

## Classification Trees

Classification trees (also called decision trees) are used as classifiers. We suppose that data comes in records of the form

$$(\mathbf{x}, y) = (x_1, x_2, \dots, x_k, y)$$

where  $k$  is fixed and  $y$  is a variable of particular interest. For example  $\mathbf{x}$  may be customer information for a credit card customer, and  $y$  a measure of how credit worthy the customer is. Given a new instance of  $\mathbf{x}$  we wish to be able to predict the corresponding  $y$ . This is called regression when  $y$  is continuous and classification when  $y$  is a discrete. We call  $\mathbf{x}$  the independent variable (input) and  $y$  the dependent variable (response or output).

### Variable Types

A variable taking values in  $[a, b]$  is continuous. A variable taking values in an ordered set  $\{a, a+1, \dots, b-1, b\}$  is ordinal. A variable taking values from an unordered set  $\{x, y, z, \dots\}$  is categorical. A special case of categorical variable is the binary or Boolean variable, which takes values in  $\{1, 0\}$  or equivalently  $\{\text{True}, \text{False}\}$ . Ordinal and categorical variables are collectively called discrete.

Decision trees treat discrete and continuous input variables differently. For the moment we assume that all the inputs are discrete; we will discuss how to deal with continuous inputs later. Note that we can always approximate a continuous variable by a discrete variable, by splitting the range  $[a, b]$  into disjoint blocks  $[c_1, c_2], (c_2, c_3], \dots$ , and using the ordinal variable defined by which block the original variable is in.

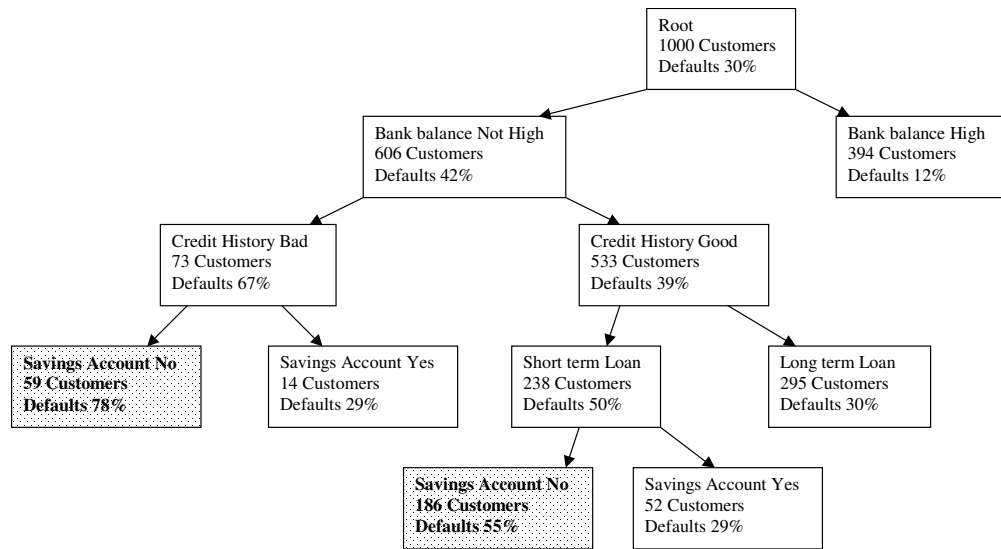
In classification the response variable is always categorical. We can construct a decision tree when  $y$  is continuous – called a regression tree – but we consider only the categorical case here.

### Growing the Tree

Suppose we have the following variables, for bank customers with personal loans.

Variable	Name	Values
$x_1$	Bank balance	High, Low, Negative, Don't Know
$x_2$	Credit history	Good, Bad
$x_3$	Savings account	Yes, No
$x_4$	Term of Loan	Long, Short
$y$	Defaults	Yes, No

This example is taken from Giudici [1] Chapter 11. In the original there were many more variables, but these were the ones that turned out to be important. A decision tree for classifying  $y$  might take the following form



At each node the tree splits according to the value of a single variable. The splits are chosen so that the subnodes are more *homogeneous*. In this case we are trying to separate out those customers who default. The final nodes are called *leaves*. By tracing the splits or decisions that lead to each leaf we obtain classification rules. For example

78% of Customers without a high bank balance, with a bad credit history and with no savings account, default.

55% of Customers without a high bank balance, with a good credit history, with no savings account and taking a short term loan, default.

The leaves themselves are classified according to the most common value of  $y$  in that leaf. Here the two shaded boxes are classified as defaulters.

In general, though not always, we use binary splits as they are more computationally tractable, and a binary tree is always capable of reproducing the nodes in a ternary or higher order tree.

There are different types of decision tree. Each uses a different rule for deciding splits. Let  $s(i)$  be the size of node  $i$ , and suppose we have a measure for the *impurity*  $I(i)$  of node  $i$ , then the gain in purity made by splitting node  $i$  into nodes  $i_0$  and  $i_1$  is

$$\text{Gain}(i; i_0, i_1) = I(i) - (I(i_0) \cdot p(i_0 | i) + I(i_1) \cdot p(i_1 | i))$$

where  $p(i_0 | i) = s(i_0)/s(i)$  and  $p(i_1 | i) = s(i_1)/s(i)$  are the proportion of records assigned to each subnode. The tree is grown recursively: at each step we choose which node to split and the variable to split on in order to maximise the Gain. We continue to do this until all nodes are pure enough, according to some stopping rule.

The following impurity measures are all used

**Gini impurity**

Used by the CART algorithm (Classification and Regression Trees). Suppose  $y$  takes on values in  $\{1, 2, \dots, m\}$ , and let  $f(i, j)$  = relative frequency of value  $j$  in node  $i$ . That is,  $f(i, j)$  is the proportion of records assigned to node  $i$  for which  $y = j$ .

$$I_G(i) = 1 - \sum_{j=1}^m f(i, j)^2$$

This simplifies somewhat in the case  $m = 2$ .

**Entropy**

Used by the C4.5 and C5.0 algorithms. This measure is based on the concept of entropy used in information theory.

$$I_E(i) = -\sum_{j=1}^m f(i, j) \log f(i, j)$$

**Misclassification rule**

The modal value  $\text{mode}(i)$  of node  $i$  is the most common value of the dependent variable  $y$  at that node

$$I_M(i) = 1 - f(i, \text{mode}(i))$$

Although seemingly sensible, this is a poor rule, for reasons explained in Section 3.1 of Therneau and Atkinson 1997.

**Variance**

If  $y$  is numerical (continuous or ordinal) then we can use the variance as a measure of the impurity of a node. Let  $y(j)$  be the value of  $y$  for record  $j$  and let  $\bar{y}(i) = \sum_{j \in i} y(j) / s(i)$  be the mean value of  $y$  over node  $i$ , then

$$I_V(i) = \frac{\sum_{j \in i} (y(j) - \bar{y}(i))^2}{s(i) - 1}$$

**Significance**

The CHAID algorithm uses a slightly different approach for splitting. For a given node  $i$  the frequency of  $j$  is  $f(i, j)$  for  $j = 1, \dots, m$ . If  $i$  were split at random into  $i0$  and  $i1$ , then we would expect  $f(i0, j)$ ,  $f(i1, j)$  and  $f(i, j)$  to be roughly the same. We can measure the deviation of the observed frequencies from the expected frequencies using

$$T(i; i0, i1) = \sum_{j=1}^m \frac{s(i0)(f(i0, j) - f(i, j))^2}{f(i, j)} + \sum_{j=1}^m \frac{s(i1)(f(i1, j) - f(i, j))^2}{f(i, j)}$$

Under the hypothesis that the split was random,  $T$  has a chi-squared distribution with  $(m - 1)$  degrees of freedom. We choose that split which gives the highest significance, or equivalently which gives the largest value of  $T$ .

To split on a categorical variable  $x$  taking values in a set  $C$ , we consider all possible binary partitions of  $C$  into sets  $A$  and  $B$ , and then divide the records into those for which  $x$  is in  $A$  and those for which  $x$  is in  $B$ . The choice of  $A$  and  $B$  which gives the best decrease in impurity is chosen. To split on a continuous or ordinal variable taking values in  $[a, b]$  or  $\{a, a+1, \dots, b\}$ , we consider partitions of the form  $[a, c]$  and  $(c, b]$  or  $\{a, \dots, c\}$  and  $\{c+1, \dots, b\}$ .

Note that all of the splitting rules above can be generalised to ternary and higher order splits.

## Interpreting the Tree

A classification tree defines a function  $F$  from the parameter space to  $\{1, 2, \dots, m\}$ . Any  $\mathbf{x} = (x_1, \dots, x_k)$  is assigned to a unique leaf of the tree, say  $i$ , according to the splitting rules. We have  $F(\mathbf{x}) = j$ , where  $j$  is the most common value of  $y$  in leaf  $i$ .

Define  $R(i)$ , the Risk for node  $i$ , to be the misclassification rate for node  $i$ . That is,  $R(i) = 1 - f(i, \text{mode}(i))$ . (This was called  $I_M$  above.)  $R(i)$  is the proportion of observations in node  $i$  for which  $F(\mathbf{x})$  is not equal to  $y$ . If  $n$  is the total number of records, then the Risk of the whole tree  $T$  is given by

$$R(T) = \sum_{i \in \text{Leaves}} R(i)s(i)/n$$

This is the overall proportion of misclassified observations.

The Gini impurity can be interpreted as the expected Risk of a node, if the classification of node  $i$  is randomised, with type  $j$  having probability  $f(i,j)$ .

## Stopping and Pruning Rules

If we let it, the tree could keep growing until each leaf was completely monotonic in  $y$  (every record had the same value), or else monotonic in  $\mathbf{x}$ . In practice this is a bad idea because it makes the tree very large and thus hard to interpret, and will reduce its accuracy as a classifier. Accordingly we use rules to stop the splitting, for example:

- Maximum number of leaves
- Maximum depth of tree
- Minimum gain (decrease in impurity)
- Minimum size of node
- Minimum significance (CHAID algorithm)

In addition to stopping rules, the CART algorithm also uses pruning. Starting with a large tree  $T$ , we choose that subtree  $S$  of  $T$  which has the smallest *complexity*, given by

$$R_\alpha(S) = R(S) + \alpha|S| \cdot R(T_0)$$

where  $|S|$  is the number of leaves of  $S$  and  $T_0$  is the root of  $T$  (so  $|T_0| = 1$ ). If  $S_1$  and  $S_2$  are subtrees of  $T$  with the same complexity, then it can be shown that one must be a subtree of the other. Thus in the case of a draw we can always choose the smaller tree. The parameter  $\alpha$  needs to be specified before hand. Larger values of  $\alpha$  produce smaller trees.

The complexity can also be used to define a stopping rule (using a small value of  $\alpha$ ):

Decreasing complexity.

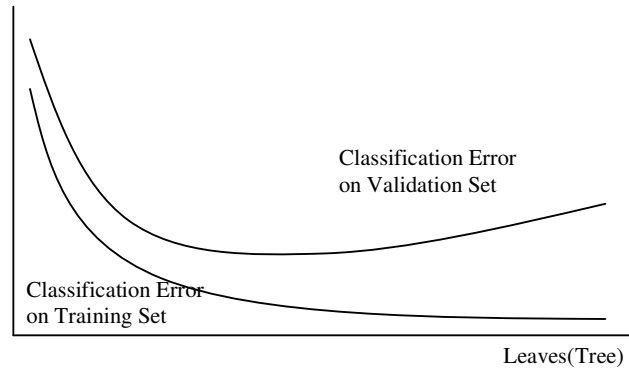
This is equivalent to requiring  $R(T_{n+1}) < R(T_n) - \alpha R(T_0)$ , where  $T_n$  is the current tree and  $T_{n+1}$  the proposed tree. (Check that for  $\alpha = 1$  this rule would prevent any splitting.)

## Optimal choice of the complexity parameter

Large trees can suffer from *overfitting*. That is, they start to fit the errors/exceptions in the data rather than the underlying trends/patterns. One way of detecting overfitting is to split the data into a *training set* and a *validation set*: the classification tree (model) is grown using the training set, then the validation set is used to estimate its true Risk. The size of the training set

depends on how much data is available, but it is not unusual for 2/3 of the available data to be used for training and the remaining 1/3 for validation.

The following is a typical graph of the classification error or Risk, for the training set and the validation set, as the number of leaves (model complexity) is allowed to grow. Clearly if we are to use the tree to classify  $x$  for which  $y$  is unknown, we would like to prune it to a size which minimises the classification error on the validation set.



A possible pruning strategy is to choose  $\alpha$  to minimise the error on the validation set. An alternative is to use Cross-Validation or Jackknifing. If we let  $T_\alpha$  be the tree obtained by minimising the complexity using complexity parameter  $\alpha$ , then we have a well defined sequence of trees indexed by  $\alpha$ , and we can choose that  $\alpha$  (or equivalently that  $T_\alpha$ ) which minimises the classification error on the validation set, or the cross-validation or jackknifing estimate of error.

### Cross-validation

Cross-validation is a very general procedure for estimating the error of a model:

1. Split the data into  $k$  sets
2. Put one set aside and fit the model with the remaining  $k - 1$  sets
3. Calculate the model fitting error (misclassification error) for the set you put aside
4. Repeat steps 2 and 3 for all  $k$  sets
5. Average the errors for each set

### Jackknife

The jackknife can be seen as a particular case of cross-validation:

1. Suppose the data set has size  $n$
2. Put one element aside and fit the model with the remaining  $n - 1$  elements
3. Calculate the model fitting error (misclassification error) for the element you put aside
4. Repeat steps 2 and 3 for all  $n$  elements
5. Average the errors for each element

Note that if we use the validation set to choose  $\alpha$ , it becomes part of the model fitting procedure, and the validation set no longer gives an independent estimate of the performance of the model.

### Weighting decisions

Suppose  $y$  is binary. In many situations we are more concerned with correctly identifying when  $y = 1$  than when  $y = 0$ . In particular this will be the case if the cost of misclassification

is very different in each case, for example when screening for cancer we like to err on the side of caution. The cost of an incorrect classification when  $y = 0$  is low, typically a further set of tests, while the cost of an incorrect classification when  $y = 1$  can be high, failure to detect the cancer in time.

There are a number of ways of skewing classification trees.

### Class weights/costs

Weight observations in class  $j$  with weight  $w(j)$ . We interpret  $w(j)$  as the cost of misclassifying an observation of class  $j$  as something else. Let  $p(i)$  be the probability that an observation chosen at random is in node  $i$ , where the probability of choosing an observation is proportional to its class weight. Then

$$p(i) \propto s(i) \sum_{j=1}^m f(i, j) w(j).$$

Let  $i_0$  and  $i_1$  be subnodes of  $i$  created by splitting it, and let  $p(i_0 | i)$  be the probability that a randomly chosen observation is in node  $i_0$  given that it is in node  $i$ , then

$$p(i_0 | i) = p(i_0) / p(i) = \frac{s(i_0) \sum_{j=1}^m f(i_0, j) w(j)}{s(i) \sum_{j=1}^m f(i, j) w(j)}$$

#### *Effect on splitting.*

We still split to maximize  $\text{Gain}(i; i_0, i_1) = I(i) - (I(i_0) * p(i_0 | i) + I(i_1) * p(i_1 | i))$ . The impurity is unchanged but the conditional probabilities have changed, which will effect the splits.

#### *Effect on interpretation and pruning.*

We classify node  $i$  as class  $j$  where  $j$  maximises  $f(i, j) w(j)$ . Let the class of node  $i$  be  $\tau(i)$  (which is just  $\text{mode}(i)$  when there are no weights), then the risk for node  $i$  becomes

$$R(i) = \sum_{j \neq \tau(i)} f(i, j) w(j).$$

The risk for the whole tree  $T$  remains  $R(T) = \sum_{i \in \text{Leaves}} R(i) s(i) / n$ . This risk can be used to calculate complexity and thus control pruning, as before.\*

### Loss matrix

Define  $L(j, c)$  to be the cost of misclassifying a class  $j$  observation as class  $c$ .

#### *Effect on splitting.*

Put  $L(j) = \sum_{c=1}^m L(j, c)$  then  $L(j)$  can be used as a class weight to modify  $p(i)$ ,  $p(i_0 | i)$  and  $p(i_1 | i)$  as above, with the same consequent effect on splitting.

#### *Effect on interpretation and pruning.*

We classify node  $i$  as before, that is, we put  $\tau(i) = \text{mode}(i)$ . However the risk for node  $i$  becomes

---

\* Class weights may be included in rpart as a cost vector, but it is not clear from the documentation whether they effect prediction and pruning or just the splitting.

$$R(i) = \sum_{j \neq \tau(i)} f(i, j) L(j, \tau(i)).$$

The risk for the whole tree  $T$  remains  $R(T) = \sum_{i \in \text{Leaves}} R(i) s(i) / n$ . This risk is then used to calculate complexity and thus control pruning, as before.

### Observation weights

Let  $(\mathbf{x}(t), y(t))$  be the  $t$ -th observation. We can associate with observation  $t$  a weight/importance  $b(t)$ . Let  $p(i)$  be the probability that an observation chosen at random is in node  $i$ , where the probability of choosing an observation is proportional to its importance. Then

$$p(i) \propto \sum_{t \in i} b(t).$$

Let  $i_0$  and  $i_1$  be subnodes of  $i$  created by splitting it, and let  $p(i_0 | i)$  be the probability that a randomly chosen observation is in node  $i_0$  given that it is in node  $i$ , then

$$p(i_0 | i) = p(i_0) / p(i) = \frac{\sum_{t \in i_0} b(t)}{\sum_{t \in i} b(t)}$$

*Effect on splitting.*

We still split to maximize  $\text{Gain}(i; i_0, i_1) = I(i) - (I(i_0) * p(i_0 | i) + I(i_1) * p(i_1 | i))$ . The impurity is unchanged but the conditional probabilities have changed, which will effect the splits.

*Effect on interpretation and pruning.*

We classify node  $i$  as class  $j$  where  $j$  maximises  $\sum_{t \in i, y(t)=j} b(t)$ . Letting the class of node  $i$  be  $\tau(i)$ , the risk for node  $i$  becomes

$$R(i) = \frac{\sum_{t \in i, y(t) \neq \tau(i)} b(t)}{s(i)},$$

and the risk for the whole tree  $T$  remains  $R(T) = \sum_{i \in \text{Leaves}} R(i) s(i) / n$ . This risk can be used to calculate complexity and thus control pruning, as before.

If the observation weights are the same for all observations of the same class, that is  $y(t) = j$  implies  $b(t) = w(j)$ , then observation weights are equivalent to class weights.<sup>†</sup>

### Computational issues

We consider the number of operations required to grow a tree, as a function of the number of observations  $n$  and their dimension  $k$ .

Given a node  $i$ , to choose its optimal split we need to consider all possible splits determined by

- Each input variable  $x_m$ ,  $m = 1, \dots, k$ ;
- Number of ways  $x_m$  can be split, call this  $d(x_m)$ ;

---

<sup>†</sup> Observation weights may be passed to `rpart` using the `weights` argument, but it is not clear if they are actually used or not.

For each possible split into  $i_0$  and  $i_1$  we need to calculate the frequencies  $f(i_0, j)$  and  $f(i_1, j)$  for each class  $j$ .

If  $x_m$  is binary then  $d(x_m) = 1$  and we can calculate the required frequencies in  $O(s(i))$  operations.

If  $x_m$  is ordinal then taking splits  $x_m < a$  and  $x_m \geq a$  we get at most  $s(i)$  splits (from the granularity of the data). By sorting the observations on  $x_m$  (which takes on average  $s(i) \log(s(i))$  operations using quicksort), we can calculate the required frequencies incrementally as we consider each possible split. Thus to consider the possible splits of an ordinal variable requires  $O(s(i) \log(s(i)))$  operations.

Thus if all variables are binary or ordinal then each split requires at most  $O(k s(i) \log(s(i)))$  operations, which scales nicely in  $k$  and  $s(i)$ . As the number of nodes grows their sizes  $s(i)$  decrease quickly, so the total number of operations still scales. The total number of leaves is  $O(\log(n))$ .

If  $x_m$  is categorical with  $p$  possible values then there are  $2^{p-1} - 1$  possible splits, which will be a problem if  $p$  is large.

## The Tree as a Piecewise Constant Approximation

### Boosting, Bagging and Forests

Decision trees are very easy to interpret, but are not particularly robust. That is, small changes in the training data can alter early splits, giving very different trees. They also tend to have a relatively high misclassification rate compared to other classification methods.

We briefly mention some techniques that can be used to improve the performance of decision trees, at the expense of taking longer.

Bagging was introduced by Breiman [2]. It uses an “average” tree based on a number of trees grown from bootstrapped resamples of the original training data. Bagging is a variance reduction technique. Random forests are a development of bagged trees which go somewhat further. See Breiman 2001.

Boosting was introduced by Schapire [4] and Freund & Schapire [3] and analysed by Friedman, Hastie & Tibshirani 1999. Boosting works by repeatedly regrowing the tree, but each time those records that were previously misclassified are given a greater weight than before.

### References

- [1] Giudici, P., 2003. *Applied Data Mining*, Wiley.
- [2] Breiman, L., 1996. Bagging predictors, *Machine Learning*, 26.
- [3] Freund, Y. & Schapire, R., 1996. Experiments with a new boosting algorithm, In *Machine Learning: Proceedings of the Thirteenth International Conference*.
- [4] Schapire, R., 1990. The strength of weak learnability, *Machine Learning* 5, 197-227.

Friedman, J. Hastie, T. & Tibshirani, R., 1999. Additive Logistic Regression: a Statistical View of Boosting, 2<sup>nd</sup> Revision.

Breiman, L., 2001. Random Forests, *Machine Learning* 45, 5-32.

Therneau, T.M. & Atkinson, E.J., 1997. An Introduction to Recursive Partitioning Using the RPART Routines, Technical Report, Mayo Foundation.

Breiman, L., Friedman, J.H., Olshen, R.A. & Stone, C.J., 1984. *Classification and Regression Trees*. Wadsworth.