Updated October 10, 1999

C5.1.3 Decision Tree Discovery

Ron Kohavi
ronnyk@CS.Stanford.EDU
Data Mining
Blue Martini Software
2600 Campus Dr. Suite 175
San Mateo, CA 94403

Ross Quinlan
quinlan@cse.unsw.edu.au
School of Computer Science and Engineering
Samuels Building, G08
University of New South Wales
Sydney 2052 Australia

Keywords: classification, induction, recursive partitioning, overfitting avoidance, cost-complexity pruning, pessimistic pruning, splitting criteria, decision rules

Abstract

We describe the two most commonly used systems for induction of decision trees for classification: C4.5 and CART. We highlight the methods and different decisions made in each system with respect to splitting criteria, pruning, noise handling, and other differentiating features. We describe how rules can be derived from decision trees and point to some difference in the induction of regression trees. We conclude with some pointers to advanced techniques, including ensemble methods, oblique splits, grafting, and coping with large data.

C5.1.3.1 C4.5

C4.5 belongs to a succession of decision tree learners that trace their origins back to the work of Hunt and others in the late 1950s and early 1960s (Hunt 1962). Its immediate predecessors were ID3 (Quinlan 1979), a simple system consisting initially of about 600 lines of Pascal, and C4 (Quinlan 1987). C4.5 has grown to about 9,000 lines of C that is available on diskette with Quinlan (1993). Although C4.5 has been superseded by C5.0, a commercial system from RuleQuest Research, this discussion will focus on C4.5 since its source code is readily available.

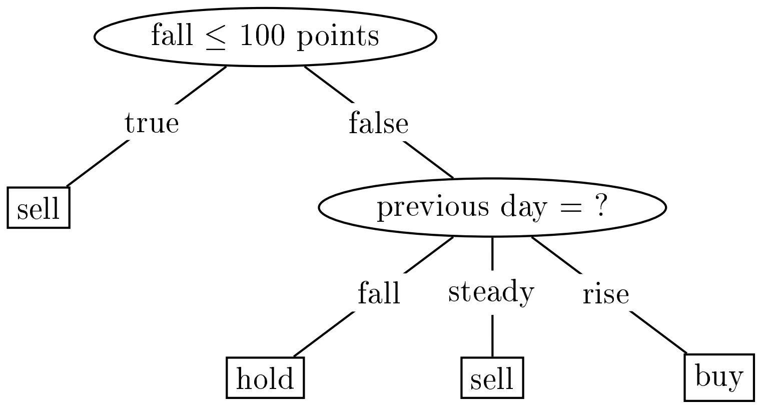


Figure C5.1.3.1: A simple decision tree

Input and Output

Input to C4.5 consists of a collection of *training cases*, each having a tuple of values for a fixed set of attributes (or independent variables) $A = \{A_1, A_2, ..., A_k\}$ and a class attribute (or dependent variable). An attribute A_a is described as *continuous* or *discrete* according to whether its values are numeric or nominal. The class attribute C is discrete and has values $C_1, C_2, ..., C_x$.

The goal is to learn from the training cases a function

$$DOM(A_1) \times DOM(A_2) \times ... \times DOM(A_k) \rightarrow DOM(C)$$

that maps from the attribute values to a predicted class.

The distinguishing characteristic of learning systems is the form in which this function is expressed. We focus here on *decision trees*, [link to section B2.4], a recursive structure that is

- a *leaf* node labelled with a class value, or
- a *test* node that has two or more outcomes, each linked to a subtree.

Figure C5.1.3.1 shows a simple example in which the tests appear in ovals, the leaves in boxes, and the test outcomes are labels on the links.

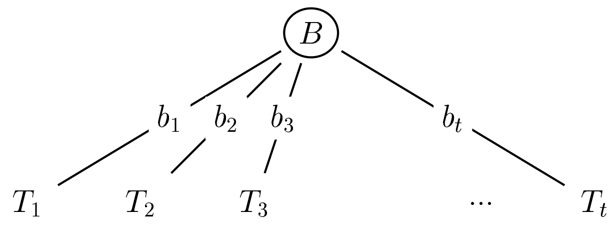
To classify a case using a decision tree, imagine a marker that is initially at the top (root) of the tree.

- If the marker is at a leaf, the label associated with that leaf becomes the predicted class.
- If the marker is at a test node, the outcome of that test is determined and the marker moved to the top of the subtree for that outcome.

Divide and Conquer

Decision tree learners use a method known as *divide and conquer* to construct a suitable tree from a training set S of cases:

- If all the cases in S belong to the same class (C_j, say), the decision tree is a leaf labelled with C_j.
- Otherwise, let B be some test with outcomes b_1, b_2, ..., b_t that produces a non-trivial partition of S, and denote by S_i the set of cases in S that has outcome b_i of B. The decision tree is



  where T_i is the result of growing a decision tree for the cases in S_i.

Candidate Tests

C4.5 uses tests of three types, each involving only a single attribute A_a. Decision regions in the instance space are thus bounded by hyperplanes, each orthogonal to one of the attribute axes.

- If A_a is a discrete attribute with z values, possible tests are:
  - "$A_a = ?$" with z outcomes, one for each value of A_a. (This is the default.)

  - "$A_a \in G$?" with $2 \leq g \leq z$ outcomes, where $G = \{G_1, G_2, ..., G_g\}$ is a partition of the values of attribute A_a. Tests of this kind are found by a greedy search for a partition G that maximizes the value of the splitting criterion (discussed below).

- If A_a has numeric values, the form of the test is "$A_a \leq \theta$" with outcomes *true* and *false*, where θ is a constant threshold. Possible values of θ are found by sorting the distinct values of A_a that appear in S, then identifying one threshold between each pair of adjacent values. (So, if the cases in S have d distinct values for A_a, d-1 thresholds are considered.)

Selecting Tests

In the divide and conquer algorithm, any test B that partitions S non-trivially will lead to a decision tree, but different Bs give different trees. Most learning systems attempt to keep the tree as small as possible because smaller trees are more easily understood and, by Occam's Razor arguments, are likely to have higher predictive accuracy (see, for instance, Quinlan & Rivest (1989)). Since it is infeasible to guarantee the minimality of the tree (Hyafil & Rivest 1976), C4.5 relies on greedy search, selecting the candidate test that maximizes a heuristic *splitting criterion.*

Two such criteria are used in C4.5, *information gain* and *gain ratio.* Let $RF(C_j, S)$ denote the relative frequency of cases in S that belong to class C_j. The information content of a message that identifies the class of a case in S is then

$$I(S) = -\sum_{j=1}^{x} RF(C_j, S) \log(RF(C_j, S)).$$

After S is partitioned into subsets S_1, S_2, ..., S_t by a test B, the information gained is then

$$G(S, B) = I(S) - \sum_{i=1}^{t} \frac{|S_i|}{|S|} I(S_i). \qquad \text{(C5.1.3.1)}$$

The gain criterion chooses the test B that maximizes $G(S, B)$.

A problem with this criterion is that it favors tests with numerous outcomes – for example, $G(S, B)$ is maximized by a test in which each S_i contains a single case. The gain ratio criterion sidesteps this problem by also taking into account the potential information from the partition itself:

$$P(S, B) = -\sum_{i=1}^{t} \frac{|S_i|}{|S|} \log\left(\frac{|S_i|}{|S|}\right). \qquad \text{(C5.1.3.2)}$$

Gain ratio then chooses, from among the tests with at least average gain, the test B that maximizes $G(S,B)/P(S,B)$.

Missing Values

Missing attribute values are a common occurrence in data, either through errors made when the values were recorded or because they were judged irrelevant to the particular case. Such lacunae affect both the way that a decision tree is constructed and its use to classify a new case.

When a decision tree is constructed from a set S of training cases, the divide and conquer algorithm selects a test on which to partition S. Let B be a potential test based on attribute A_a with outcomes b_1, b_2, ...,b_t. Denote by S_0 the subset of cases in S whose values of A_a are unknown, and hence whose outcome of B cannot be determined. As before, let S_i denote those cases with (known) outcome b_i of B. The information gained by B is reduced because we learn nothing about the cases in S_0; Equation C5.1.3.1 now becomes

$$G(S,B) = \frac{|S - S_0|}{|S|} G(S - S_0, B).$$

The split information is increased to reflect the additional "outcome" of the test (namely, the fact that it cannot be determined for the cases in S_0). Equation C5.1.3.2 is modified to

$$P(S,B) = -\frac{|S_0|}{|S|} \log\left(\frac{|S_0|}{|S|}\right) - \sum_{i=1}^{t} \frac{|S_i|}{|S|} \log\left(\frac{|S_i|}{|S|}\right).$$

Both changes have the effect of reducing the desirability of tests involving attributes with a high proportion of missing values.

When a test B has been chosen, C4.5 does not build a separate decision tree for the cases in S_0. Instead, they are notionally fragmented into fractional cases and added to the subsets corresponding to known outcomes. The cases in S_0 are added to each S_i with weight $|S_i|/|S - S_0|$.

Missing attribute values also complicate the use of the decision tree to classify a case. Instead of a single class, the initial result of the classification is a class probability distribution determined as follows: Let $CP(T,Y)$ denote the result of classifying case Y with decision tree T.

- If T is a leaf, $CP(T,Y)$ is the relative frequency of training cases that reach T.

- If T is a tree whose root is test B, and the outcome of B on case Y is known (b_i, say), then
$$CP(T, Y) = CP(T_i, Y)$$
where, as before, T_i is the decision tree for outcome b_i.

- Otherwise, all outcomes of B are explored and combined probabilistically, giving
$$CP(T, Y) = \sum_{i=1}^{t} \frac{|S_i|}{|S - S_0|} CP(T_i, Y).$$
Note that the weight of the subtree T_i depends on the proportion of training cases that have outcome b_i, interpreted as the prior probability of that outcome.

When the class probability distribution resulting from classifying case Y with the decision tree has been determined, the class with the highest probability is chosen as the predicted class.

Avoiding Overfitting

The divide and conquer algorithm partitions the data until every leaf contains cases of a single class, or until further partitioning is impossible because two cases have the same values for each attribute but belong to different classes. Consequently, if there are no conflicting cases, the decision tree will correctly classify all training cases. This so-called *overfitting* is generally thought to lead to a loss of predictive accuracy in most applications (Quinlan 1986).

Overfitting can be avoided by a *stopping criterion* that prevents some sets of training cases from being subdivided (usually on the basis of a statistical test of the significance of the best test), or by removing some of the structure of the decision tree after it has been produced. Most authors agree that the latter is preferable since it allows potential interactions among attributes to be explored before deciding whether the result is worth keeping.

C4.5 employs a mechanism of the latter kind. Before discussing it, we introduce the heuristic on which it is based.

Estimating True Error Rates

Consider some classifier Z formed from a subset S of training cases, and suppose that Z misclassifies M of the cases in S. The *true* error rate of Z is its accuracy over the entire universe from which the training set was sampled. The true error rate is usually markedly higher than the classifier's *resubstitution* error rate on the training cases (here $M/|S|$), which might be near zero for an unpruned decision tree.

The true error rate is often estimated by measuring Z's error rate on a collection of *unseen* cases that were not used in its construction; this is the best strategy when a substantial set of unseen cases is available. In many applications, though, data is scarce and all of it is needed to construct the classifier. C4.5 estimates the true error rate of Z using only the values M and $|S|$ from the training set as follows.

If an event occurs M times in N trials, the ratio M/N is an estimate of the probability p of the event. We can go further and derive confidence limits for p; for a given confidence CF, an upper limit p_r can be found such that $p \leq p_r$ with probability 1-CF. Following (Diem 1962, page 185), p_r satisfies the following equations:

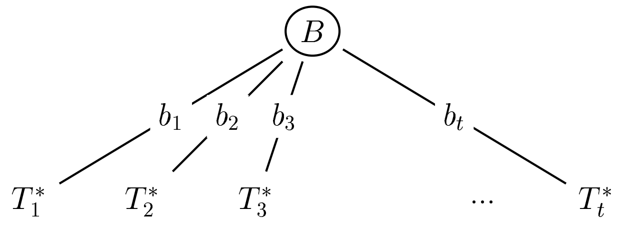
$$CF = \begin{cases} (1-p_r)^N & \text{for } M = 0 \\ \sum_{i=0}^{M} \binom{N}{i} p_r^i (1-p_r)^{N-i} & \text{for } M > 0 \end{cases}$$

(The same source gives a quickly-computable approximation for p_r in the latter case.)

Now, the classifier Z can be viewed as causing M error events in $|S|$ "trials". Since Z was constructed to fit the cases in S, and so tends to minimize the apparent error rate, the upper bound p_r is used as a more conservative estimate of the error rate of Z on unseen cases. In the following, we will use $U_{CF}(M, N)$ to denote the error bound p_r above. C4.5 uses a default CF value of 0.25, but this can be altered to cause higher or lower levels of pruning.

Pruning Decision Trees

After a decision tree is produced by the divide and conquer algorithm, C4.5 prunes it in a single bottom-up pass. Let T be a non-leaf decision tree, produced from a training set S, of the form



where each T_i^* has already been pruned. Further, let T_f^* be the subtree corresponding to the most frequent outcome of B, and let L be a leaf labelled with the most frequent class in S. Let the number of cases in S misclassified by T, T_f^*, and L be E_T, $E_{T_f^*}$, and E_L respectively. C4.5's tree pruning algorithm considers the three corresponding estimated error rates

- $U_{CF}(E_T, |S|)$,
- $U_{CF}(E_L, |S|)$, and
- $U_{CF}(E_{T_f^*}, |S|)$.

Depending on whichever is lower, C4.5

- leaves T unchanged;
- replaces T by the leaf L; or
- replaces T by its subtree T_f^*.

This form of pruning is computationally efficient and gives quite reasonable results in most applications.

C5.1.3.2 CART

CART, an acronym for Classification And Regression Trees, is described in the book by Breiman, Friedman, Olshen & Stone (1984). The use of trees in the statistical community dates back to AID (Automatic Interaction Detection) by Morgan & Sonquist (1963), and to later work on THAID by Morgan and Messenger in the early 1970s.

CART® is also the name of the system currently implementing the methodology described in the above book. It is sold by Salford Systems.

The basic methodology of divide and conquer described in C4.5 is also used in CART. The differences are in the tree structure, the splitting criteria, the pruning method, and the way missing values are handled.

Tree Structure

CART constructs trees that have only binary splits. This restriction simplifies the splitting criterion because there need not be a penalty for multi-way splits (Kononenko 1995*b*). Futhermore, if the label is binary, the binary split restriction allows CART to optimally partition categorical attributes (minimizing any concave splitting criteria) to two subsets of values in time that is linear time in the number of attribute values (Breiman et al. 1984, Theorem 4.5). The restriction has its disadvantages, however. The tree may be less interpretable with multiple splits occurring on the same attribute at adjacent levels. There may be no good binary split on an attribute that has a good multi-way split (Kononenko 1995*a*), which may lead to inferior trees.

Splitting Criteria

CART uses the Gini diversity index as a splitting criterion. Let $RF(C_j, S)$ denote the relative frequency of cases in S that belong to class C_j. The Gini index is defined as:

$$I_{\text{gini}}(S) = 1 - \sum_{j=1}^{x} RF(C_j, S)^2 \quad ,$$

and the information gain due to a split is computed as in Equation C5.1.3.1.

A *class probability tree* predicts a class distribution for an example instead of a single class. The common measure of performance for a class probability tree is the mean squared error. For each class j, let $C_j(e)$ be the indicator variable that is one if the class for the example e is j and zero otherwise. The *mean squared error*, or MSE, is defined as:

$$\text{MSE} = E_e \left[\sum_{j=1}^{x} (C_j(e) - P_j(\text{e}))^2 \right]$$

where the expectation is over all examples, and $P_j(e)$ represents the probability assigned to class j for example e by the probabilistic classifier.

The interesting observation about the Gini diversity index is that it minimizes the resubstitution estimate for the mean squared error.

CART also supports the *twoing* splitting criterion, which can be used for multi-class problems. At each node, the classes are separated into two superclasses containing disjoint and mutually exhaustive classes. A splitting criterion for a two-class problem is used to find the attribute and the two superclasses that optimize the two-class criterion. The approach gives “strategic” splits in the sense that several classes that are similar are grouped together.

Pruning

CART uses a pruning technique called *minimal cost complexity pruning*, which assumes that the bias in the resubstitution error of a tree increases linearly with the number of leaf nodes. The cost assigned to a subtree is the sum of two terms: the resubstitution error and the number of leaves times a complexity parameter α. Formally,

$$R_\alpha = R(T) + \alpha \cdot \text{number-of-leaves} \quad .$$

It can be shown that, for every value of α, there exists a unique smallest tree minimizing R_α (Breiman et al. 1984, Proposition 3.7). Note that, although α runs through a continuum of values, there are at most a finite number of possible subtrees. There is thus a sequence of trees minimizing R_α, $T_1 \succ T_2 \succ \cdots \succ \{T_t\}$, created by varying α from zero to infinity. The trees are nested: each tree is contained in the previous one. An efficient "weakest-link" pruning algorithm can be constructed to compute T_{k+1} from T_k.

When a large dataset is available, selecting the best α to minimize the true error can be done by setting aside a holdout set (e.g., a third of the data) and constructing $T_1 \succ T_2 \succ \cdots \succ \{T_t\}$ from the data, excluding the holdout set. The examples in the holdout set can then be classified using each tree, giving an estimate of the true error of each tree. The α matching the tree that minimizes the error can then be used as the pruning parameter to prune the tree built from the whole dataset.

The pruning step described above is relatively fast, but because a holdout set was used, a second tree is usually built using the whole dataset in order to make efficient use of all the data. Building a second tree effectively doubles the induction time.

Moreover, if the dataset is small, the error estimates have high variance and precious data (held out) are not used in building the initial tree. CART therefore uses 10-fold cross validation (Stone 1974, Kohavi 1995) to improve the error estimates and utilize more data. The procedure for pruning using 10-fold cross validation is more complex since multiple trees must be built and pruned. We refer the reader to Breiman et al. (1984) for details. The time complexity of the pruning step when 10-fold cross-validation is used is a factor of 10 more expensive than C4.5's pruning, but it does tend to produce smaller trees (Oates & Jensen 1997).

Even when cross-validation is used, the pruning parameter is sometimes unstable. Also, to improve comprehensibility, it is sometimes preferable to choose a smaller tree that has comparable accuracy to the best tree. CART employs the "1 SE rule," which chooses the smallest tree whose estimated mean error rate is within one standard error (estimated standard deviation) of the estimated mean error rate of the best tree.

Missing Values

Unlike C4.5, CART does not penalize the splitting criterion during the tree construction if examples have unknown values for the attribute used in the split. The criterion uses only those instances for which the value is known. Unlike C4.5, CART finds several *surrogate* splits that can be used instead of the original split. During classification, the first surrogate split based on a known attribute value is used.

The surrogates cannot be chosen based on the original splitting criterion because the subtree at each node is constructed based on the original split selected. The surrogate splits are therefore chosen to maximize a measure of predictive association with the original split. This procedure works well if there are attributes that are highly correlated with the chosen attribute.

Regression Trees

As its name implies, CART also supports building regression trees. Regression trees are somewhat simpler than classification trees because the growing and pruning criteria used in CART are the same. The regression tree structure is similar to a classification tree, except that each leaf predicts a real number.

The resubstitution estimate is the mean squared error:

$$R(S) = \frac{1}{n} \sum_i (y_i - h(t_i))^2 \quad ,$$

where y_i is the real-valued label for example t_i and $h(t_i)$ is the (real-valued) prediction. The splitting criteria is chosen to minimize the resubstitution estimate. Pruning is done in a manner similar to the cost complexity pruning described above.

In CART, each leaf predicts a constant value; model trees generalize to building a model at each leaf (Frank, Wang, Inglis, Holmes & Witten 1998).

C5.1.3.3 Advanced Methods

Decision trees have been extended in many ways. We provide a short list of pointers to important topics.

1. The induction of *Oblique Decision Trees* allow tests at the nodes to be linear combinations of attributes. Murthy, Kasif & Salzberg (1994) describe an induction system that allows such tests. Breiman et al. (1984) also describes some mechanisms for supporting such splits in CART. The main advantage of oblique splits is their ability to create splits that are not axis-orthogonal. The disadvantage is loss of comprehension.

2. Many different pruning methods have been proposed in the literature. Esposito, Malerba & Semeraro (1995) provide a comparison of pruning and grafting methods. Kearns & Mansour (1998) provide a theoretically-justified pruning algorithm. Quinlan & Rivest (1989), Mehta, Rissanen & Agrawal (1995), and Wallace & Patrick (1993) describe MDL- (minimum description length) and MML- (minimum message length) based pruning methods.

3. C4.5 also contains a mechanism to re-express decision trees as ordered lists of if-then rules. Each path from the root of the tree to a leaf gives the conditions that must be satisfied if a case is to be classified by that leaf. C4.5 generalizes this prototype rule by dropping any conditions that are irrelevant to the class, guided again by the heuristic for estimating true error rates. The set of rules is reduced further based on the MDL principle described above (see C8.2.1). There are usually substantially fewer final rules than there are leaves on the tree, and yet the accuracy of the tree and the derived rules is similar. Rules have the added advantage of being more easily understood by people.

4. Ensemble methods that build multiple trees can dramatically reduce the error, but usually result in huge structures that are incomprehensible. Breiman (1996) describes a Bagging (Bootstrap Aggregating) procedure. Schapire (1990) introduced boosting, which was later enhanced in Freund & Schapire (1995). Kohavi & Kunz (1997) describe option trees, which provide the advantages of voting methods, yet keep a tree-like structure that can be shown to users. Empirical comparisons were done in Quinlan (1996) and Bauer & Kohavi (1999).

5. Most implementation of decision trees require loading the data into memory. Shafer, Agrawal & Mehta (1996) describe the SPRINT algorithm, which can

scale out of core. Variants and other methods for scaling to larger datasets are described in Freitas & Lavington (1998). Several commercial systems implementing decision trees also provide parallel implementations.

6. Decision trees choose a split and do not revisit choices. Lookahead methods are described in Murthy & Salzberg (1995).

7. Utgoff (1997) describes how to update decision trees incrementally as more data is made available.

8. Most algorithms assume 0-1 costs for mistakes. Descriptions on how to generalize to loss matrices are given in Breiman et al. (1984) and a comparison is given in Pazzani, Merz, Murphy, Ali, Hume & Brunk (1994). More information can be found in Turney (1997).

9. Lazy decision trees (Friedman, Kohavi & Yun 1996) conceptually choose the best tree for a given test instance. In practice, only a path needs to be constructed.

10. Oblivious decision trees conduct the same split across a whole level (Kohavi & Li 1995) and can be converted into a graph or a decision table (Kohavi & Sommerfield 1998).

References

Bauer, E. & Kohavi, R. (1999), ‘An empirical comparison of voting classification algorithms: Bagging, boosting, and variants’, *Machine Learning* **36**, 105–139.

Breiman, L. (1996), ‘Bagging predictors’, *Machine Learning* **24**, 123–140.

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

Diem, K., ed. (1962), *Documenta Geigy Scientific Tables*, Geigy Pharmaceuticals.

Esposito, F., Malerba, D. & Semeraro, G. (1995), Simplifying decision trees by pruning and grafting: New results, *in* N. Lavrac & S. Wrobel, eds, ‘Machine Learning: ECML-95 (Proc. European Conf. on Machine Learning, 1995)’, Lecture Notes in Artificial Intelligence 914, Springer Verlag, Berlin, Heidelberg, New York, pp. 287–290.

Frank, E., Wang, Y., Inglis, S., Holmes, G. & Witten, I. H. (1998), 'Using model trees for classification', *Machine Learning* **32**(1), 63–76.

Freitas, A. A. & Lavington, S. H. (1998), *Mining Very Large Databases With Parallel Processing*, Kluwer Academic Publishers.

Freund, Y. & Schapire, R. E. (1995), A decision-theoretic generalization of on-line learning and an application to boosting, *in* 'Proceedings of the Second European Conference on Computational Learning Theory', Springer-Verlag, pp. 23–37. To appear in Journal of Computer and System Sciences.

Friedman, J., Kohavi, R. & Yun, Y. (1996), Lazy decision trees, *in* 'Proceedings of the Thirteenth National Conference on Artificial Intelligence', AAAI Press and the MIT Press, pp. 717–724.

Hunt, E. B. (1962), *Concept Learning: An Information Processing Problem*, Wiley.

Hyafil, L. & Rivest, R. L. (1976), 'Constructing optimal binary decision trees is NP-complete', *Information Processing Letters* **5**(1), 15–17.

Kearns, M. & Mansour, Y. (1998), A fast, bottom-up decision tree pruning algorithm with near-optimal generalization, *in* J. Shavlik, ed., 'Machine Learning: Proceedings of the Fifteenth International Conference', Morgan Kaufmann Publishers, Inc., pp. 269–277.

Kohavi, R. (1995), A study of cross-validation and bootstrap for accuracy estimation and model selection, *in* C. S. Mellish, ed., 'Proceedings of the 14th International Joint Conference on Artificial Intelligence', Morgan Kaufmann, pp. 1137–1143. `http://robotics.stanford.edu/~ronnyk`.

Kohavi, R. & Kunz, C. (1997), Option decision trees with majority votes, *in* D. Fisher, ed., 'Machine Learning: Proceedings of the Fourteenth International Conference', Morgan Kaufmann Publishers, Inc., pp. 161–169. Available at `http://robotics.stanford.edu/users/ronnyk`.

Kohavi, R. & Li, C.-H. (1995), Oblivious decision trees, graphs, and top-down pruning, *in* C. S. Mellish, ed., 'Proceedings of the 14th International Joint Conference on Artificial Intelligence', Morgan Kaufmann, pp. 1071–1077.

Kohavi, R. & Sommerfield, D. (1998), Targeting business users with decision table classifiers, *in* R. Agrawal, P. Stolorz & G. Piatetsky-Shapiro, eds, 'Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining', AAAI Press, pp. 249–253.

Kononenko, I. (1995*a*), A counter example to the stronger version of the binary tree hypothesis, *in* 'ECML-95 workshop on Statistics, machine learning, and knowledge discovery in databases', pp. 31–36.

Kononenko, I. (1995*b*), On biases in estimating the multivalued attributes, *in* C. S. Mellish, ed., 'Proceedings of the 14th International Joint Conference on Artificial Intelligence', Morgan Kaufmann. `http://ai.fri.uni-lj.si/papers/index.html`.

Mehta, M., Rissanen, J. & Agrawal, R. (1995), MDL-based decision tree pruning, *in* U. M. Fayyad & R. Uthurusamy, eds, 'Proceedings of the first international conference on knowledge discovery and data mining', AAAI Press, pp. 216–221.

Morgan, J. N. & Sonquist, J. A. (1963), 'Problems in the analysis of survey data, and a proposal', *Journal of the American Statistical Association* **58**, 415–434.

Murthy, S. K., Kasif, S. & Salzberg, S. (1994), 'A system for the induction of oblique decision trees', *Journal of Artificial Intelligence Research* **2**, 1–33.

Murthy, S. & Salzberg, S. (1995), Lookahead and pathology in decision tree induction, *in* C. S. Mellish, ed., 'Proceedings of the 14th International Joint Conference on Artificial Intelligence', Morgan Kaufmann, pp. 1025–1031.

Oates, T. & Jensen, D. (1997), The effects of training set size on decision tree complexity, *in* D. Fisher, ed., 'Machine Learning: Proceedings of the Fourteenth International Conference', Morgan Kaufmann, pp. 254–262.

Pazzani, M., Merz, C., Murphy, P., Ali, K., Hume, T. & Brunk, C. (1994), Reducing misclassification costs, *in* 'Machine Learning: Proceedings of the Eleventh International Conference', Morgan Kaufmann.

Quinlan, J. R. (1979), Discovering rules from large collections of examples: A case study, *in* D. Michie, ed., 'Expert Systems in the Micro Electronic Age', Edinburgh University Press.

Quinlan, J. R. (1986), 'Induction of decision trees', *Machine Learning* **1**, 81–106. Reprinted in Shavlik and Dietterich (eds.) *Readings in Machine Learning.*

Quinlan, J. R. (1987), Inductive knowledge acquisition: A case study, *in* J. R. Quinlan, ed., 'Applications of Expert Systems', Addison-Wesley, chapter 9, pp. 157—173.

Quinlan, J. R. (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, California.

Quinlan, J. R. (1996), Bagging, boosting, and c4.5, *in* 'Proceedings of the Thirteenth National Conference on Artificial Intelligence', AAAI Press and the MIT Press, pp. 725–730.

Quinlan, J. R. & Rivest, R. L. (1989), 'Inferring decision trees using the minimum description length principle', *Information and Computation* **80**, 227–248.

Schapire, R. E. (1990), 'The strength of weak learnability', *Machine Learning* **5**(2), 197–227.

Shafer, J., Agrawal, R. & Mehta, M. (1996), Sprint: a scalable prallel classifier for data mining, *in* 'Proceedings of the 22nd International Conference on Very Large Databases (VLDB)'.

Stone, M. (1974), 'Cross-validatory choice and assessment of statistical predictions', *Journal of the Royal Statistical Society B* **36**, 111–147.

Turney, P. (1997), Cost-sensitive learning.
`http://ai.iit.nrc.ca/bibliographies/cost-sensitive.html`.

Utgoff, P. E. (1997), 'Decision tree induction based on efficient tree restructuring', *Machine Learning* **29**, 5.

Wallace, C. & Patrick, J. (1993), 'Coding decision trees', *Machine Learning* **11**, 7–22.