that Brieman et. al. created to implement these procedures was called CART for Classification And Regression Trees.

Classification Trees

There are two key ideas underlying classification trees. The first is the idea of recursive partitioning of the space of the independent variables. The second is of pruning using validation data. In the next few sections we describe recursive partitioning, subsequent sections explain the pruning methodology.

Recursive Partitioning

Let us denote the dependent (categorical) variable by y and the independent variables by x1, x2, x3 …xp. Recursive partitioning divides up the p dimensional space of the x variables into non-overlapping rectangles. This division is accomplished recursively. First one of the variables is selected, say xi and a value of xi, say si is chosen to split the p dimensional space into two parts: one part is the p-dimensional hyper-rectangle that contains all the points with $xi \leq si$ and the other part is the hyper-rectangle with all the points with $xi > ci$. Then one of these two parts is divided in a similar manner by choosing a variable again (it could be xi or another variable) and a split value for the variable. This results in three rectangular regions. (From here onwards we refer to hyper-rectangles simply as rectangles.) This process is continued so that we get smaller and smaller rectangles. The idea is to divide the entire x-space up into rectangles such that each rectangle is as homogenous or 'pure' as possible. By 'pure' we mean containing points that belong to just one class. (Of course, this is not always possible, as there may be points that belong to different classes but have exactly the same values for every one of the independent variables.) Let us illustrate recursive partitioning with an example.

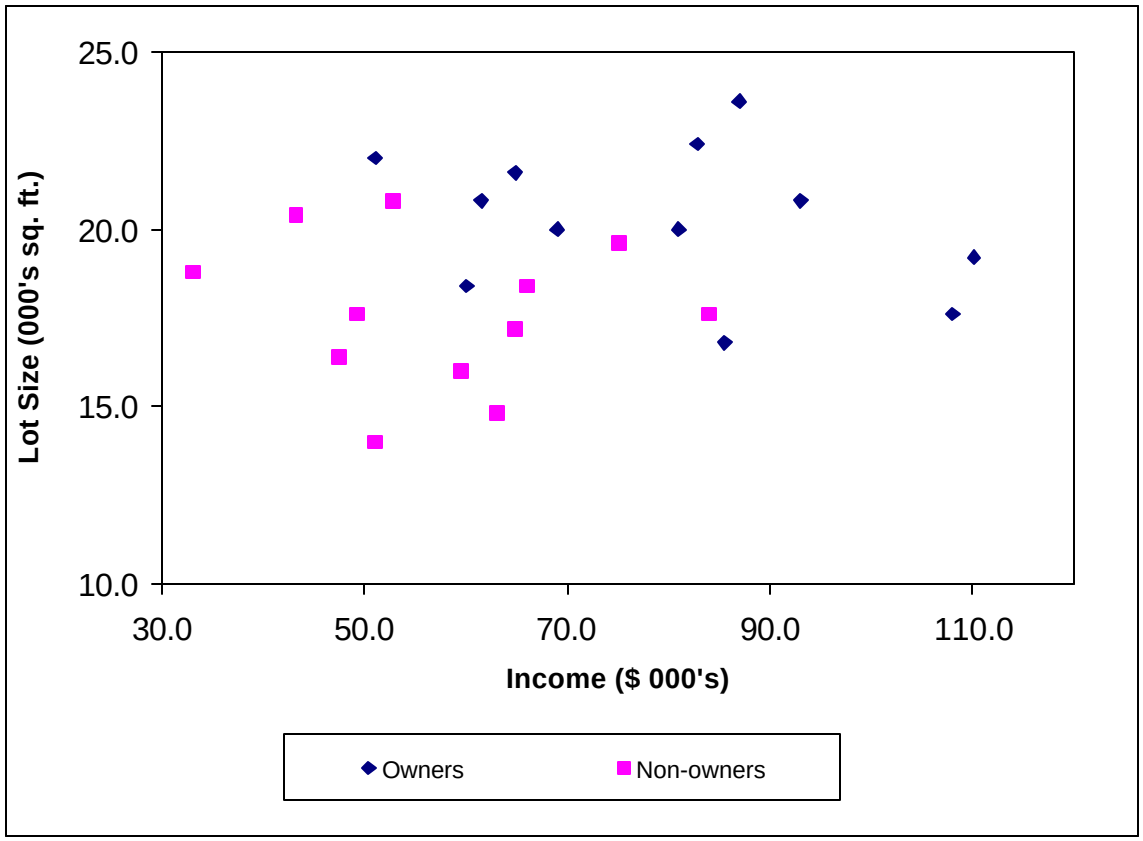
Example 1 (Johnson and Wichern)
A riding-mower manufacturer would like to find a way of classifying families in a city into those that are likely to purchase a riding mower and those who are not likely to buy one. A pilot random sample of 12 owners and 12 non-owners in the city is undertaken. The data are shown in Table I and plotted in Figure 1 below. The independent variables here are Income (x1) and Lot Size (x2). The categorical y variable has two classes: owners and non-owners.

Table 1

Observation	Income ($ 000's)	Lot Size (000's sq. ft.)	Owners=1, Non-owners=2
1	60	18.4	1
2	85.5	16.8	1
3	64.8	21.6	1
4	61.5	20.8	1
5	87	23.6	1

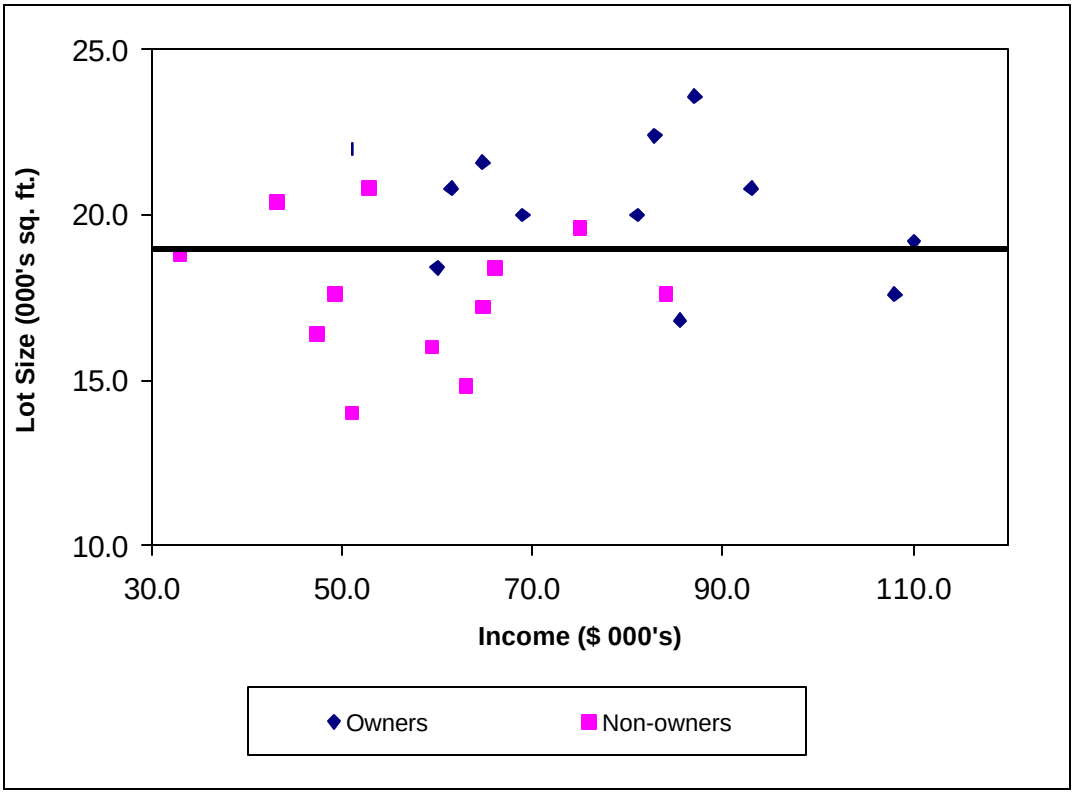
6	110.1	19.2	1
7	108	17.6	1
8	82.8	22.4	1
9	69	20	1
10	93	20.8	1
11	51	22	1
12	81	20	1
13	75	19.6	2
14	52.8	20.8	2
15	64.8	17.2	2
16	43.2	20.4	2
17	84	17.6	2
18	49.2	17.6	2
19	59.4	16	2
20	66	18.4	2
21	47.4	16.4	2
22	33	18.8	2
23	51	14	2
24	63	14.8	2

Figure 1



If we apply CART to this data it will choose x2 for the first split with a splitting value of 19. The (x1,x2) space is now divided into two rectangles, one with the Lot Size variable, x2 ≤ 19 and the other with x2> 19. See Figure 2.

Figure 2



Notice how the split into two rectangles has created two rectangles each of which is much more homogenous than the rectangle before the split. The upper rectangle contains points that are mostly owners (9 owners and 3 non-owners) while the lower rectangle contains mostly non-owners (9 non-owners and 3 owners).

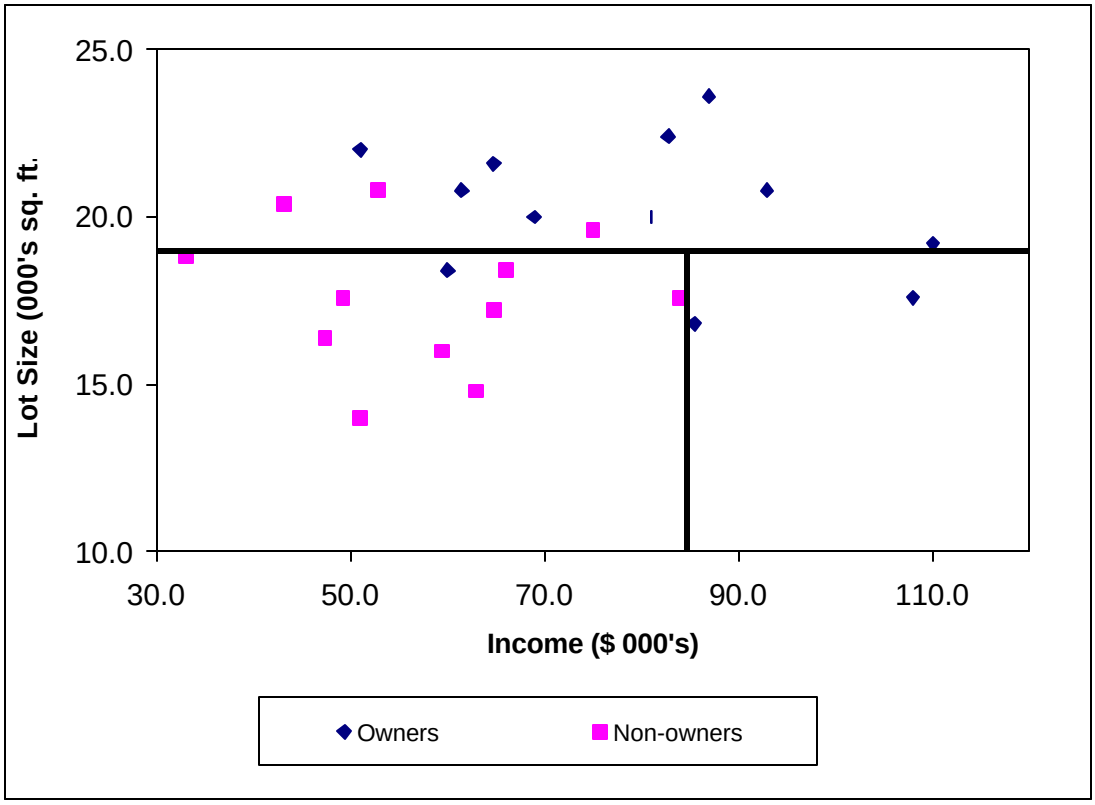
How did CART decide on this particular split? It examined each variable and all possible split values for each variable to find the best split. What are the possible split values for a variable? They are simply the mid-points between pairs of consecutive values for the variable. The possible split points for x1 are {38.1, 45.3, 50.1, … , 109.5} and those for x2 are {14.4, 15.4, 16.2, … , 23}. These split points are ranked according to how much they reduce impurity (heterogeneity of composition). The reduction in impurity is defined as the impurity of the rectangle before the split

minus the sum of the impurities for the two rectangles that result from a split. There are a number of ways we could measure impurity. We will describe the most popular measure of impurity: the Gini index. If we denote the classes by k, k=1, 2, … , C, where C is the total number of classes for the y variable, the Gini impurity index for a rectangle A is defined by

$I(A) = 1 - \sum_{k=1}^{C} p_k^2$ $I(A) = 1 - \sum_{k=1}^{C} p_k^2$ where p_k p_k is the fraction of observations in rectangle A that belong to class k. Notice that I(A) = 0 if all the observations belong to a single class and I(A) is maximized when all classes appear in equal proportions in rectangle A. Its maximum value is (C-1)/C

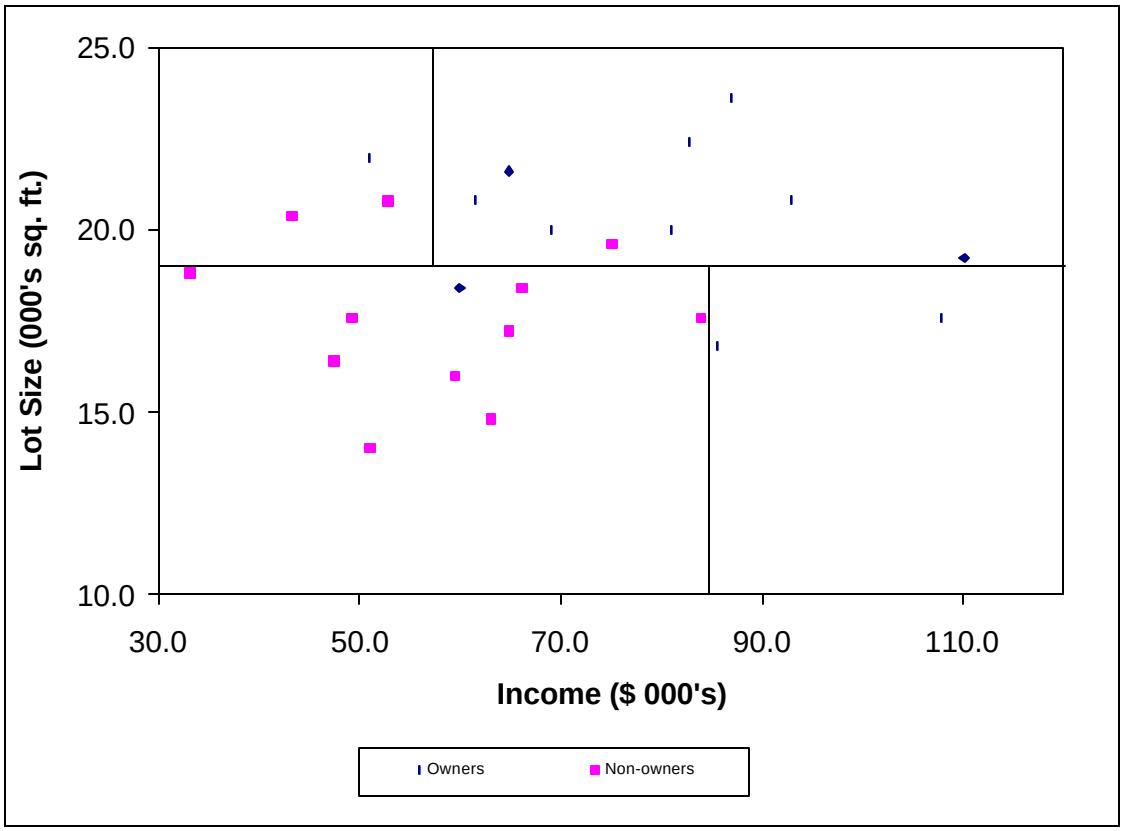
The next split is on the Income variable, x1 at the value 84.75. Figure 3 shows that once again the CART procedure has astutely chosen to split a rectangle to increase the purity of the resulting rectangles. The left lower rectangle which contains data points with x1≤ 84.75 and x2≤ 19 has all but one points that are non-owners; while the right lower rectangle which contains data points with x1 > 84.75 and x2≤ 19 consists exclusively of owners.

Figure 3



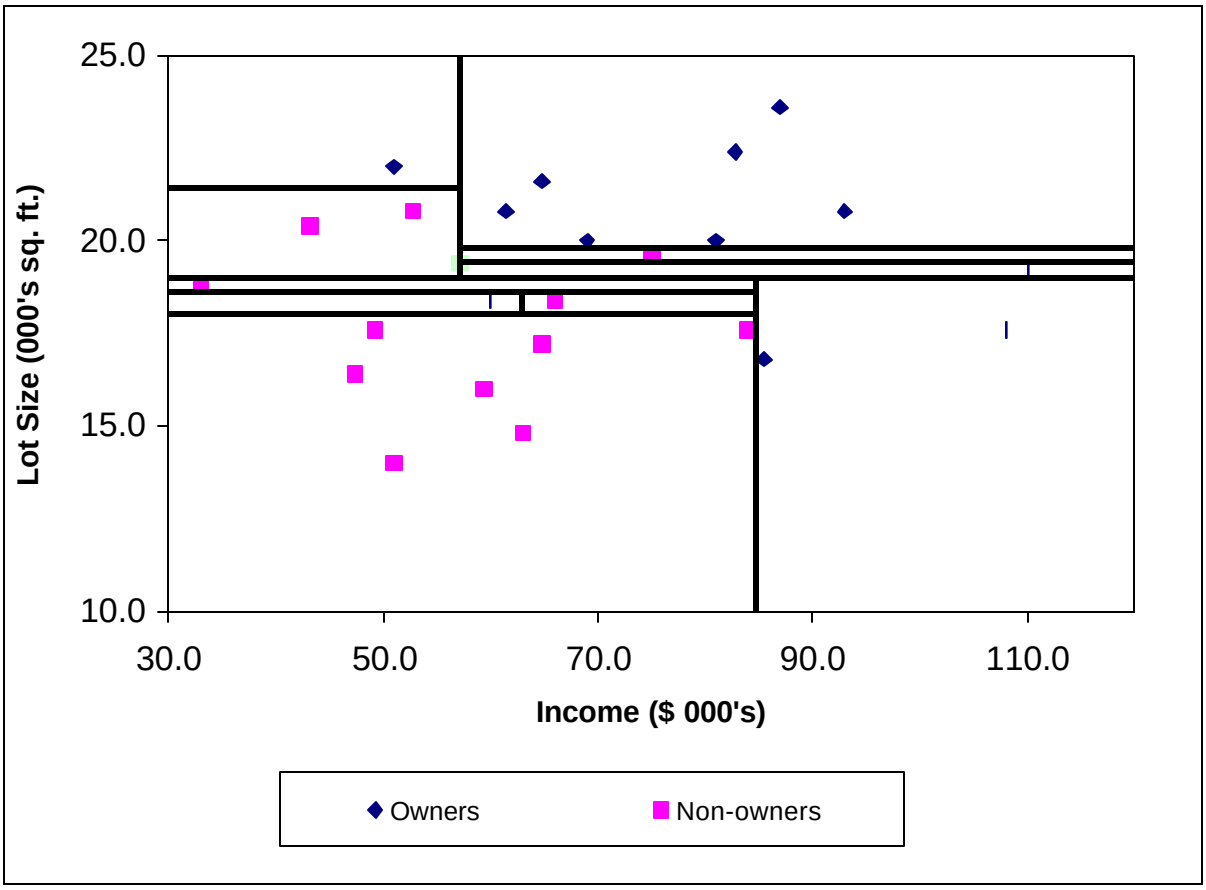
The next split is shown below:

Figure 4



We can see how the recursive partitioning is refining the set of constituent rectangles to become purer as the algorithm proceeds. The final stage of the recursive partitioning is shown in Figure 5.

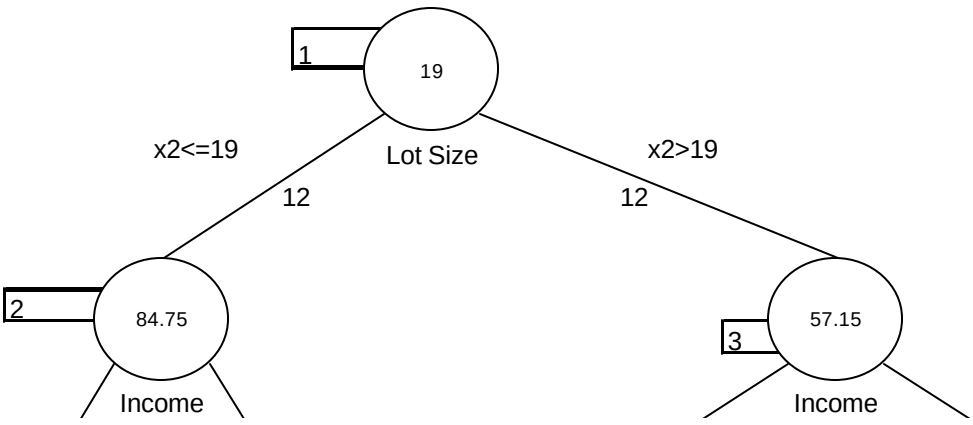
Figure 5



Notice that now each rectangle is pure – it contains data points from just one of the two classes.

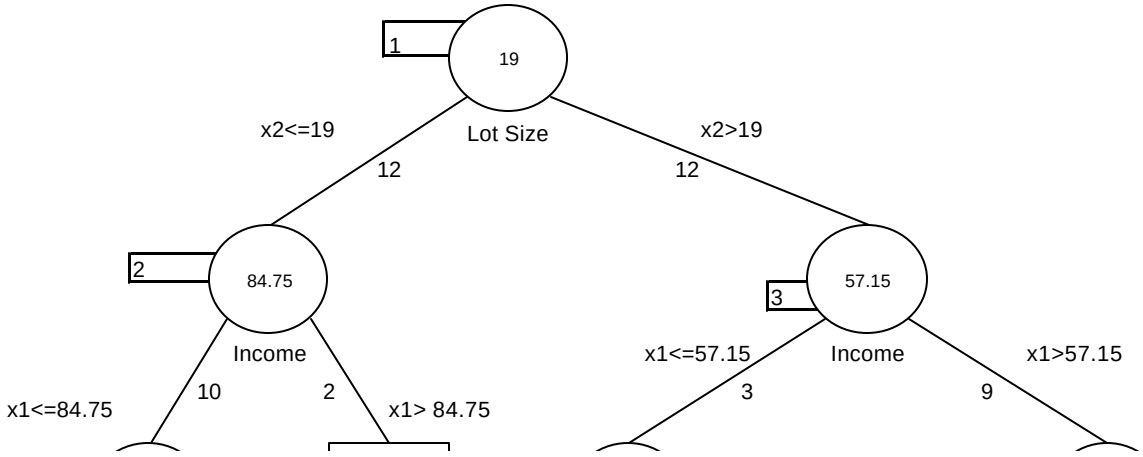
The reason the method is called a classification tree algorithm is that each split can be depicted as a split of a node into two successor nodes. The first split is shown as a branching of the root node of a tree in Figure 6.

Figure 6



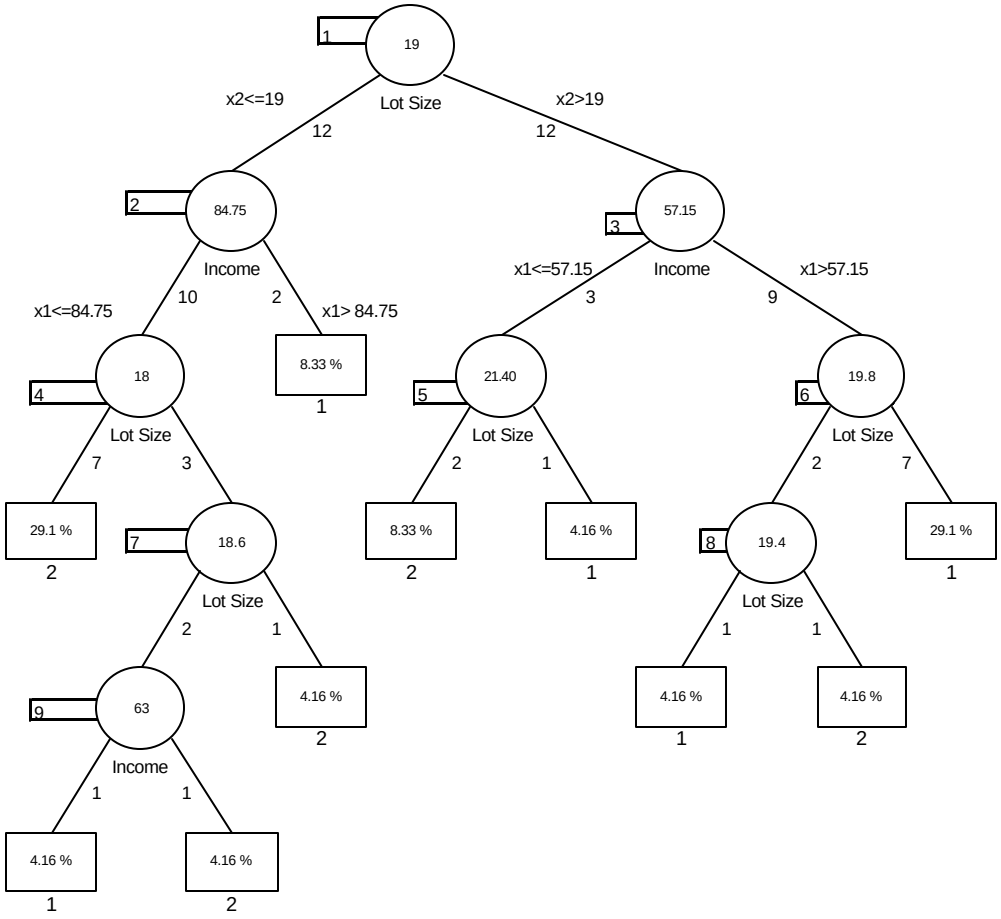
The tree representing the first three splits is shown in Figure 7 below.

Figure 7



The full tree is shown in Figure 8 below. We have represented the nodes that have successors by circles. The numbers inside the circle are the splitting values and the name of the variable chosen for splitting at that node is shown below the node. The numbers on the left fork at a decision node shows the number of points in the decision node that had values less than or equal to the splitting value while the number on the right fork shows the number that had a greater value. These are called decision nodes because if we were to use a tree to classify a new observation for which we knew only the values of the independent variables we would 'drop' the observation down the tree in such a way that at each decision node the appropriate branch is taken until we get to a node that has no successors. Such terminal nodes are called the leaves of the tree. Each leaf node corresponds to one of the final rectangles into which the x-space is partitioned and is depicted as a rectangle-shaped node. When the observation has dropped down all the way to a leaf we can predict a class for it by simply taking a 'vote' of all the training data that belonged to the leaf when the tree was grown. The class with the highest vote is the class that we would predict for the new observation. The number below the leaf node is the class with the most votes in the rectangle. The % value in a leaf node shows the percentage of the total number of training observations that belonged to that node. It is useful to note that the type of trees grown by CART (called binary trees) have the property that the number of leaf nodes is exactly one more than the number of decision nodes.

Figure 8



Pruning

The second key idea in the CART procedure, that of using the validation data to prune back the tree that is grown from the training data using independent validation data, was the real innovation. Previously, methods had been developed that were based on the idea of recursive partitioning but they had used rules to prevent the tree from growing excessively and over-fitting the training data. For example, CHAID (Chi-Squared Automatic Interaction Detection) is a recursive partitioning method that predates CART by several years and is widely used in database marketing applications to this day. It uses a well-known statistical test (the chi-square test for independence) to assess if splitting a node improves the purity by a statistically significant amount. If the test does not show a significant improvement the split is not carried out. By contrast, CART uses validation data to prune back the tree that has been deliberately overgrown using the training data.

The idea behind pruning is to recognize that a very large tree is likely to be over-fitting the training data. In our example, the last few splits resulted in rectangles with very few points (indeed four rectangles in the full tree have just one point). We can see intuitively that these last splits are likely to be simply capturing noise in the training set rather than reflecting patterns that would occur in future data such as the validation data. Pruning consists of successively selecting a

decision node and re-designating it as a leaf node (thereby lopping off the branches extending beyond that decision node (its "subtree") and thereby reducing the size of the tree).The pruning process trades off misclassification error in the validation data set against the number of decision nodes in the pruned tree to arrive at a tree that captures the patterns but not the noise in the training data. It uses a criterion called the "cost complexity" of a tree to generate a sequence of trees which are successively smaller to the point of having a tree with just the root node. (What is the classification rule for a tree with just one node?). We then pick as our best tree the one tree in the sequence that gives the smallest misclassification error in the validation data.

The cost complexity criterion that CART uses is simply the misclassification error of a tree (based on the validation data) plus a penalty factor for the size of the tree. The penalty factor is based on a parameter, let us call it α, that is the per node penalty. The cost complexity criterion for a tree is thus $Err(T) + \alpha |L(T)|$ where $Err(T)$ is the fraction of validation data observations that are misclassified by tree T, $L(T)$ is the number of leaves in tree T and α is the per node penalty cost: a number that we will vary upwards from zero. When $\alpha = 0$ there is no penalty for having too many nodes in a tree and the best tree using the cost complexity criterion is the full grown unpruned tree. When we increase α to a very large value the penalty cost component swamps the misclassification error component of the cost complexity criterion function and the best tree is simply the tree with the fewest leaves, namely the tree with simply one node. As we increase the value of α from zero at some value we will first encounter a situation where for some tree T_1 formed by cutting off the subtree at a decision node we just balance the extra cost of increased misclassification error (due to fewer leaves) against the penalty cost saved from having fewer leaves. We prune the full tree at this decision node by cutting off its subtree and redesignating this decision node as a leaf node. Let's call this tree T_1. We now repeat the logic that we had applied previously to the full tree, with the new tree T_1 by further increasing the value of α. Continuing in this manner we generate a succession of trees with diminishing number of nodes all the way to the trivial tree consisting of just one node.

From this sequence of trees it seems natural to pick the one that gave the minimum misclassification error on the validation data set. We call this the Minimum Error Tree.

Let us use the Boston Housing data to illustrate. Shown below is the output that XLMiner generates when it is using the training data in the tree growing phase of the algorithm

Training Log

Growing the Tree	
#Nodes	Error
0	36.18
1	15.64
2	5.75
3	3.29
4	2.94
5	1.88
6	1.42
7	1.26
8	1.2
9	0.63
10	0.59
11	0.49
12	0.42
13	0.35
14	0.34
15	0.32
16	0.25
17	0.22
18	0.21
19	0.15
20	0.09
21	0.09
22	0.09
23	0.08
24	0.05
25	0.03
26	0.03
27	0.02
28	0.01
29	0
30	0

Training Misclassification Summary

Classification Confusion Matrix			
	Predicted Class		
Actual Class	1	2	3
1	59	0	0
2	0	194	0
3	0	0	51

Error Report			
Class	# Cases	# Errors	% Error
1	59	0	0.00
2	194	0	0.00
3	51	0	0.00
Overall	304	0	0.00

These are cases in the training data

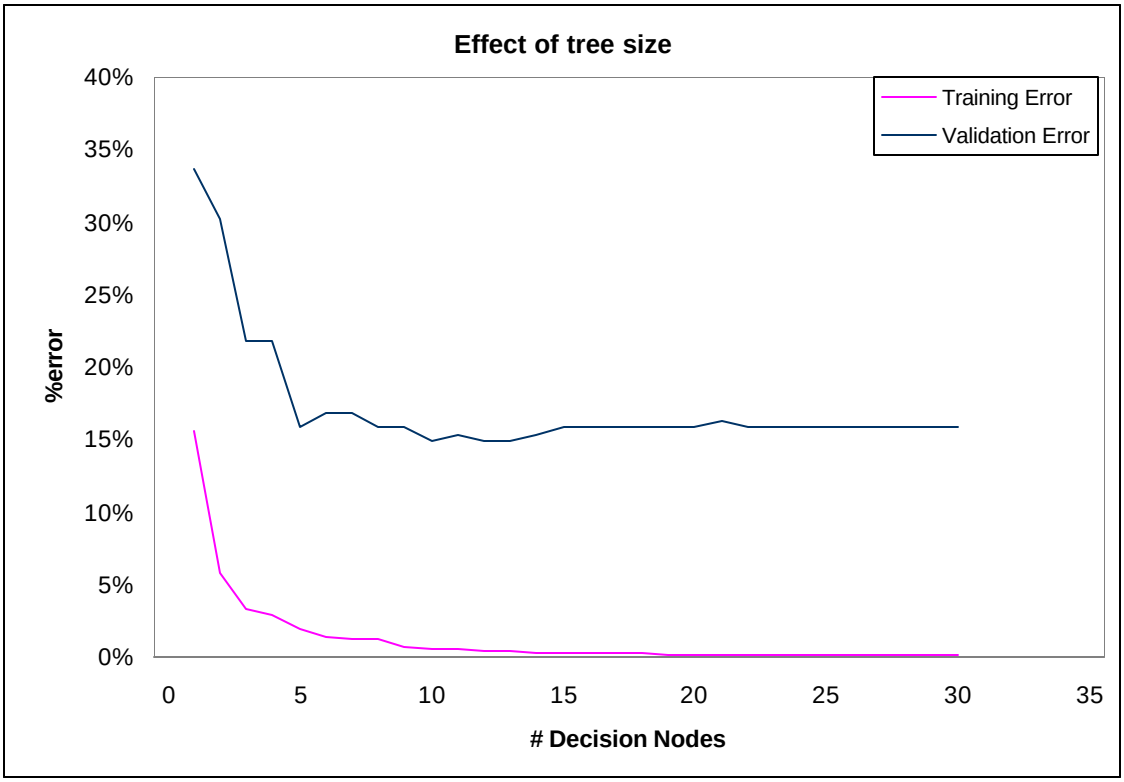
The top table logs the tree-growing phase by showing in each row the number of decision nodes in the tree at each stage and the corresponding (percentage) misclassification error for the training data applying the voting rule at the leaves. We see that the error steadily decreases as the number of decision nodes increases from zero (where the tree consists of just the root node) to thirty. The error drops steeply in the beginning, going from 36% to 3% with just an increase of decision nodes from 0 to 3. Thereafter the improvement is slower as we increase the size of the tree. Finally we stop at a full tree of 30 decision nodes (equivalently, 31 leaves) with no error in the training data, as is also shown in the confusion table and the error report by class.

The output generated by XLMiner during the pruning phase is shown below.

# Decision Nodes	Training Error	Validation Error			
30	0.00%	15.84%			
29	0.00%	15.84%			
28	0.01%	15.84%			
27	0.02%	15.84%			
26	0.03%	15.84%			
25	0.03%	15.84%			
24	0.05%	15.84%			
23	0.08%	15.84%			
22	0.09%	15.84%			
21	0.09%	16.34%			
20	0.09%	15.84%			
19	0.15%	15.84%			
18	0.21%	15.84%			
17	0.22%	15.84%			
16	0.25%	15.84%			
15	0.32%	15.84%			
14	0.34%	15.35%			
13	0.35%	14.85%			
12	0.42%	14.85%			
11	0.49%	15.35%			
10	0.59%	14.85%	<-- Minimum Error Prune	Std. Err.	0.02501957
9	0.63%	15.84%			
8	1.20%	15.84%			
7	1.26%	16.83%			
6	1.42%	16.83%			
5	1.88%	15.84%	<-- Best Prune		
4	2.94%	21.78%			
3	3.29%	21.78%			
2	5.75%	30.20%			
1	15.64%	33.66%			

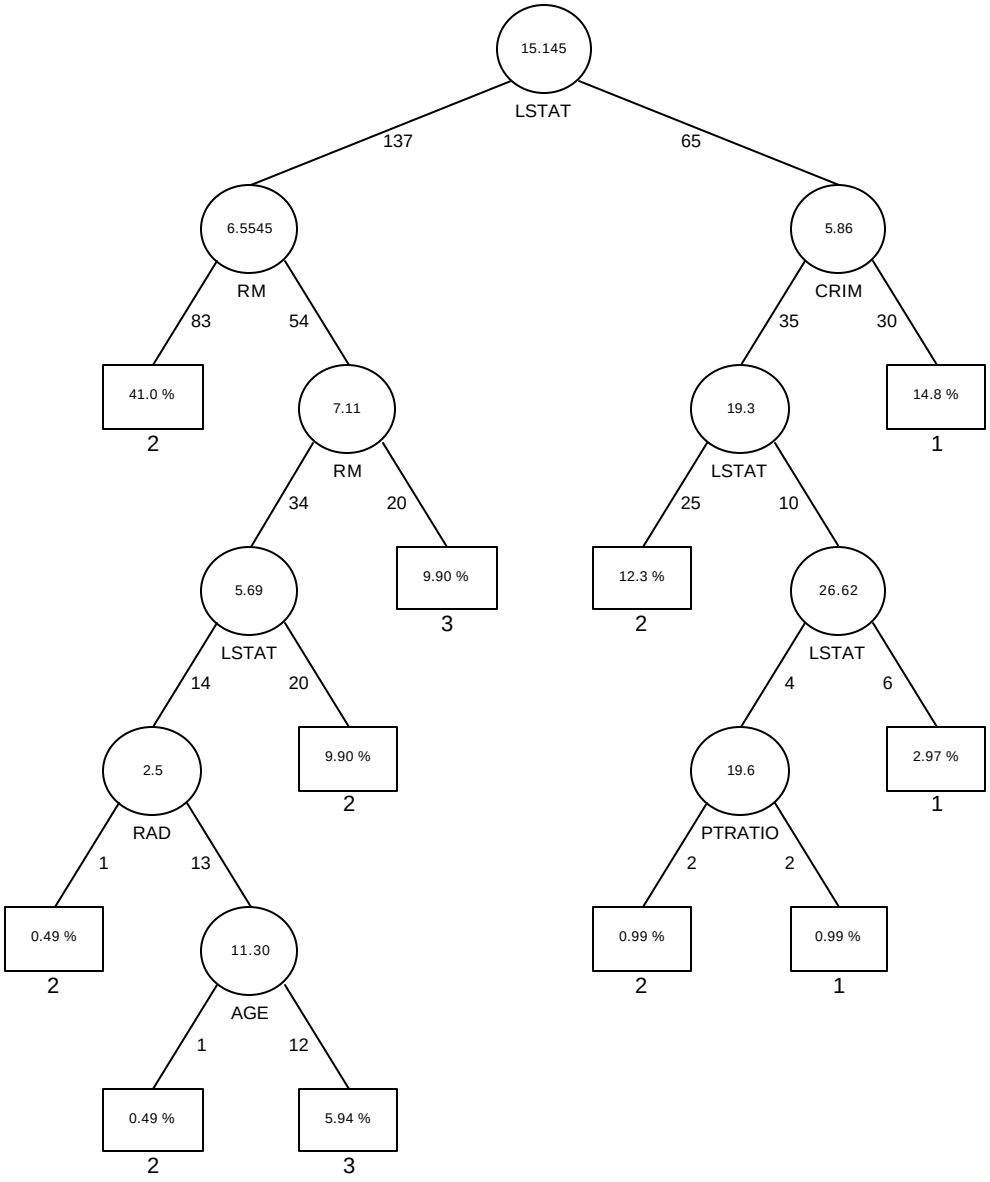
Notice now that as the number of decision nodes decreases the error in the validation data has a slow decreasing trend (with some fluctuation) up to a 14.85% error rate for the tree with 10 nodes. This is more readily visible from the graph below. Thereafter the error increases, going up

sharply when the tree is quite small. The Minimum Error Tree is selected to be the one with 10 decision nodes (why not the one with 13 decision nodes?)



This Minimum Error tree is shown in Figure 9.

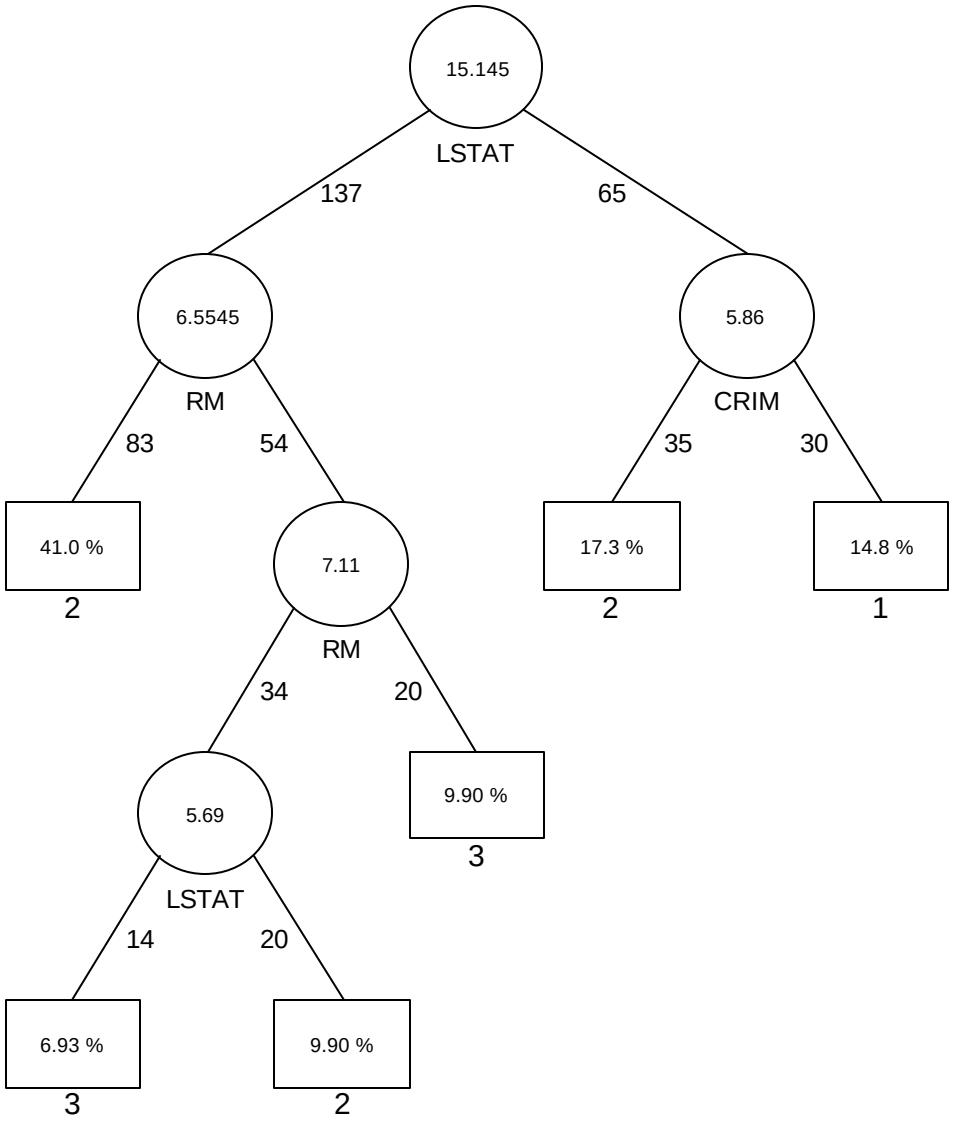
Figure 9



You will notice that the XLMiner output from the pruning phase highlights another tree besides the Minimum Error Tree. This is the Best Pruned Tree, the tree with 5 decision nodes. The reason this tree is important is that it is the smallest tree in the pruning sequence that has an error that is within one standard error of the Minimum Error Tree. The estimate of error that we get from the validation data is just that: it is an estimate. If we had had another set of validation data the minimum error would have been different. The minimum error rate we have computed can be viewed as an observed value of a random variable with standard error (estimated standard deviation) equal to $\sqrt{E_{\min}(1-E_{\min})/N_{val}}$ where $E_{\min}$ is the error rate (as a fraction) for the

minimum error tree and N_{val} is the number of observations in the validation data set. For our example $E_{\min}$ = 0.1485 and N_{val} = 202, so that the standard error is 0.025. The Best Pruned Tree is shown in Figure 10.

Figure 10



We show the confusion table and summary of classification errors for the Best Pruned Tree below.

Validation Misclassification Summary

Classification Confusion Matrix			
	Predicted Class		
Actual Class	1	2	3
1	25	10	0
2	5	120	9
3	0	8	25

Error Report			
Class	# Cases	# Errors	% Error
1	35	10	28.57
2	134	14	10.45
3	33	8	24.24
Overall	202	32	15.84

It is important to note that since we have used the validation data in classification, strictly speaking it is not fair to compare the above error rates directly with the other classification procedures that use only the training data to construct classification rules. A fair comparison would be to partition the training data (TDother) for other procedures into a further partition into training (TDtree) and validation data (VDtree) for classification trees. The error rate for the classification tree created from using TDtree to grow the tree and then VDtree to prune it can now be compared using the validation data used by the other classifiers (VDother) as new hold out data..

Classification Rules from Trees

One of the reasons tree classifiers are very popular is that they provide easily understandable classification rules (at least if the trees are not too large). Each leaf is equivalent to a classification rule. For example, the upper left leaf in the Best Pruned Tree above, gives us the rule:

IF(LSTAT ≤ 15.145) AND (ROOM ≤ 6.5545) THEN CLASS = 2

Such rules are easily explained to managers and operating staff compared to outputs of other classifiers such as discriminant functions. Their logic is certainly far more transparent than that of weights in neural networks!

The tree method is a good off-the-shelf choice of classifier

We said at the beginning of this chapter that classification trees require relatively little effort from developers. Let us give our reasons for this statement. Trees need no tuning parameters. There is no need for transformation of variables (any monotone transformation of the variables will give the same trees). Variable subset selection is automatic since it is part of the split selection; in our example notice that the Best Pruned Tree has automatically selected just three variables (LSTAT, RM and CRIM) out of the set thirteen variables available. Trees are also intrinsically robust to outliers as the choice of a split depends on the ordering of observation values and not on the absolute magnitudes of these values.

Finally the CART procedure can be readily modified to handle missing data without having to impute values or delete observations with missing values. The method can also be extended to

incorporate an importance ranking for the variables in terms of their impact on quality of the classification.

Notes:

1. We have not described how categorical independent variables are handled in CART. In principle there is no difficulty. The split choices for a categorical variable are all ways in which the set of categorical values can be divided into two subsets. For example a categorical variable with 4 categories, say {1,2,3,4} can be split in 7 ways into two subsets: {1} and {2,3,4}; {2} and {1,3,4}; {3} and {1,2,4}; {4} and {1,2,3};{1.2} and {3,4}; {1,3} and {2,4}; {1,4} and {2,3}. When the number of categories is large the number of splits becomes very large. XLMiner supports only binary categorical variables (coded as numbers). If you have a categorical independent variable that takes more than two values, you will need to replace the variable with several dummy variables each of which is binary in a manner that is identical to the use of dummy variables in regression.

2. There are several variations on the basic recursive partitioning scheme described above. A common variation is to permit splitting of the x variable space using straight lines (planes for p =3 and hyperplanes for p> 3) that are not perpendicular to the co-ordinate axes. This can result in a full tree that is pure with far fewer nodes particularly when the classes lend themselves to linear separation functions. However this improvement comes at a price. First, the simplicity of interpretation of classification trees is lost because we now split on the basis of weighted sums of independent variables where the weights may not be easy to interpret. Second the important property remarked on earlier of the trees being invariant to monotone transformations of independent variables is no longer true.

3. Besides CHAID, another popular tree classification method is ID3 (and its successor C4.5). This method was developed by Quinlan a leading researcher in machine learning and is popular with developers of classifiers who come from a background in machine learning.

As usual for regression we denote the dependent (categorical) variable by y and the independent variables by x1, x2, x3 …xp. Both key ideas underlying classification trees carry over with small modifications for regression trees..