

Knowledge Acquisition from Databases

Xindong Wu
Monash University, Australia

Chapter 4

Constructing Decision Trees With ID3 and C4.5

4.1 DEVELOPER AND BACKGROUND

ID3 [Quinlan 79] is a top-down algorithm developed by Quinlan out of the Concept Learning System (CLS) by Hunt [Hunt et al. 66].

CLS is a learning mechanism that accepts a set of training pairs and constructs a representation in the form of a decision tree, which is equivalent to a disjunctive rule. The decision tree is structured so that each leaf node has a target output associated with it. An arbitrary input is processed by simply applying the tree to the input (i.e., propagating the input down through the tree). This produces a leaf node, which in turn yields the target output.

Main steps in the CLS algorithm can be described as follows:

S1: $T \leftarrow$ the whole training set.

 Create a T node.

S2: If all examples in T are positive, create a 'yes' node with T as its parent and stop.

S3: If all examples in T are negative, create a 'no' node with T as its parent and stop.

S4: Select an attribute X with values v_1, \dots, v_N and partition T into subsets T_1, \dots, T_N according to their values on X.

 Create N T_i nodes ($i = 1, \dots, N$) with T as their parent and $X = v_i$ as the label of the branch from T to T_i .

S5: For each T_i do: $T \leftarrow T_i$ and go to S2.

4.2 THE ID3 ALGORITHM

Quinlan modified the CLS algorithm in two ways. First, he added a process known as *windowing*. This was designed to enable the algorithm to cope with very large training sets.

If the training set is very large, then rather than process the entire set in one, it may be more efficient to process a small sample first. If the sample is a representative of the complete set, the decision tree produced will be similar to the one we would get by processing the entire training set. Once we have produced a tentative tree we can then gradually perfect it. To do this we simply search through the training set looking for any *(input, output)* pairs that are not properly represented and each time we find such an exception we modify the tree appropriately. However, in recent work, windowing does not feature very strongly. Some evidence [Wirth & Catlett 88] suggests that windowing typically provides very little benefit.

Second, and more importantly, Quinlan devised an information theoretic heuristic (the *entropy* or *information gain* measure) that decided how to split the inputs at each stage of the tree-growing process, thus enabling smaller and therefore more efficient decision trees to be constructed.

ID3 works as follows.

Suppose $T = PE \cup NE$ where PE is the set of positive examples and NE is the set of negative examples, $p = |PE|$ and $n = |NE|$. An example e will be determined to belong to PE with probability $p/(p+n)$ and NE with probability $n/(p+n)$. By employing the information theoretic heuristic, a decision tree is considered as a source of message, “ PE ” or “ NE ,” with the expected information needed to generate this message, given by

$$I(p, n) = \begin{cases} -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n} & \text{when } p \neq 0 \text{ and } n \neq 0 \\ 0 & \text{otherwise.} \end{cases} \quad (4.1)$$

If attribute X with value domain $\{v_1, \dots, v_N\}$ is used for the root of the decision tree, it will partition T into $\{T_1, \dots, T_N\}$ where T_i contains those examples in T that have value v_i of X . Let T_i contain p_i examples of PE and n_i of NE . The expected information required for the subtree for T_i is $I(p_i, n_i)$. The expected information required for the tree with X as root, $EI(X)$, is then obtained as weighted average.

$$EI(X) = \sum_{i=1}^N \frac{p_i + n_i}{p + n} I(p_i, n_i) \quad (4.2)$$

where the weight for the i -th branch is the proportion of the examples in T that belong to T_i . The information gained by branching on X , $G(X)$, is therefore

$$G(X) = I(p, n) - EI(X). \quad (4.3)$$

ID3 examines all candidate attributes, chooses X to maximize $G(X)$, constructs the tree, and then uses the same process recursively to construct decision trees for residual subsets T_1, \dots, T_N . For each T_i ($i = 1, \dots, N$): If all the examples in T_i are positive, create a “yes” node and halt; if all the examples in T_i are negative, create a “no” node and halt; otherwise select another attribute in the same way as given earlier.

The maximum number of leaf nodes in the decision tree is $p + n$ (the total number of examples in the training set) and the maximum length from the root to each leaf node is a where a is the number of attributes. So the total number of nodes in the decision tree is always less than $a(p + n)$. At the root, ID3 needs to check each example’s value on each attribute X to find out $G(X)$. The time complexity for this is $O(ba(p + n))$ where b is the maximum number of possible values for an attribute. Time complexity at other nodes is always less than that at the root. Therefore, the worst time complexity for ID3 is $O(ba(p + n)a(p + n)) = O(ba^2(p + n)^2)$.

4.3 AN EXAMPLE: PLAY AND DON’T PLAY

The decision tree generated by ID3 for the example set in Table 4.1 is shown in Figure 4.1.

At the root of the decision tree, $T = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}\}$, $p = 7$ and $n = 7$. We have four candidate attributes, $\{OUTLOOK, TEMPERATURE, HUMIDITY, WINDY\}$.

$$I(7, 7) = -\frac{7}{7+7} \log_2 \frac{7}{7+7} - \frac{7}{7+7} \log_2 \frac{7}{7+7} = 1$$

$$EI(OUTLOOK) = \frac{3+3}{14} I(3, 3) + \frac{3+0}{14} I(3, 0) + \frac{1+4}{14} I(1, 4) = 0.69$$

$$EI(TEMPERATURE) = \frac{2+3}{14} I(2, 3) + \frac{2+2}{14} I(2, 2) + \frac{3+2}{14} I(3, 2) = 1.24$$

Table 4.1. Cases of *Play* and *Don't Play* (adapted from [Quinlan 86b])

<i>Order</i>	<i>Outlook</i>	<i>Temperature</i>	<i>Humidity</i>	<i>Windy</i>	Decision
1	rain	hot	high	true	<i>Don't Play</i>
2	rain	cool	normal	true	<i>Don't Play</i>
3	overcast	mild	high	true	<i>Play</i>
4	overcast	mild	normal	false	<i>Play</i>
5	rain	hot	high	false	<i>Play</i>
6	overcast	cool	normal	true	<i>Play</i>
7	sunny	hot	normal	true	<i>Don't Play</i>
8	sunny	mild	high	true	<i>Don't Play</i>
9	sunny	mild	normal	false	<i>Play</i>
10	rain	cool	normal	false	<i>Play</i>
11	rain	hot	high	false	<i>Play</i>
12	sunny	hot	high	false	<i>Don't Play</i>
13	sunny	cool	normal	false	<i>Don't Play</i>
14	rain	mild	normal	true	<i>Don't Play</i>

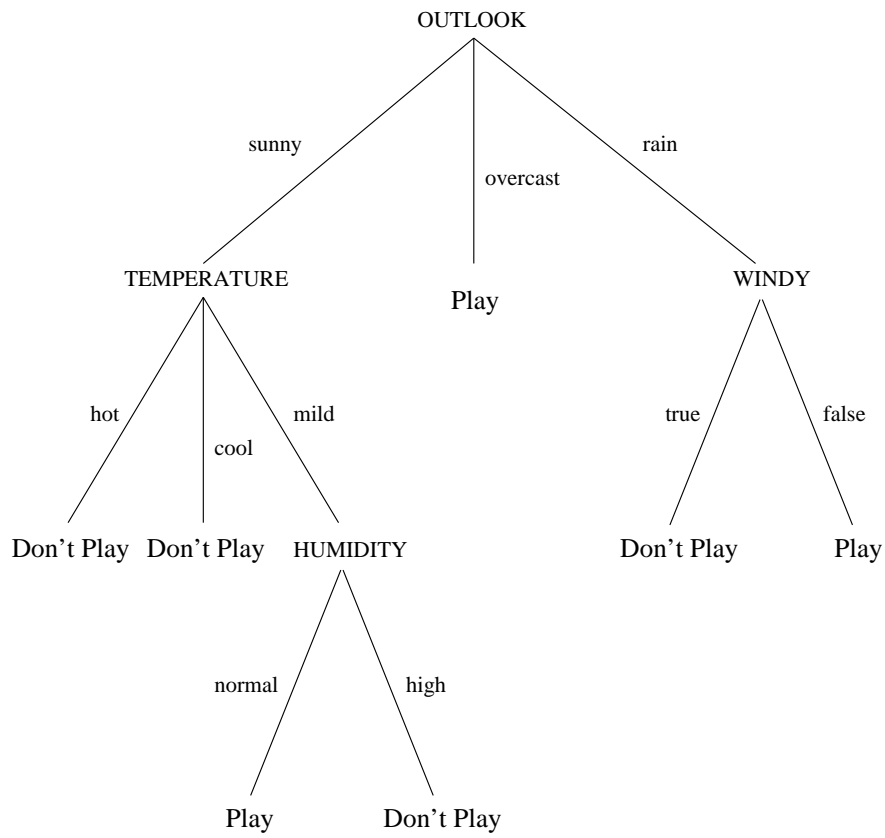


Figure 4.1. A decision tree (by ID3) for Table 4.1.

$$EI(HUMIDITY) = \frac{3+3}{14}I(3,3) + \frac{4+4}{14}I(4,4) = 1$$

$$EI(WINDY) = \frac{2+5}{14}I(2,5) + \frac{5+2}{14}I(5,2) = 1.15$$

Because $I(7,7)$ is the same for $G(OUTLOOK)$, $G(TEMPERATURE)$, $G(HUMIDITY)$, and $G(WINDY)$, maximizing $G(X)$ is equivalent to minimizing $EI(X)$. $OUTLOOK$ is chosen in this case to split the 14 examples into three subsets: $T1 = \{e_1, e_2, e_5, e_{10}, e_{11}, e_{14}\}$, $T2 = \{e_3, e_4, e_6\}$, and $T3 = \{e_7, e_8, e_9, e_{12}, e_{13}, e_{14}\}$.

As an exercise, the reader is advised to work out the tree by following the algorithm described in Section 4.2.

4.4 ADVANTAGES AND DISADVANTAGES

One of the great advantages of the ID3 method is the fact that it does not require users to specify background knowledge in the form of, say, generalization hierarchies.¹ This means that ID3 can be applied to any syntactically well-formed training set. This, together with the high performance of the algorithm, has enabled ID3 to form the central component in several commercial packages.

However, ID3 also has some limitations. First, its decision tree representation is less convenient for manipulations than the variable-valued logic (see footnote 2 of Chapter 2) and production rules. When a single decision tree is not sufficient to represent all the expertise of a domain, a number of decision trees of the domain need to be converted into decision rules before they can be used by a rule-based system. Although the conversion of decision trees to rules is not very difficult if we do not try to simplify the rules produced (see Section 4.5.5), the rules transformed from decision trees are still too simple to express things like memberships (\in). There has been a long-term dispute that decision trees are too simple to represent the real world. We can accommodate numerical computation and uncertainty calculus into a

¹Núñez argued that for this reason, most of the time the ID3 family of algorithms are neither logical nor understandable to experts and therefore he made some improvements (i.e., executing different types of generalization and reducing the classification cost) on ID3 in his algorithm by means of background knowledge in [Núñez 91].

rule-based representation, as seen in Chapter 8, but it is not easy to do so with decision trees. Of course, for those domains where a decision tree is sufficient to express the expertise, we can use the decision tree directly by designing a non-rule-based problem solver.

Second, once an attribute is selected, all arcs labeled by values that attribute takes must be expanded. This can make resulting paths longer than those actually needed because, by the time specific concepts (leaves on the decision tree) are developed, irrelevant variables may have been introduced. Also, because at each arc only one value can be labeled, the number of branches (paths) is large.

Third, the ID3 algorithm is sensitive to the order of attributes (see footnote 1 of Chapter 2) and the redundancy of examples. When some examples appear more than once, the final decision tree could be a very different one.

Finally, although the information theoretic heuristic can usually generate efficient decision trees, ID3 is still heuristic, which means it is not guaranteed to find the simplest decision tree that characterizes the given training instances because the information theoretic heuristic is by no means complete, suffers from excessive complexity [Utgoff 89], and is therefore usually incomprehensible to experts because it needs to examine all candidate attributes to choose one at each nonleaf node.

4.5 RECENT DEVELOPMENT OF ID3

The original ID3 algorithm has been extended in several ways to improve its various capacities such as noise handling and incremental induction in its successors such as C4.5 [Quinlan 93] and ID5R [Utgoff 89]. This chapter gives an account of these developments. Further details of noise handling are given in Chapter 6.

4.5.1 Noise Handling During Induction

ID3 can be easily adapted to handle noise by virtue of its top-down approach to decision tree construction. During induction, all possible attribute tests are considered when growing a leaf in a decision tree and the entropy measure is used to choose one to place at each node. In noisy environments, we can halt tree growth when no more significant information gain can be found. ID3's capacity to handle noisy data has been studied in

[Quinlan 86a] and [Quinlan et al. 87]. Noise handling in decision tree based induction algorithms has also been studied independently as a statistical technique [Breiman et al. 84] and shows a convergence between machine learning research in AI and statistics [Gams et al. 91].

4.5.2 Incremental Learning

There are several common problems in all kinds of inductive learning algorithms:

1. First, when an example set is very large, how can they speed up their learning processes?
2. Second, when an example set is not a static repository of data, for example, examples may be added, deleted, or changed, the induction on the example set cannot be a one-time process, so how can induction algorithms deal with the changing examples?
3. Finally, when some inconsistency (e.g., noise) is found in an example set or a knowledge base just produced, how can they remove it?

One possible way to solve those problems is incremental learning, which means dividing a large example set into a number of subsets and treating each subset each time. Although no existing algorithm has found a complete solution to these problems, a lot of work has been done in this direction. For instance, ID4 [Schlimmer & Fisher 86], ID5R [Utgoff 89], and the windowing technique in ID3 can be viewed as good examples of research on incremental learning.

Generally speaking, incremental induction usually takes more time because it needs to restructure decision trees or rules when some new examples do not fit the decision trees or rules developed so far. This is a common trade-off between time and space in computer science.

4.5.3 Constructive Learning

None of ID3-like algorithms need explicit, built-in background knowledge. That is why they are sometimes called *empirical learning* methods, which are different in nature from the knowledge-rich learning methods, such as AM [Lenat 79] and EURISKO [Lenat 83] developed by Lenat, learning by analogy, explanation-based learning, and inductive logic programming.

However, there is always implicit background knowledge embedded in the formulation of solution spaces and in the representation of examples. When a solution space turns out to be inadequate, which is often called *the imperfect knowledge problem*, representation modification is needed and the modification process typically involves searching for useful new descriptive features (constructive induction) in terms of existing features or attributes.

Constructive learning (including constructive induction of decision trees, [Matheus 89, Matheus et al. 89, Michalski 86, Wnek & Michalski 94]) has become a strong theme in inductive learning research. One of the difficulties in constructive learning is that the complexity in some cases (such as iterative feature construction) is extreme, but there are situations in which it is a necessary part of learning.

4.5.4 Postpruning of Decision Trees

The basic ID3 algorithm tends to construct exact decision trees. However, in many real-world problems such as medical diagnosis and image recognition, the classification cannot be exact due to noise and/or uncertainty in data. As a result, a tree constructed by ID3 may not be able to capture the proper relations in data. Decision tree pruning mechanisms have been designed in many systems such as CART [Breiman et al. 84], ASSISTANT [Cestnik et al. 87], and C4.5 [Quinlan 93] to prevent this phenomenon. Once a nonleaf subtree meets a specific criterion (e.g., with an equal or smaller number of misclassifications), it is replaced by a leaf.

Pruning can usually simplify decision trees. The simplified trees can normally classify more accurately unseen cases in noisy environments.

4.5.5 Decompiling Decision Trees Into Production Rules

Decompiling decision trees has been implemented in C4 [Quinlan et al. 87] and C4.5 [Quinlan 93]. It contains three basic steps [Corlett 83, Quinlan 87a]:

1. Traverse a decision tree to obtain a number of conjunctive rules. Each path from the root to a leaf in the tree corresponds to a conjunctive rule with the leaf as its conclusion.
2. Check each condition in each conjunctive rule to see if it can be dropped without more misclassification than expected on the original training examples or new test examples.

3. If some conjunctive rules are the same after Step 2, then keep only one of them.

Transformation of decision trees to production rules provides a way of combining different trees into the same knowledge base for more complicated domains. The final production rules produced are expected to be both simpler than the original decision trees and more accurate when classifying new examples in noisy environments.

However, dropping conditions from the decision-tree-traversal rules in Step 2 is something like a new induction algorithm that can work on the original example sets but in a way totally different from the ID3-like algorithms. Therefore, the time complexity for the transformation is expensive.

When converting the decision tree in Figure 4.1 of Chapter 4 to production rules, we get the following results:

If *OUTLOOK = overcast* then *Play*.
 If *OUTLOOK = rain* and *WINDY = true* then *Don't Play*.
 If *OUTLOOK = rain* and *WINDY = false* then *Play*.
 If *OUTLOOK = sunny* and *TEMPERATURE = hot* then *Don't Play*.
 If *OUTLOOK = sunny* and *TEMPERATURE = cool* then *Don't Play*.
 If *OUTLOOK = sunny* and *TEMPERATURE = mild* and *HUMIDITY = normal* then *Play*.
 If *OUTLOOK = sunny* and *TEMPERATURE = mild* and *HUMIDITY = high* then *Don't Play*.

Here, no conditions can be dropped from the decision-tree-traversal rules. In this case, Step 2 is redundant.

4.5.6 Binarization of Decision Trees

Binarization in CART [Breiman et al. 84], ASSISTANT [Cestnik et al. 87], and NewID [Boswell 90] groups the attribute values into two subsets. In the binary trees, each nonleaf node has exactly two child nodes. When an attribute has more than two values, the values have to be split into two groups.

Binarization can usually produce smaller decision trees. However, as indicated in [Quinlan 88b], there are two major problems in those systems. First, binarization could lead to decision trees that are even more unintelligible to human experts than the ordinary case due to unrelated attribute values being grouped together and multiple tests on the same attributes in the binary decision trees. Second, binarization requires a large increase in computation to properly split the attribute values.

4.5.7 A New Selection Criterion for Decision Tree Construction

As ID3 has been found to operate unsatisfactorily when there are attributes with varying numbers of discrete possible values, [Quinlan 88b] proposed a new heuristic, called *the gain ratio criterion*, instead of the entropy measure, $G(X)$ (see Section 4.2), adopted in ID3 for selecting tests in decision tree generation. In the gain ratio criterion, $G(X)/IV(X)$ is used to replace $G(X)$ where

$$IV(X) = \sum_{i=1}^N \frac{p_i + n_i}{p + n} \log_2 \left(\frac{p_i + n_i}{p + n} \right) \quad (4.4)$$

and N , p , and n are as mentioned in Section 4.2 of Chapter 4. When $IV(X)$ is not zero, [Quinlan 88b] suggested that from among those attributes with an average-or-better gain, select the attribute that maximizes the above ratio.

Although it is believed that the new gain ratio criterion can typically outperform the entropy measure, however, in many cases, even when there are attributes with varying numbers of discrete possible values, the new criterion cannot improve the decision trees produced by ID3 at all. Figures 4.2 and 4.3 show two decision trees produced by the new gain ratio heuristic and the entropy measure, respectively, on the same example set in Table 4.2. The decision tree produced by the new criterion (Figure 4.3) is a little more complicated than the decision tree produced by the original ID3 algorithm (Figure 4.2): Both need seven conjunctive rules (paths from the root to terminals in each decision tree) but the new criterion needs one more conjunction. Experiments with the MONK's problems in Section 7.3 of Chapter 7 provide further evidence in this regard. The example set in Table 4.2 also demonstrates ID3's heuristic rather than complete property: A decision tree made by hand (Figure 4.4) is clearly smaller than the tree

produced by ID3. This is quite natural because constructing the smallest decision trees is an intractable problem.

Table 4.2. Cases of T and F

<i>Order</i>	X_1	X_2	X_3	X_4	Class
1	1	a	a	1	F
2	1	a	b	1	F
3	1	a	c	1	F
4	1	a	a	0	F
5	1	b	c	1	T
6	0	b	b	0	T
7	0	a	c	1	T
8	1	b	a	0	T
9	1	b	a	1	T
10	1	c	c	0	F
11	1	c	b	1	F
12	0	c	b	0	T
13	0	a	a	0	T
14	0	c	c	1	F
15	0	c	a	0	T
16	1	a	b	0	F
17	0	a	a	1	T
18	0	b	a	1	T

The rules correspond to the decision tree in Figure 4.2 (the conditions in the **bold** type style can be dropped):

	$X_2 = b$		$X_2 = a$ and $X_1 = 1$
∨	$X_2 = a$ and $X_1 = 0$		∨ $X_2 = c$ and $X_3 = c$
∨	$X_2 = c$ and $X_3 = a$		∨ $X_2 = c$ and $X_3 = b$ and
∨	$X_2 = c$ and $X_3 = b$ and		$X_1 = 1$
	$X_1 = 0$		
→	The T class.	→	The F class.

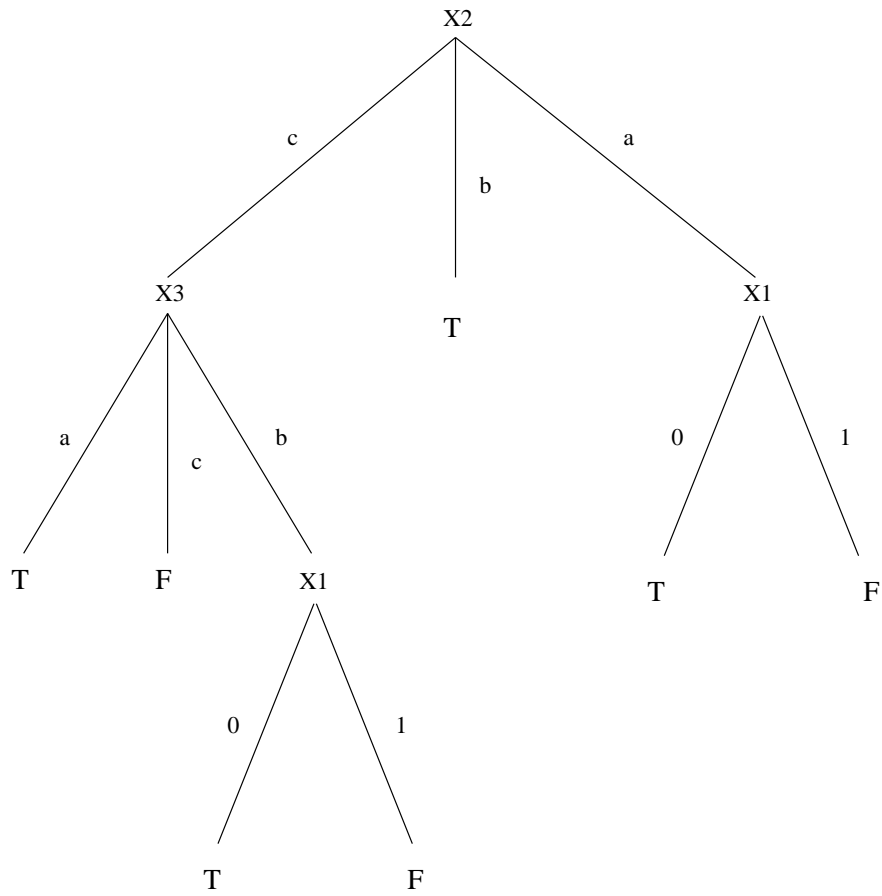


Figure 4.2. A decision tree (by ID3) for Table 4.2.

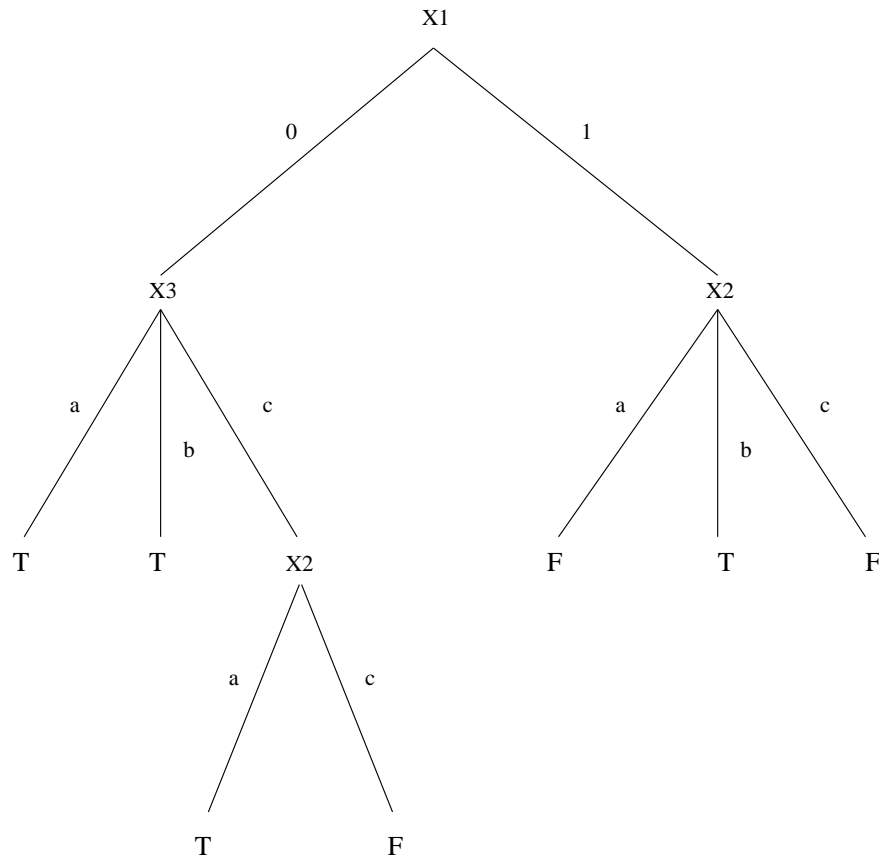


Figure 4.3. A decision tree (by the *gain ratio* heuristic) for Table 4.2.

The decision tree in Figure 4.3 is equivalent to the following decision rules:

- | | | |
|---|--------------------------------|--------------------------------------|
| | $\mathbf{X1 = 1}$ and $X2 = b$ | |
| ∨ | $X1 = 0$ and $X3 = a$ | $X1 = 1$ and $X2 = a$ |
| ∨ | $X1 = 0$ and $X3 = b$ | ∨ $X1 = 1$ and $X2 = b$ |
| ∨ | $X1 = 0$ and $X3 = c$ and | ∨ $\mathbf{X1 = 0}$ and $X3 = c$ and |
| | $X2 = a$ | $X2 = c$ |
| → | The <i>T</i> class. | → |
| | | The <i>F</i> class. |

Meanwhile, Figure 4.4 corresponds to the following decision rules.

- | | |
|--|--|
| $\mathbf{X1 = 1}$ and $X2 = b$
\vee $X1 = 0$ and $X4 = 0$
\vee $X1 = 0$ and $X4 = 1$ and
$X2 = a$
\rightarrow The T class. | $X1 = 1$ and $X2 = a$
\vee $X1 = 1$ and $X2 = c$
\vee $\mathbf{X1 = 0}$ and $X4 = 1$ and
$X2 = c$
\rightarrow The F class. |
|--|--|

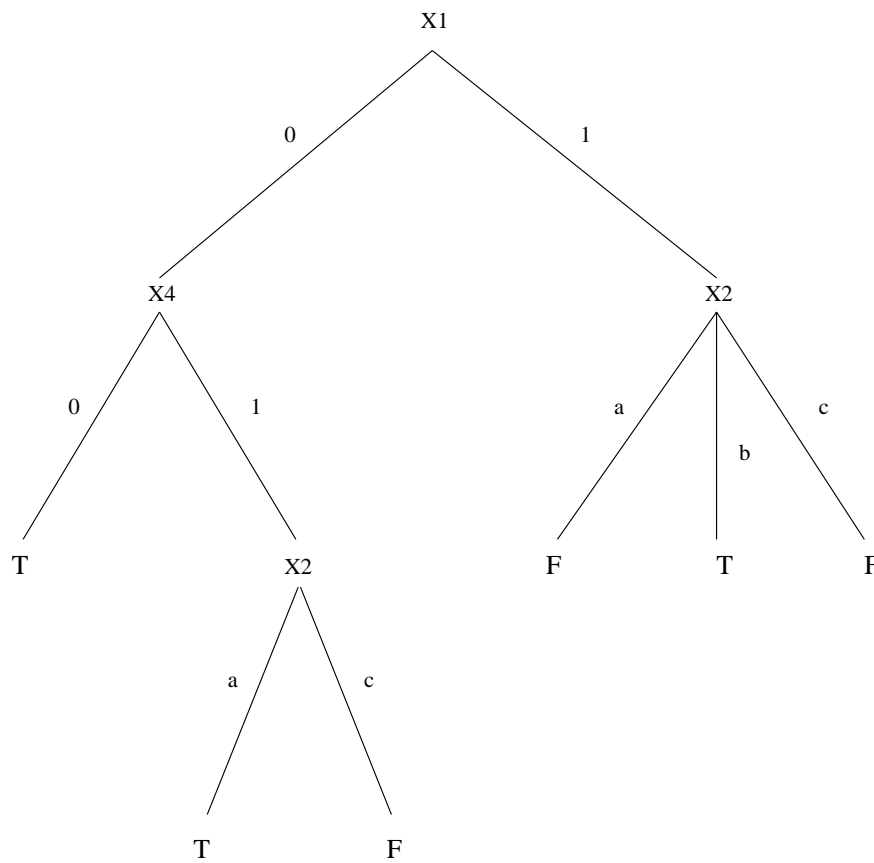


Figure 4.4. A briefer decision tree (by hand) for Table 4.2.

Figure 4.4 is clearly smaller than both Figure 4.2 and Figure 4.3 in terms of the numbers of leaves and nodes (including leaves) in decision trees or the numbers of conjunctive rules and conjunctions in the equivalent rules to the decision trees.

4.5.8 Structured Induction

ID3's simplicity is largely attributable to the following two restrictions placed on its application domains [Quinlan 88a]:

- The induction is a kind of classification, that is, the knowledge we are trying to capture is that of assigning a case to one of a set of mutually exclusive classes.
- Each case is described in terms of a fixed collection of or attributes. An attribute may have a set of discrete possible values or might be a real-valued numerical variable.

Although induction offers a considerable shortcut in comparison to those methods of rule generation such as explanation-based learning and inductive logic programming that couple both deduction and induction, decision tree-based algorithms provide large decision trees that are opaque to the user in large problem domains. Shapiro [87] developed the technique of structured induction. The basic idea is to split the whole complex problem that might be very large in size into a number of subproblems by using domain knowledge and to apply the ID3 algorithm to each of the subproblems. Shapiro's work concerned solutions to the chess endgames King and Pawn vs. King (black-to-move) and King and Pawn vs. King and Rook (white-to-move, white pawn on a 7) as trial problems of measurable complexity. The resultant systems contained humanly understandable decision trees that were synthesized from expert supplied examples.

4.5.9 Conclusions

In addition to the development already mentioned, other features such as handling real-valued attributes [Fayyad & Irani 92, Quinlan 93] have also been studied. The most recent successor of ID3 is C4.5 [Quinlan 93], whose main features can be outlined as follows:

- The gain ratio heuristic for selecting tests.
- Sensible mechanisms for handling missing information.
- Postpruning of decision trees.
- Facilities to convert sets of decision trees to production rules.