Bivariate Decision Trees

Jan C. Bioch, Onno van der Meer, Rob Potharst

Department of Computer Science
Erasmus University Rotterdam
P.O. Box 1738, 3000 DR Rotterdam
The Netherlands
{bioch,vdmeer,robp}@cs.few.eur.nl

Abstract

Decision trees with tests based on a single variable, as produced by methods such as ID3, C4.5 etc., often require a large number of tests to achieve an acceptable accuracy. This makes interpretation of these trees, which is an important reason for their use, disputable. Recently, a number of methods for constructing decision trees with multivariate tests have been presented. Multivariate decision trees are often smaller and more accurate than univariate trees; however, the use of linear combinations of the variables may result in trees that are hard to interpret. In this paper we consider trees with test based on combinations of at most two variables. We show that bivariate decision trees are an interesting alternative to both uni– and multivariate trees.

1 Introduction

For classification problems decision tree methods such as ID3 and C4.5 [Qui96] continue to be very popular, notwithstanding a number of obvious drawbacks. One drawback is the formidable size of many of these trees, even after pruning, in all but the most toy-like practical situations. To attain the desired accuracy, often dozens, sometimes hundreds of tests are needed, resulting in a decision tree, which is besides tedious to handle far from easily interpreted. To be sure, one of the main reasons why decision tree methods are preferred to competitors, such as neural networks, is supposedly their ease of interpretation. One way to get smaller but equally accurate trees is to base the test in a node of the tree not on a single variable, but on a number of variables [BU95]. Up until now, most implementations of this idea, usually called multivariate decision trees, make use of tests, which gauge the value of a linear combination of many variables per node. Though the resulting trees become much smaller, they are equally hard to interpret, as anyone knows, who has ever faced the problem of interpreting the conjunction of a number of tests like

$$1.3450x_1 - 0.6333x_2 - 0.2214x_3 + 0.8779x_4 + 0.5667x_5 - 1.1432x_6 \leq 0.8883 \qquad (1)$$

or worse. In this paper we investigate the possibility of restricting the number of variables per test to two, thus preventing hardships like (1) while at the same time we hope to obtain sufficiently accurate and not too large trees. In section 2 we start with describing both univariate and multivariate decision tree methods. In section 3 two variants of our own proposed method are described. The next two sections apply our methods, first to a constructed dataset, and subsequently to five well known real life data sets. In the last section we conclude that our so-called bivariate decision trees can be competitive with multivariate trees in both size and accuracy, without sacrificing the interpretability of the resulting trees.

2 Inducing Binary Decision Trees

In this paper we will study tree classifiers (see e.g. [Rip96]) for multiclass classification problems. The trees we will consider consist of nodes, leaves and branches, e.g.

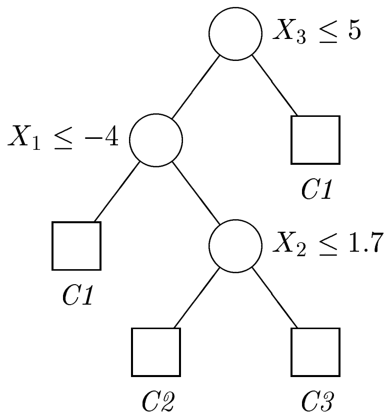


Figure 1: Binary Decision Tree: Example

With each node a test is associated, which in general will be of the form

$$\sum_{i=1}^{n} a_i X_i \le T. \tag{2}$$

Here $X_1, \ldots, X_n$ are features, the a_i are real coefficients and T is a real cut-off point. With each leaf one of the classes $C_1, \ldots, C_k$ is associated. Furthermore, all splits will be *binary*: go left if the test is satisfied, and right if not. Thus, the example tree of Figure 1 will classify an example as C_1, if $X_3 \le 5$ and $X_1 \le -4$ or if $X_3 > 5$. For the induction of such a decision tree, we will make use of a training set of examples or data points, for each of which both the values of all the features, and a class label are given.

2.1 Univariate Trees

Univariate decision trees recursively partition the training set by finding tests of the form $X_i \le T$ that minimize an impurity measure. Using this test the data is partitioned into two groups (left/right) according to whether the data points satisfy the inequality.

As an example consider the following (sorted) six values for X and corresponding class labels:

	1	2	3	4	5	6
X	-0.9	-0.5	-0.1	0.4	0.7	0.9
Class	1	1	2	1	2	2

The possible values for the threshold T are the midpoints between two adjacent values of X. If we use a convex splitting criterion, we know from the work of Fayyad and Irani [FI93] that we need only consider midpoints between values of X with different class labels. For instance, (0.4+0.7)/2 is a possible threshold and leads to the following partition:

	Class 1	Class 2	Total
Left	3	1	4
Right	0	2	2

There are many splitting criteria available, however, we limit our attention to the Gini Index as proposed by Breiman et al [BFOS84] in the form as given by Murthy [MKSB93] for C classes:

$$I = \left(N_l(1 - \sum_{i=1}^{k} (L_i/N_l)^2) + N_r(1 - \sum_{i=1}^{k} (R_i/N_r)^2) \right) /N,$$

where $L_i(R_i)$ is the number of examples of class i in the left (right) node, and $N_l(N_r)$ is the total number of examples in the left (right) node. Calculating the Gini Index for the example gives

$$\left(4(1.0 - (3/4)^2 - (1/4)^2) + 2(1.0 - (0/2)^2 - (2/2)^2)\right) /6.$$

2.2 Multivariate Trees

Several methods for constructing multivariate decision trees exist. These include CART [BFOS84], Recursive Least Squares [BU95], Linear Machine Decision Trees [UB91] and OC1 [MKSB93, Mur95]. Of these only the first and last produce binary trees for any number of classes, so we confine our discussion to CART and its descendant OC1.

2.2.1 CART

In their epoch making 1984 book *Classification And Regression Trees*, abbreviated CART, Breiman *et al.* [BFOS84] describe both univariate and multivariate classification trees. The algorithm they propose for the multivariate case searches in each node for the linear test, which splits the data for that node into two subsets showing the least impurity. For a given node, this is done as follows:

1. Normalize the data by centering all the data points for the node at their medians, and dividing by their interquartile ranges; we then get N data points $x^{(1)}, \dots, x^{(N)}$ with $x^{(j)} = (x_1^{(j)}, \dots, x_n^{(j)})$ a vector of values for each of the n features.

2. Find the best univariate split $X_i \leq T$, in the way as described above.

3. Next, in a number of cycles, we try to improve upon a given split $V = \sum_{i=1}^{n} a_i X_i \leq T$ by successively considering splits of the form:

$$V - \delta(X_i + \gamma) \leq T. \tag{3}$$

A complete cycle consists of all updates of form (3). For instance, for $i = 1$, we can find the best δ by calculating

$$u^{(j)} = \frac{v^{(j)} - T}{x_1^{(j)} + \gamma}$$

for $j = 1, \ldots, N$, taking γ as a fixed value. By considering the midpoints of the ordered $u^{(j)}$ as possible candidates for δ, we can calculate the best split 3. It is proposed, that this procedure is performed for three different values of γ, namely $-0.25, 0$ and 0.25. The best of these three is then taken as the δ and γ that are used to update V to V' as follows:

$$V' = \sum_{i=1}^{n} a_i' X_i$$

with

$$a_i' = \begin{cases} a_i & \text{for } i \neq 1 \\ a_1 - \delta & \text{otherwise} \end{cases}$$

and

$$T' = T + \delta\gamma.$$

This step is repeated subsequently for the other variables $X_2, \ldots, X_n$. If, at the end of such a cycle we have obtained test $V = \sum a_i X_i \leq T$, we next try to improve T by finding the best test $\sum a_i X_i \leq T$, fixing the a_i and changing only the threshold T.

4. We stop cycling when the decrease in impurity from one cycle to the next is below a predetermined small level ϵ.

This algorithm aims at finding a global minimum for the impurity of the considered linear splits. However, the algorithm can get stuck in a local minimum.

2.3 Example of CART parameter updating

We illustrate the process by which cart updates the parameters of a multivariate test with an example based on a linear separable training set with two inputs and two classes.

After normalizing the data, the best univariate split is found to be $X_i \leq 0.469418$, see figure 2. The initial linear combination that we aim to improve with the CART algorithm is therefore $V = 1X_1 + 0X_2 \leq 0.469418$, conform equation 3 above.

As a first step, we want to find a new value for a_i that improves the quality of the split. For the three value of γ we find the best δ :

γ	δ	Impurity
-0.25	1.294015	0.496528
0	-0.000459	0.215278
0.25	-0.000196	0.215279

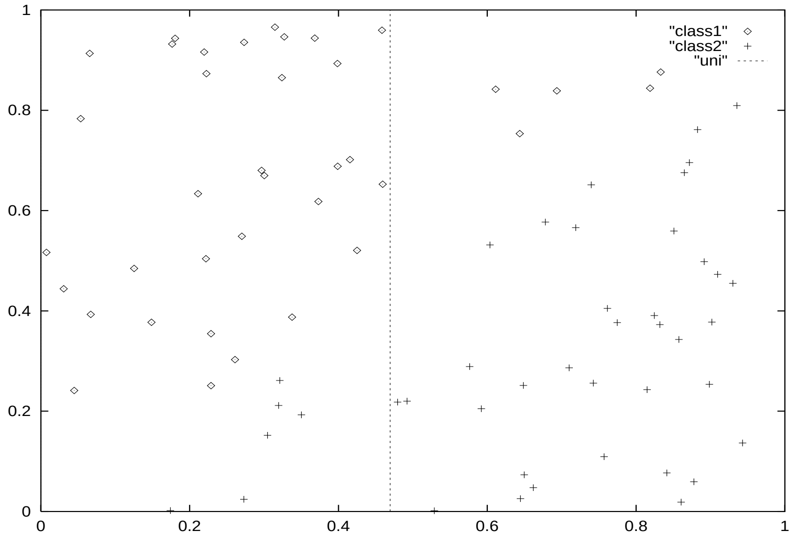


Figure 2: Linear separable data set with best univariate split

We set γ^* and δ^* to the best values in these three runs : $\delta^* = -0.000459$ and $\gamma^* = 0$. We update a_i and T as follows: $a_i = a_i - \delta^*$ and $T = T + \gamma^*\delta^*$. In this case we obtain $a_1 = 1 + 0.000459$ and $T = 0.469418 - 0 * 0.000459$.

We repeat this procedure for X_2 starting from the new values for a_1 and T. From the following table we obtain $a_2 = -0.610118$ and $T = 0.469418 - 0.25 * 0.610118 = 0.316889$.

γ	δ	Impurity
-0.25	0.610118	0.160417
0	0.017975	0.215278
0.25	0.006548	0.215278

Next, we keep the a_1 and a_2 constant and find a new value for T based on the $v^(j)$; that is for every item in the training set we calculate $\sum_i a_i x_i$ and find an optimal value T_0. In the example we find the value $T_0 = -0.182293$ and update $T = T - T_0 = 0.134596$.

We repeat this procedure of updating the a_i and T until the change in impurity is smaller than a threshold (e.g. 0.0001) or the impurity equals zero.

In this example, we continue by updating a_1 :

γ	δ	Impurity
-0.25	0.332095	0.000000
0	0.076605	0.027083
0.25	0.003661	0.052846

We see that using $\gamma = -0.25$ results in a perfect split , so the final split after updating a_1 and T is $0.668365x1 - 0.610118x2 \leq 0.051572$.

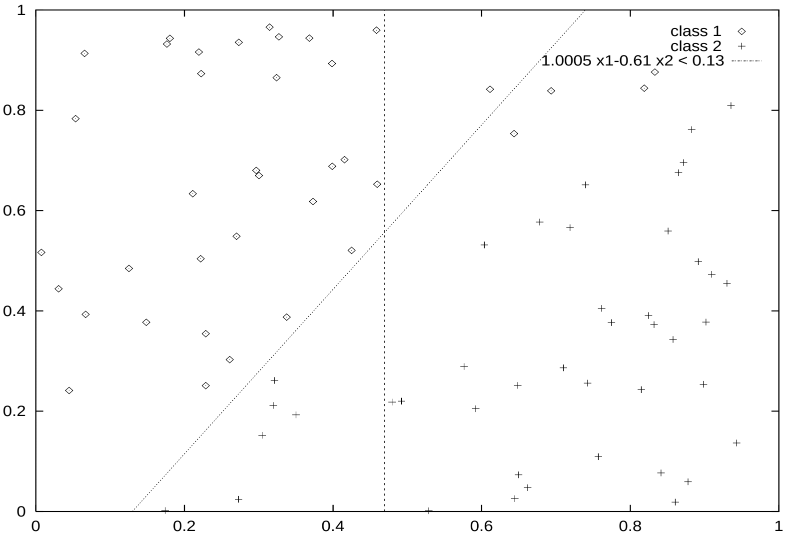


Figure 3: Split after one step of the CART parameter update

2.4 OC1

The linear combination search algorithm of CART, as described in the previous subsection, may not always get close to a globally optimal split. This problem has been addressed by Murthy [MKSB93] in the OC1 method. The parameter update method of OC1 follows the CART method, but includes random perturbations of the parameters when a local minimum is reached and restarts from random locations (the first starting location is the best univariate split).

3 Bivariate Decision Trees

With multivariate decision trees, each node contains a test of the form (2). In order to enhance interpretability, we want to reduce the number of variables used in (2) to at most two, making it a test of the form

$$a_i X_i + a_j X_j \leq T. \tag{4}$$

Of course, in different nodes, different choices for variables X_i and X_j may be used. Our proposed method has two variants, which we will call BIT1 and BIT2. In the first variant, in each node we look for the best test of type (4) allowing all combinations of X_i and X_j, and all possible coefficients a_i and a_j. The second variant only allows certain values of a_i and a_j: we look for the best test of the type

$$X_i + X_j \leq T \text{ or } X_i - X_j \leq T.$$

Although it may look like this restricts the set of admissible tests, the input space partition induced by a number of such tests is still much more flexible than in the univariate case, as can be seen from Figure 5(a). Both variants only use a bivariate test if it improves upon the best univariate test.

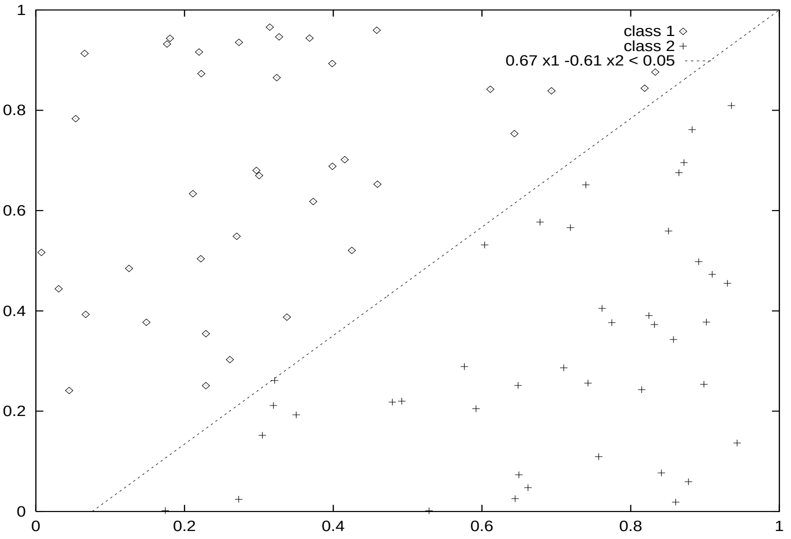


Figure 4: Final split obtained by cart

3.1 BIT1

For each combination of attributes X_i and X_j of the form $a_iX_i + a_jX_j \leq T$ we use the CART method of parameter updating to find values for a_i, a_j and T. This process is started by setting the parameters to values corresponding to the univariate test for attribute X_i. For instance, if the best univariate test for X_2 is $X_2 \leq 4$, and we combine X_2 and X_3, we set $a_2 = 1$, $a_3 = 0$ and $T = 4$.

If we have n variables we need to look at $n(n-1)$ combinations:

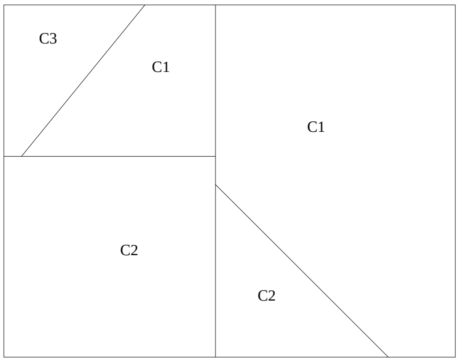
1 Find best univariate split
2 Find best bivariate split:
  for $i = 1$ to n
    for $j = 1$ to n and $j \neq i$
      find $a_iX_i + a_jX_j \leq T$

If the bivariate split has lower impurity than the univariate split, we use the bivariate split. Another possibility is to choose the bivariate split only if the improvement in impurity exceeds a threshold, say 10 percent; however, experiments suggest that this procedure often leads to increased tree size.

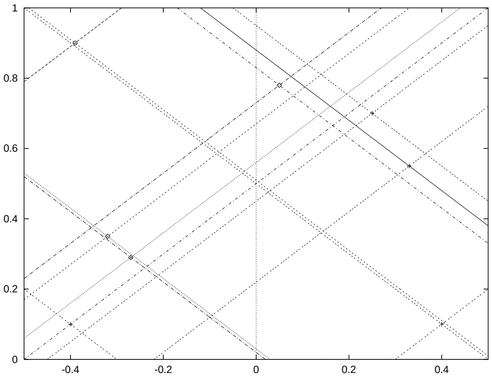
3.2 BIT2

A much faster method is to consider tests of the form $X_i + X_j \leq T$ or $X_i - X_j \leq T$, in addition to the best univariate test. Where univariate tests form lines parallel to the axis, these bivariate tests form 45 degree lines. Selecting the optimal line is analogous to finding the best univariate split. The combinations of X_i and X_j are easily calculated from the training data and give new univariate problems, so we can use the procedure based on the results of Fayyad and Irani described above. The procedure of finding the best test is similar to that of BIT1. Note that this method is not only faster than BIT1, it also leads to simpler tests. Furthermore, it is flexible in modeling class configurations in input space.

Figure 5b shows a data set with two attributes and two classes and the 45 degree lines proposed by this method.



(a) Input space partition



(b) Candidate lines BIT2

Figure 5: Examples of the effect of 45 degree lines

This method assumes that all attributes are on the same scale. If one attribute takes on values in $[1, \ldots, 2]$ and a second values from $[3, \ldots, 187]$, the combination of the attributes will have no effect. This can be solved by scaling all attributes to have the same range, e.g. $[0, 1]$ and adjusting the solution for the original values. For instance, if we find the test $X_1 + X_2 \leq 1.5$ for the scaled data, the test on the original data is

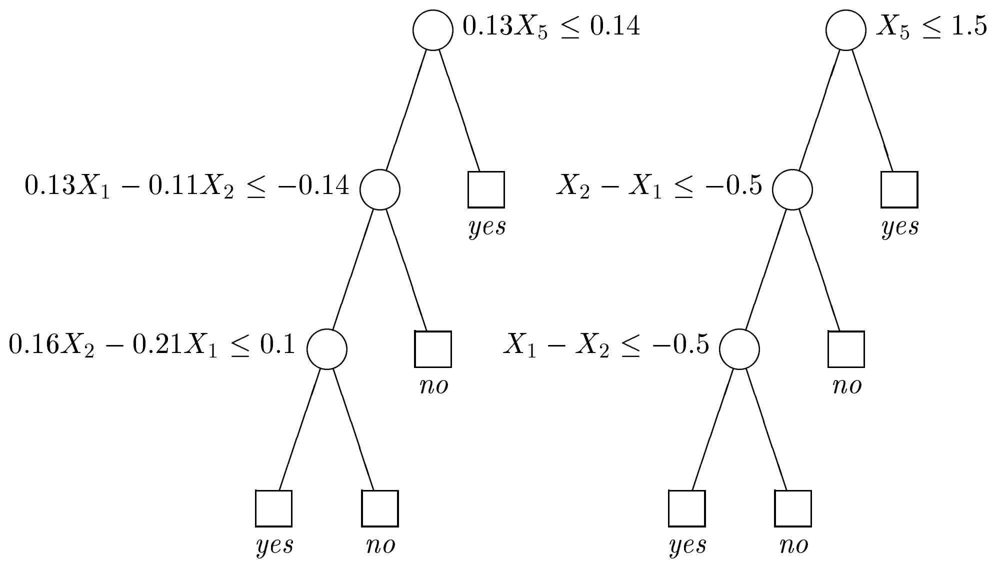
$$\frac{X_1 - 1}{2 - 1} + \frac{X_2 - 3}{187 - 3} \leq 1.5$$

or $X_1 + 0.00543X_2 \leq 2.5163$.

4 Capturing the model

In this section we will investigate whether our proposed bivariate decision tree method is able to recover an underlying model from an artificial dataset, in which interactions between two variables occur. We will test our method on two artificial data sets. The first is the monk 1 data set [Thr91]. This data set contains 6 attributes and two classes. The rule that generates the data is: if $(x_1 = x_2$ or $x_5 = 1)$ then *yes* else *no*. Note that x_5 takes on values from $[1, 2, 3, 4]$ and x_1 and x_2 from $[1, 2, 3]$. The bivariate methods produce the trees shown in figure 6, which capture the rule perfectly. In the BIT2 tree, the first node represents the $x_5 = 1$ condition as $x_5 \leq 1.5$, and the $x_1 = x_2$ condition requires two tests.

As a second example we constructed a data set again with six attributes and two classes. The rule that generates the data is: if $(x_1 \leq x_2)$ and $(x_4 \leq x_6)$ then *no* else *yes*. A univariate decision tree approximates this rule with a series of orthogonal splits, resulting in a very large tree containing about 40 leaves. The tree produced by OC1, on the other



(a) BIT1 Tree

(b) BIT2 Tree

Figure 6: Trees for the monk1 example

hand, contains only one test:

$$1.3450x_1 - 0.6333x_2 - 0.2214x_3 + 0.8779x_4 + 0.5667x_5 - 1.1432x_6 \leq 0.8883.$$

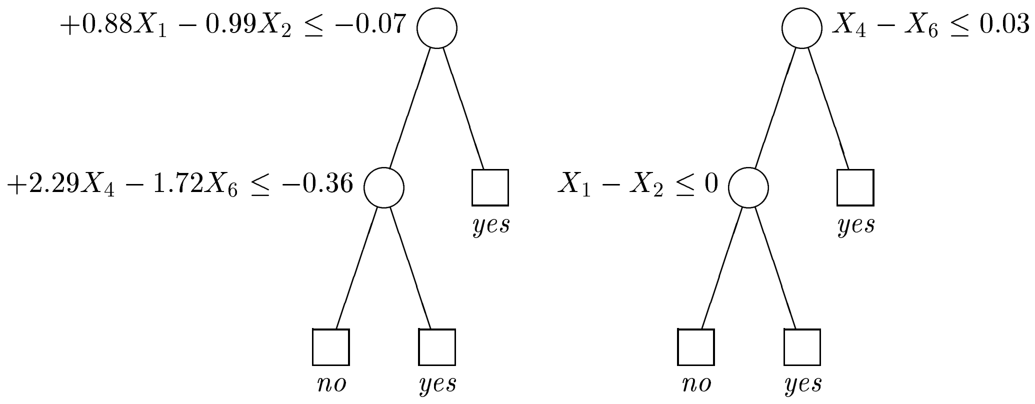
It is hard to see the structure of the problem in this representation. The bivariate methods can represent this problem exactly with only two tests, and do not require pruning. Notice that the tree resulting from the BIT2 method is more accurate than both the BIT1 tree and the multivariate tree produced by OC1. Figure 7 shows the trees produced by the bivariate methods.

5 Experiments

5.1 Experimental Setup

We used the following experimental setup for the bivariate tree methods BIT1 and BIT2, and OC1 (uni– and multivariate). We performed ten complete tenfold cross validations for each data set. In each tenfold cross validation the data was split into ten disjoint partitions of 90 percent training data and ten percent testing data, with each method using the same data. The final results are the mean values obtained by averaging over ten such cross validations.

We set up the methods mentioned to use the same pruning procedure and the same cross validation partitions of the data. The pruning procedure was cost–complexity pruning with the 1-Standard Error rule [BFOS84] using ten percent of the training data. Again, the data used for pruning was identical for all four methods. All methods used the Gini Index as the impurity measure. All attributes in the datasets were scaled to a range of $[0, 1]$.



(a) BIT1 Tree (b) BIT2 Tree

Figure 7: Trees for the second example

A further comparison was made with the results from a similar cross–validation experiment as given by Quinlan [Qui96] for C4.5 release 8.

5.2 Description of the Datasets

This section briefly describes the five data sets used in the experiments (see table 1). All of them are well-known; for instance they are among the datasets used in [Qui96]. The datasets were obtained from the UCI Repository of Machine Learning Databases [MA92].

1. Glass data: the aim here is to correctly identify a piece of glass as belonging to one of six possible classes, based on the refractive index and chemical composition of the glass.

2. Diabetes data: contains records of signs of diabetes in Pima Indian females over 21 years based on eight continuous attributes.

3. Breast cancer: the data was compiled by Dr. William H. Wolberg at the University of Wisconsin Hospitals [MW90] and has nine attributes and a diagnosis of the tumor as benign (65.5% of the cases) or malignant (34.5%).

4. Heart disease data: tests presence (44.4% of the cases) or absence of heart disease based on 13 patient characteristics.

5. Wave data: this data set was constructed by Breiman et al [BFOS84] as an example, and has 21 variables and three classes and is quite difficult to learn.

5.3 Results

The results from the experiments are summarized in table 2 for the pruned trees and table 3 for the unpruned trees. The tables give the mean accuracy and mean tree size from ten cross–validation runs, together with the standard error. From these table we can conclude

name	cases	attr	classes
glass	214	9	6
diabetes(pima)	768	8	2
breast cancer	699	9	2
heart	270	13	2
wave	300	21	3

Table 1: Summary of the Datasets

method	glass	diabetes	cancer	heart	wave
BIT1	65.3±1.1	74.3±0.7	95.4±0.3	78.5±0.3	76.1±1.3
	6.2±2.1	5.2±2.5	2.8±0.2	4.1±0.5	5.0±1.6
BIT2	64.8±0.9	74.7±0.7	95.4±0.3	78.5±0.3	76.2±1.0
	6.8±2.5	5.8±2.7	2.6±0.2	4.1±0.5	5.2±1.3
oc1 - uni	63.8±2.8	73.8±1.1	94.5±0.8	75.9±0.8	69.1±1.8
	8.1±4.5	8.4±6.3	6.4±2.2	5.2±1.1	4.8±2.1
oc1 - mlt	64.0±0.9	74.2±0.8	95.0±0.5	77.4±0.4	78.2±1.4
	7.9±2.1	4.3±3.6	2.8±0.9	3.9±0.7	3.8±0.9
c4.5	68.2±0.8	74.6±0.3	94.7±0.2	77.0±0.2	72.7±0.3
	45.7±0.4	44.0±1.6	25.0±0.5	19.1±0.6	44.6±0.4

Table 2: Accuracy and Size for Pruned Trees

that the bivariate methods often perform much better than univariate trees, while the size is usually quite close to the multivariate trees. The BIT2 methods only performs slightly worse than the BIT1 method on the glass data. As the BIT1 method is computationally much more expensive, the BIT2 method would be more useful in practice.

6 Minimal Cost-Complexity Pruning

Cost-complexity pruning operates by recursively constructing a series of trees of decreasing size by cutting the *weakest link* from the previous tree, starting from the full tree constructed from the training data. To implement cost-complexity pruning it is necessary to have separate pruning data. This can be achieved by cross-validation pruning (see breiman) or by setting a subset of the training data apart for pruning; the latter is the approach we discuss here.

The weakest link is the node $\hat{t}$ in the tree that has the minimal value of g, where g is defined as follows:

$$g(t) = \frac{\text{Cost of Node} - \text{Cost of Subtree}}{\text{Number of Leaves of Node} - 1} \tag{5}$$

Here the cost of a node is the number of wrongly classified points at the node divided by the total number of data points in the training set, and the cost of a subtree of a node is sum of the node-costs of the children of the node. Figure 8 displays a tree before its weakest link is cut; the node-cost, subtree-cost and number of leaves of the decision-nodes is shown.

	glass	diabetes	cancer	heart	wave
BIT1	65.9±1.3	70.9±0.9	94.4±0.5	73.5±0.5	68.6±1.8
	24.3±2.1	75.3±1.2	8.2±0.3	15.1±0.6	14.9±0.8
BIT2	63.3±1.3	70.2±1.4	93.1±0.4	73.0±0.4	69.0±2.1
	27.8±2.4	75.5±2.3	11.1±0.3	19.9±0.5	15.1±1.1
oc1 - uni	66.8±3.0	70.5±4.3	93.9±0.9	71.5±0.8	68.6±2.7
	44.6±4.8	122.4±8.5	20.5±2.1	41.9±0.9	36.3±1.3
oc1 - mlt	66.3±1.5	69.9±1.2	91.2±1.1	79.3±0.5	71.0±1.2
	28.4±1.3	50.3±4.2	16.1±0.8	21.2±0.8	24.2±0.8
c4.5	67.5±0.9	74.2±0.4	94.2±0.2	77.6±0.2	72.6±0.3
	45.5±0.4	52.3±1.3	41.4±0.3	48.4±0.5	45.4±0.4

Table 3: Accuracy and Size for Unpruned Trees

After finding the value of g for each node, the node with the lowest g is *leavified*: the subtree of the node is cut and the node becomes a leaf corresponding to the largest class at the node. The new tree that is the result of cutting the weakest link is again subjected to this procedure, until a tree with one leaf is left. The result of cutting the weakest link from the tree of figure 8 is displayed in figure 9.

In the final step the accuracy of the trees on the pruning set is used to select the best tree. As an example, the following table shows a series of trees, and for each tree the error on the pruning data and the number of leaves.

tree#	1	2	3	4	5	6	7	8
error (R)	0.382	0.382	0.353	0.324	0.324	0.353	0.353	0.353
size	15	13	12	7	6	5	4	2

If we use the 1-Standard Error rule, we first find the lowest error rate, $\hat{R} = 0.324$ in this case, and calculate the standard error as $\sqrt{\hat{R}(1-\hat{R})/N_t}$, where N_t is the size of the pruning set. In this case this gives $\mathrm{SE}(\hat{R}) = 0.082$. The tree selected by the 1-Standard Error rule is the smallest tree that ratifies $R \leq \hat{R} + \mathrm{SE}(\hat{R})$. In this example the tree selected would be tree 8.

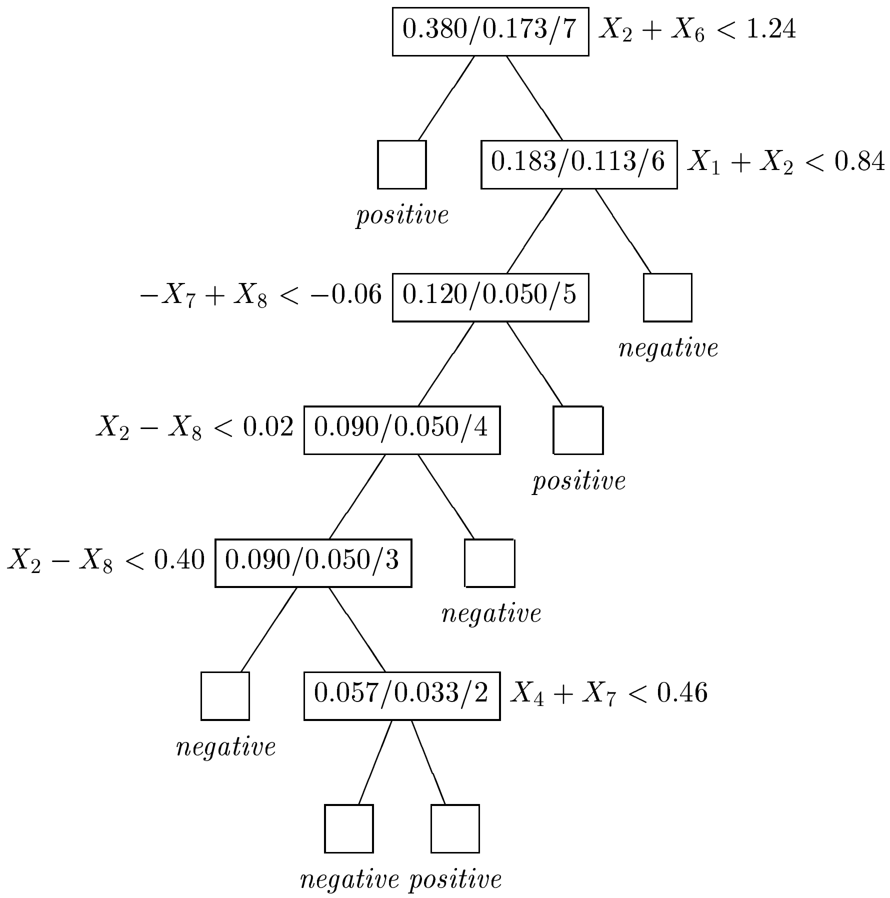


Figure 8: Tree before cutting the weakest link

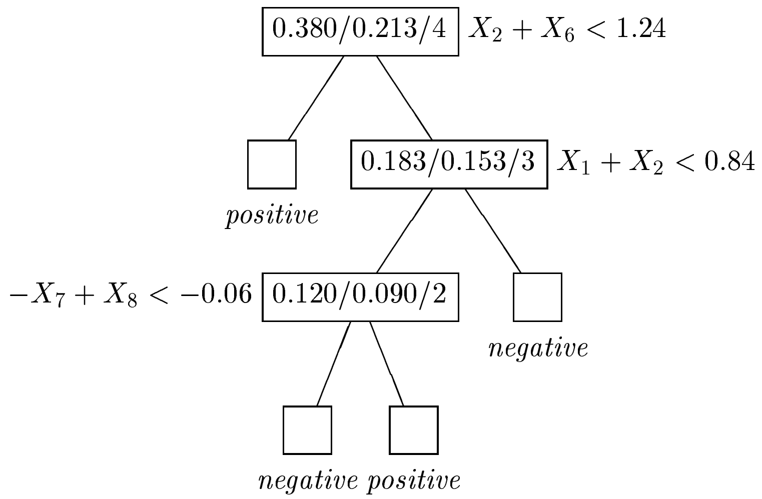


Figure 9: Tree after cutting the weakest link (see figure 8)

7 Conclusion

We have discussed two methods of constructing bivariate decision trees, in which per test node at most two attributes are used. We have shown that bivariate tree methods can improve the interpretability of the resulting trees (both small trees and simple tests), with accuracy at least as good as C4.5. We found both the BIT1 and the BIT2 method to give good results. In addition, despite its simplicity BIT2 proved to be quite effective. Topics for further research: we are currently investigating the issue of scaling for the BIT2 method. This method could also be extended to include interaction terms of the form $X_i X_j$.

Appendix: BIT2 Trees for the datasets

In this appendix we have collected a number of trees that were generated in the course of our experiments. They are meant to be illustrative examples of the kind of trees that we saw passing.

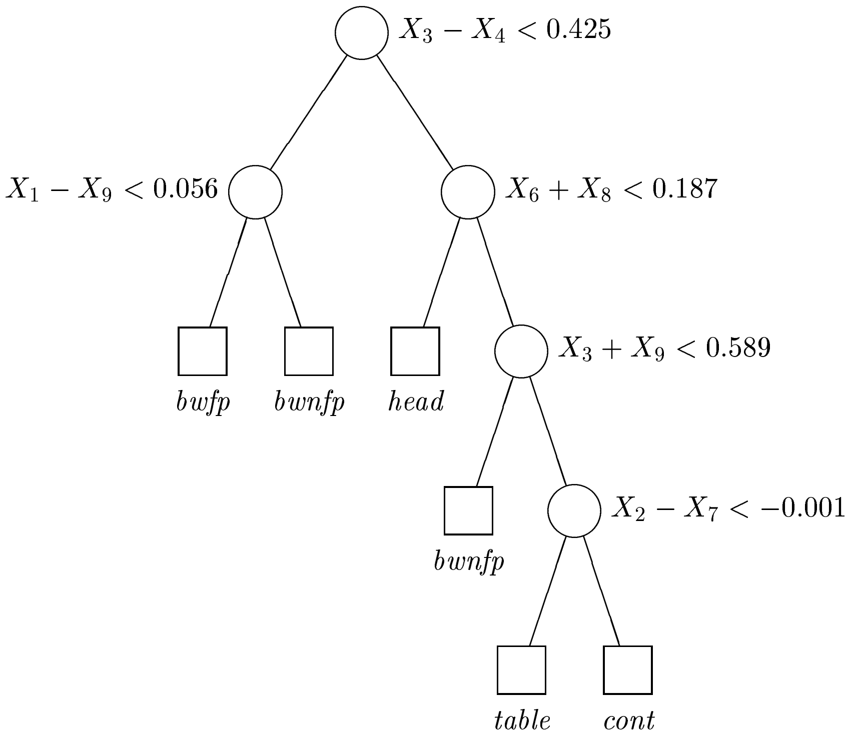


Figure 10: BIT2 tree for the scaled glass data

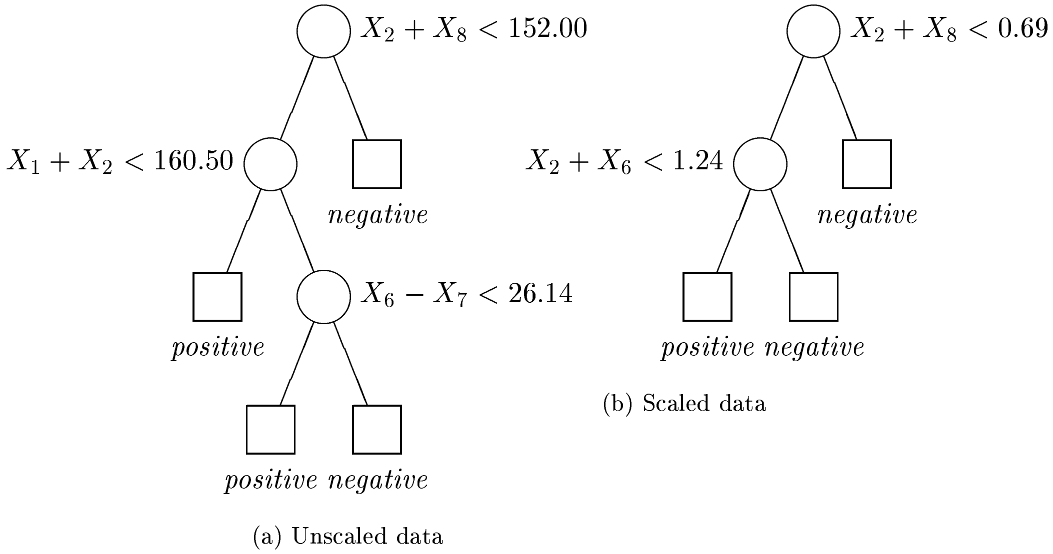


Figure 11: BIT2 trees for the diabetes data: the effect of scaling. Both trees used the same train/prune/test set partition; however the left tree is constructed from unscaled data, whereas the right tree used scaled data. The left is both larger and less accurate.

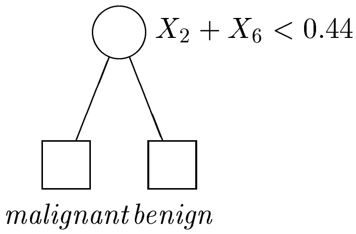


Figure 12: BIT2 tree for the scaled cancer data

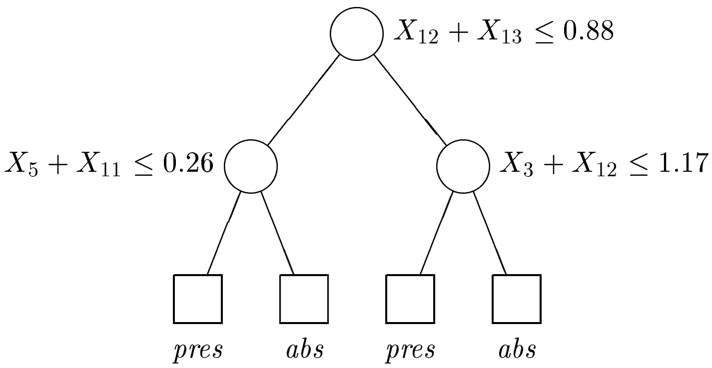


Figure 13: BIT2 tree for the scaled heart data

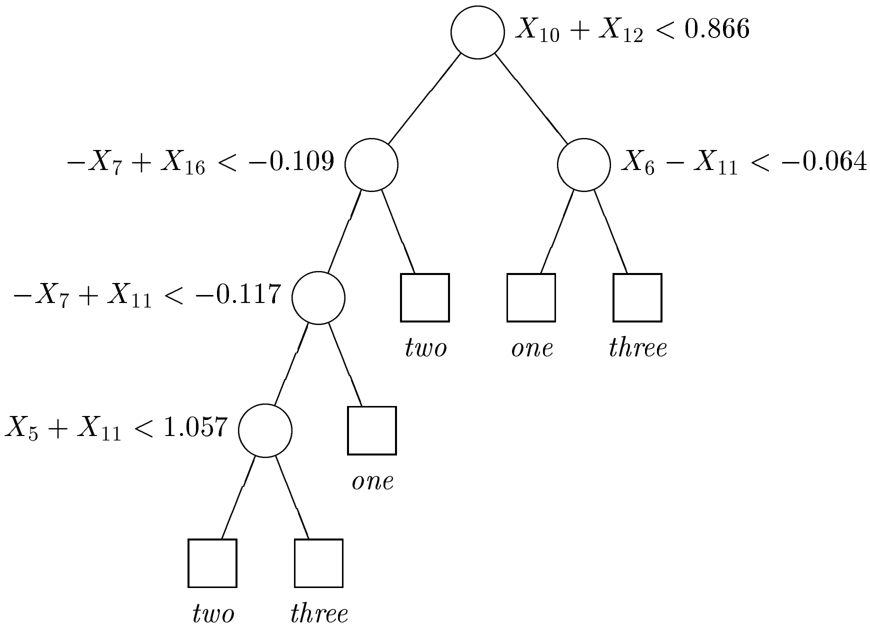


Figure 14: BIT2 tree for the scaled wave data

References

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth (reprinted by Chapman & Hall), 1984.

[BU95] C.E. Brodley and P.E. Utgoff. Multivariate decision trees. *Machine Learning*, pages 45–77, 1995.

[FI93] U.M. Fayyad and K.B. Irani. Multi–interval discretization of continuous–valued attributes for classification learning. In *Proceedings Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022–1027. San Fransisco: Morgan Kaufmann, 1993.

[MA92] P.M. Murphy and D.W. Aha. UCI repository of machine learning databases and domain theories. Technical report, University of California, 1992. Web Page: `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[MKSB93] S. Murthy, S. Kasif, S. Salzberg, and R. Beigel. OC1: Randomized induction of oblique decision trees. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 322–327, 1993.

[Mur95] S. Murthy. *On Growing Better Decision Trees from Data.* PhD thesis, The John Hopkins University, Baltimore, 1995.

[MW90] O.L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1–18, 1990.

[Qui96] J.R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[Rip96] B.D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.

[Thr91] S. Thrun. Comparison of machine learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, Department of Computer Science, 1991.

[UB91] P.E. Utgoff and C.E. Brodley. Linear machine decision trees. Technical Report COINS Technical Report 91–10, University of Massachusetts, Department of Computer and Information Science, 1991.