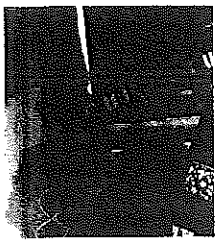
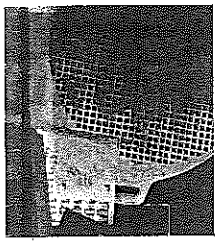
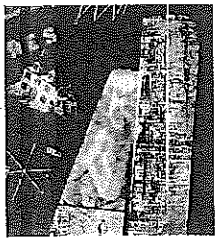




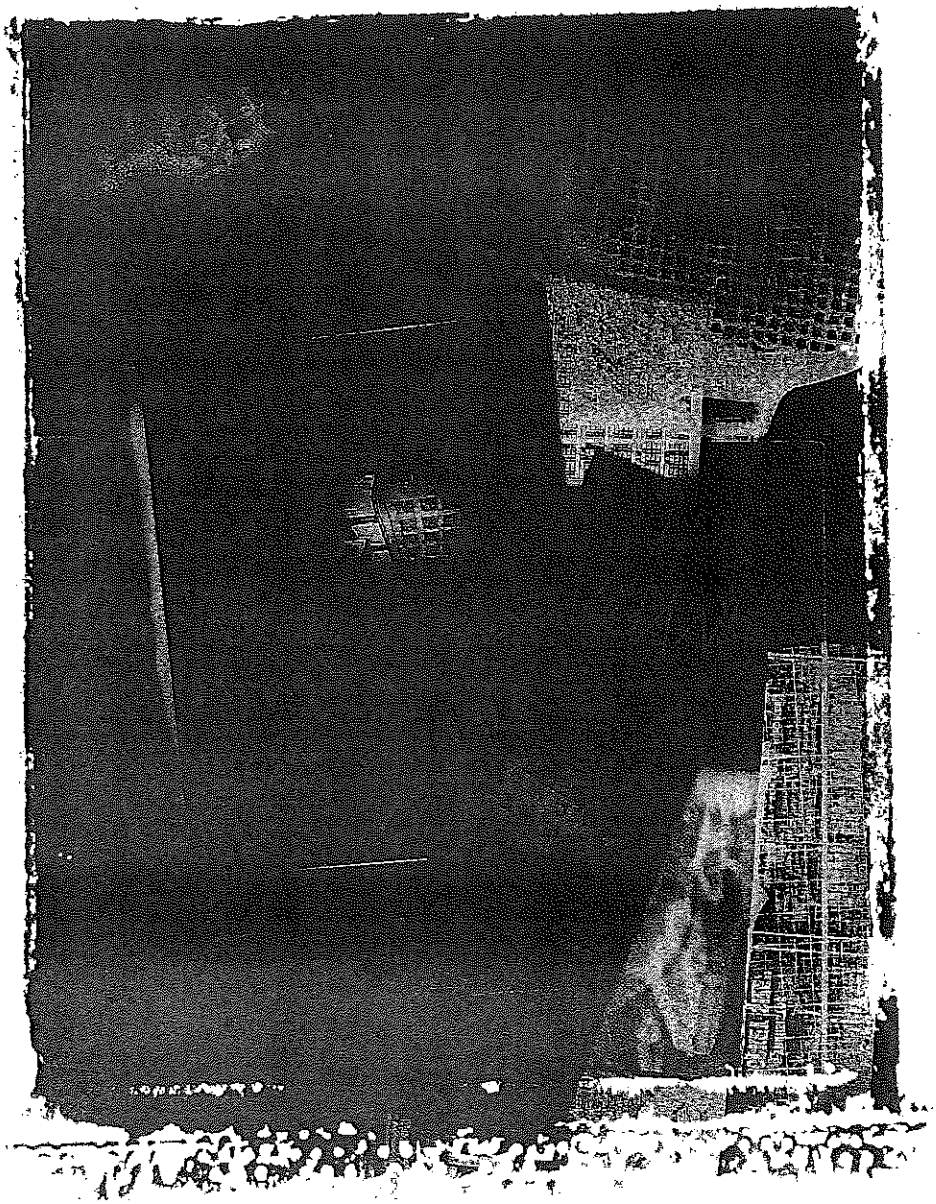


George F Luger

ARTIFICIAL INTELLIGENCE

Structures and Strategies for Complex Problem Solving

Fourth Edition



ADDISON
WESLEY

Search-based learning, like all search problems, must deal with the combinatorics of problem spaces. Because the candidate elimination algorithm performs breadth-first search, it can be inefficient. If an application is such that G and S grow excessively, it may be useful to develop heuristics for pruning states from G and S, implementing a *beam search* (see Chapter 4) of the space.

Another approach to this problem, discussed in Section 9.4, involves using an *inductive bias* to further reduce the size of the concept space. Such biases constrain the language used to represent concepts. LEX imposed a bias through the choice of concepts in its generalization hierarchy. Though not complete, LEX's concept language was strong enough to capture many effective heuristics; of equal importance, it reduced the size of the concept space to manageable proportions. Biased languages are essential in reducing the complexity of the concept space, but they may leave the learner incapable of representing the concept it is trying to learn. In this case, candidate elimination would fail to converge on the target concept, leaving G and S empty. This trade-off between expressiveness and efficiency is an essential issue in learning.

Failure of the algorithm to converge may also be due to some noise or inconsistency in the training data. The problem of learning from noisy data is particularly important in realistic applications, where data cannot be guaranteed to be complete or consistent. Candidate elimination is not at all noise resistant. Even a single misclassified training instance can prevent the algorithm from converging on a consistent concept. One solution to this problem maintains multiple G and S sets. In addition to the version space derived from all training instances, it maintains additional spaces based on all but one of the training instances, all but two of the training instances, etc. If G and S fail to converge, the algorithm can examine these alternatives to find those that remain consistent. Unfortunately, this approach leads to a proliferation of candidate sets and is too inefficient to be practical in most cases.

Another issue raised by this research is the role of prior knowledge in learning. LEX's concept hierarchy summarized a great deal of knowledge about algebra; this knowledge was essential to the algorithm's performance. Can greater amounts of domain knowledge make learning even more effective? Section 9.5 examines this problem.

An important contribution of this work is its explication of the relationship between knowledge representation, generalization, and search in inductive learning. Although candidate elimination is only one of many learning algorithms, it raises general questions concerning complexity, expressiveness, and the use of knowledge and data to guide generalization. These problems are central to all machine learning algorithms; we continue to address them throughout this chapter.

9.3 The ID3 Decision Tree Induction Algorithm

ID3 (Quinlan 1986*a*), like candidate elimination, induces concepts from examples. It is particularly interesting for its representation of learned knowledge, its approach to the management of complexity, its heuristic for selecting candidate concepts, and its potential for handling noisy data. ID3 represents concepts as *decision trees*, a representation that

allows us to determine the classification of an object by testing its values for certain properties.

For example, consider the problem of estimating an individual's credit risk on the basis of such properties as credit history, current debt, collateral, and income. Table 9.1 lists a sample of individuals with known credit risks. The decision tree of Figure 9.13 represents the classifications in Table 9.1, in that this tree can correctly classify all the objects in the table. In a decision tree, each internal node represents a test on some property, such as **credit history** or **debt**; each possible value of that property corresponds to a branch of the tree. Leaf nodes represent classifications, such as **low** or **moderate** risk. An individual of unknown type may be classified by traversing this tree: at each internal node, test the individual's value for that property and take the appropriate branch. This continues until reaching a leaf node and the object's classification.

Note that in classifying any given instance, this tree does not use all the properties present in Table 9.1. For instance, if a person has a good credit history and low debt, we may, according to the tree, ignore her collateral and income and classify her as a low risk. In spite of omitting certain tests, this tree correctly classifies all the examples.

NO.	RISK	CREDIT HISTORY	DEBT	COLLATERAL	INCOME
1.	high	bad	high	none	$0 to $15k
2.	high	unknown	high	none	$15 to $35k
3.	moderate	unknown	low	none	$15 to $35k
4.	high	unknown	low	none	$0 to $15k
5.	low	unknown	low	none	over $35k
6.	low	unknown	low	adequate	over $35k
7.	high	bad	low	none	$0 to $15k
8.	moderate	bad	low	adequate	over $35k
9.	low	good	low	none	over $35k
10.	low	good	high	adequate	over $35k
11.	high	good	high	none	$0 to $15k
12.	moderate	good	high	none	$15 to $35k
13.	low	good	high	none	over $35k
14.	high	bad	high	none	$15 to $35k

Table 9.1 Data from credit history of loan applications

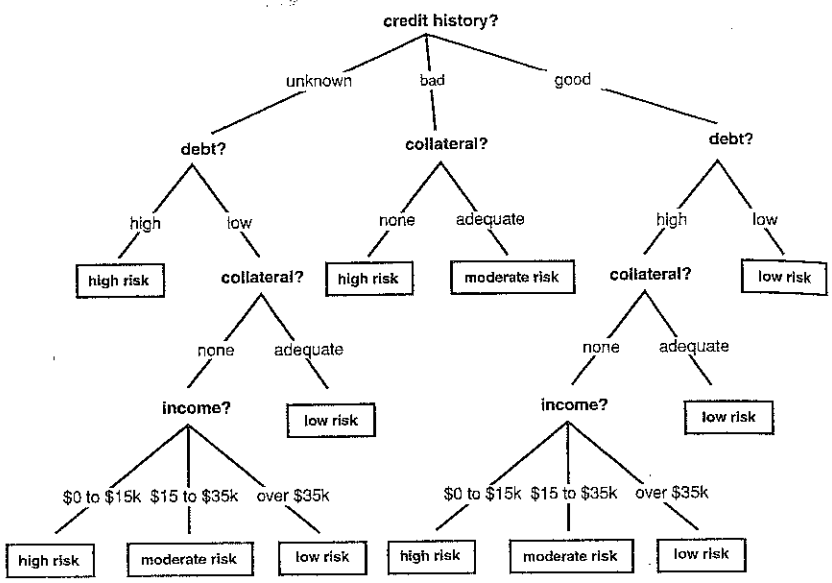


Figure 9.13 A decision tree for credit risk assessment.

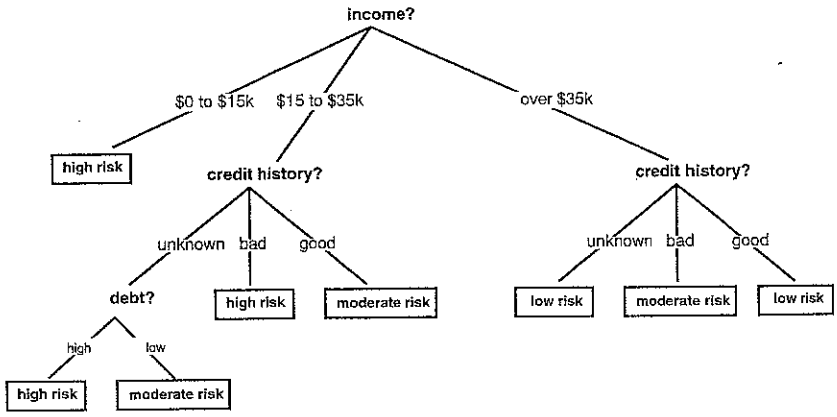
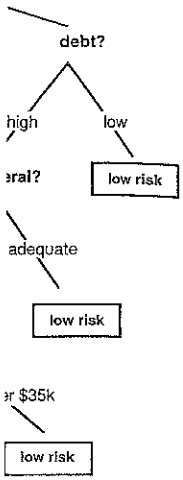
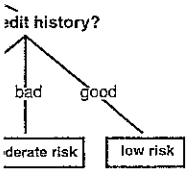


Figure 9.14 A simplified decision tree for credit risk assessment.

In general, the size of the tree necessary to classify a given set of examples varies according to the order with which properties are tested. Figure 9.14 shows a tree that is considerably simpler than that of Figure 9.13 but that also classifies correctly the examples in Table 9.1.





ssment.

set of examples varies
14 shows a tree that is
es correctly the exam-

Given a set of training instances and a number of different decision trees that correctly classify them, we may ask which tree has the greatest likelihood of correctly classifying unseen instances of the population. The ID3 algorithm assumes that this is the simplest decision tree that covers all the training examples. The rationale for this assumption is the time-honored heuristic of preferring simplicity and avoiding unnecessary assumptions. This principle, known as *Occam's Razor*, was first articulated by the medieval logician William of Occam in 1324:

> It is vain to do with more what can be done with less. . . . Entities should not be multiplied beyond necessity.

A more contemporary version of Occam's Razor argues that we should always accept the simplest answer that correctly fits our data. In this case, it is the smallest decision tree that correctly classifies all given examples.

Although Occam's Razor has proven itself as a general heuristic for all manner of intellectual activity, its use here has a more specific justification. If we assume that the given examples are sufficient to construct a valid generalization, then our problem becomes one of distinguishing the necessary properties from the extraneous ones. The simplest decision tree that covers all the examples should be the least likely to include unnecessary constraints. Although this idea is intuitively appealing, it is an assumption that must be empirically tested; Section 9.3.3 presents some of these empirical results. Before examining these results, however, we present the ID3 algorithm for inducing decision trees from examples.

9.3.1 Top-Down Decision Tree Induction

ID3 constructs decision trees in a top-down fashion. Note that for any property, we may partition the set of training examples into disjoint subsets, where all the examples in a partition have a common value for that property. ID3 selects a property to test at the current node of the tree and uses this test to partition the set of examples; the algorithm then recursively constructs a subtree for each partition. This continues until all members of the partition are in the same class; that class becomes a leaf node of the tree. Because the order of tests is critical to constructing a simple decision tree, ID3 relies heavily on its criteria for selecting the test at the root of each subtree. To simplify our discussion, this section describes the algorithm for constructing decision trees, assuming an appropriate test selection function. In Section 9.3.2, we present the selection heuristic of the ID3 algorithm.

For example, consider the way in which ID3 constructs the tree of Figure 9.14 from Table 9.1. Beginning with the full table of examples, ID3 selects **income** as the root property using the selection function described in Section 9.3.2. This partitions the example set as shown in Figure 9.15, with the elements of each partition being listed by their number in the table.

The induction algorithm begins with a sample of correctly classified members of the target categories. ID3 constructs a decision tree according to the algorithm:

```
function induce_tree (example_set, Properties)

begin
if all entries in example_set are in the same class
  then return a leaf node labeled with that class
  else if Properties is empty
    then return leaf node labeled with disjunction of all classes in example_set
    else begin
      select a property, P, and make it the root of the current tree;
      delete P from Properties;
        for each value, V, of P,
          begin
            create a branch of the tree labeled with V;
            let partition_v be elements of example_set with values V for property P;
            call induce_tree(partition_v, Properties), attach result to branch V
          end
    end
end
```

ID3 applies the **induce_tree** function recursively to each partition. For example, the partition {1, 4, 7, 11} consists entirely of high-risk individuals; ID3 creates a leaf node accordingly. ID3 selects the **credit history** property as the root of the subtree for the partition {2, 3, 12, 14}. In Figure 9.16, **credit history** further divides this four element partition into {2, 3}, {14}, and {12}. Continuing to select tests and construct subtrees in this fashion, ID3 eventually produces the tree of Figure 9.14. The reader can work through the remainder of this construction; we present a LISP implementation in Section 15.13.

Before presenting ID3's test selection heuristic, it is worth examining the relationship between the tree construction algorithm and our view of learning as search through a concept space. We may think of the set of all possible decision trees as defining a version space. Our operations for moving through this space consist of adding tests to a tree. ID3 implements a form of greedy search in the space of all possible trees: it adds a subtree to the current tree and continues its search; it does not backtrack. This makes the algorithm highly efficient; it also makes it dependent upon the criteria for selecting properties to test.

9.3.2 Information Theoretic Test Selection

We may think of each property of an instance as contributing a certain amount of information to its classification. For example, if our goal is to determine the species of an animal, the discovery that it lays eggs contributes a certain amount of information to that goal. ID3 measures the information gained by making each property the root of the current subtree. It then picks the property that provides the greatest information gain.

Information theory (Shannon 1948) provides a mathematical basis for measuring the information content of a message. We may think of a message as an instance in a universe of possible messages; the act of transmitting a message is the same as selecting one of these possible messages. From this point of view, it is reasonable to define the information

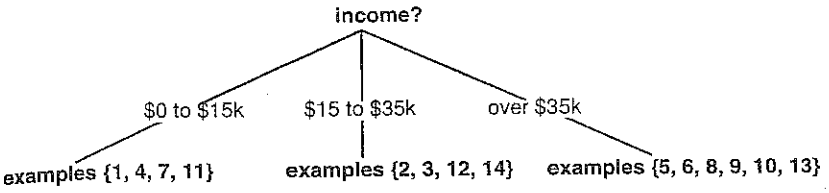


Figure 9.15 A partially constructed decision tree.

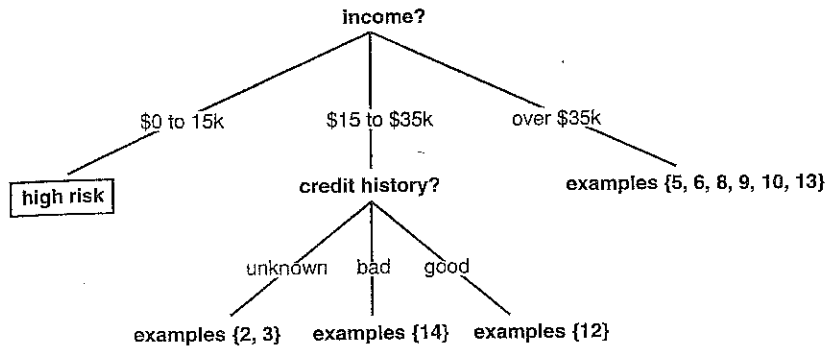


Figure 9.16 Another partially constructed decision tree.

content of a message as depending upon both the size of this universe and the frequency with which each possible message occurs.

The importance of the number of possible messages is evident in an example from gambling: compare a message correctly predicting the outcome of a spin of the roulette wheel with one predicting the outcome of a toss of an honest coin. Because roulette can have more outcomes than a coin toss, a message concerning its outcome is of more value to us: winning at roulette also pays better than winning at a coin toss. Consequently, we should regard this message as conveying more information.

The influence of the probability of each message on the amount of information is evident in another gambling example. Assume that I have rigged a coin so that it will come up heads $^3/_4$ of the time. Because I already know enough about the coin to wager correctly $^3/_4$ of the time, a message telling me the outcome of a given toss is worth less to me than it would be for an honest coin.

Shannon formalized this by defining the amount of information in a message as a function of the probability of occurrence p of each possible message, namely, $-\log_2 p$. Given a universe of messages, $M = \{m_1, m_2, \ldots m_n\}$ and a probability, $p(m_i)$, for the occurrence of each message, the expected information content of a message M is given by:

$$I[M] = \left(\sum_{i=1}^{n} -p(m_i)\ \log_2\ (p(m_i)) \right) = E[-\log_2 p(m_i)]$$

The information in a message is measured in bits. For example, the information content of a message telling the outcome of the flip of an honest coin is:

$$\begin{aligned} I[\text{Coin toss}] &= -p(\text{heads})\log_2(p(\text{heads})) - p(\text{tails})\log_2(p(\text{tails})) \\ &= -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) \\ &= 1 \text{ bit} \end{aligned}$$

However, if the coin has been rigged to come up heads 75 per cent of the time, then the information content of a message is:

$$\begin{aligned} I[\text{Coin toss}] &= -\frac{3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) \\ &= -\frac{3}{4}*(-0.415) - \frac{1}{4}*(-2) \\ &= 0.811 \text{ bits} \end{aligned}$$

This definition formalizes many of our intuitions about the information content of messages. Information theory is widely used in computer science and telecommunications, including such applications as determining the information-carrying capacity of communications channels, developing data compression algorithms, and developing noise resistant communication strategies. ID3 uses information theory to select the test that gives the greatest information gain in classifying the training examples.

We may think of a decision tree as conveying information about the classification of examples in the decision table; the information content of the tree is computed from the probabilities of the different classifications. For example, if we assume that all the examples in Table 9.1 occur with equal probability, then:

$$p(\text{risk is high}) = {}^{6}/_{14},\ p(\text{risk is moderate}) = {}^{3}/_{14},\ p(\text{risk is low}) = {}^{5}/_{14}$$

It follows that the distribution described in Table 9.1, $D_{9.1}$, and, consequently, any tree that covers those examples, is:

$$\begin{aligned} I[D_{9.1}] &= -\frac{6}{14}\log_2\left(\frac{6}{14}\right) - \frac{3}{14}\log_2\left(\frac{3}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) \\ &= -\frac{6}{14}*(-1.222) - \frac{3}{14}*(-2.222) - \frac{5}{14}*(-1.485) \\ &= 1.531 \text{ bits} \end{aligned}$$

The information gain provided by making a test at the root of the current tree is equal to the total information in the tree minus the amount of information needed to complete the classification after performing the test. The amount of information needed to complete the tree is defined as the weighted average of the information in all its subtrees. We compute the weighted average by multiplying the information content of each subtree by the percentage of the examples present in that subtree and summing these products.

Assume a set of training instances, C. If we make property P, with n values, the root of the current tree, this will partition C into subsets, $\{C_1, C_2, \ldots C_n\}$. The expected information needed to complete the tree after making P the root is:

$$E[P] = \sum_{i=1}^{n} \frac{|C_i|}{|C|} I[C_i]$$

The gain from property P is computed by subtracting the expected information to complete the tree from the total information content of the tree:

$$\text{gain}(P) = I[C] - E[P]$$

In the example of Table 9.1, if we make **income** the property tested at the root of the tree, this partitions the table of examples into the partitions $C_1 = \{1,4,7,11\}$, $C_2 = \{2,3,12,14\}$, and $C_3 = \{5,6,8,9,10,13\}$. The expected information needed to complete the tree is:

$$E[\text{income}] = \frac{4}{14} * I[C_1] + \frac{4}{14} * I[C_2] + \frac{6}{14} * I[C_3]$$

$$= \frac{4}{14} * 0.0 + \frac{4}{14} * 1.0 + \frac{6}{14} * 0.650$$

$$= 0.564 \text{ bits}$$

The information gain for the distribution of Table 9.1 is:

$$\text{gain(income)} = I[D_{9.1}] - E[\text{income}]$$
$$= 1.531 - 0.564$$
$$= 0.967 \text{ bits}$$

Similarly, we may show that

$$\text{gain(credit history)} = 0.266$$
$$\text{gain(debt)} = 0.581$$
$$\text{gain (collateral)} = 0.756$$

Because income provides the greatest information gain, ID3 will select it as the root of the tree. The algorithm continues to apply this analysis recursively to each subtree until it has completed the tree.

9.3.3 Evaluating ID3

Although the ID3 algorithm produces simple decision trees, it is not obvious that such trees will be effective in predicting the classification of unknown examples. ID3 has been evaluated in both controlled tests and applications and has proven to work well in practice.

Quinlan, for example, has evaluated ID3's performance on the problem of learning to classify boards in a chess endgame (Quinlan 1983). The endgame involved white, playing with a king and a rook, against black, playing with a king and a knight. ID3's goal was to learn to recognize boards that led to a loss for black within three moves. The attributes were different high-level properties of boards, such as "an inability to move the king safely." The test used 23 such attributes.

Once board symmetries were taken into account, the entire problem domain consisted of 1.4 million different boards, of which 474,000 were a loss for black in three moves. ID3 was tested by giving it a randomly selected training set and then testing it on 10,000 different boards, also randomly selected. Quinlan's tests gave the results found in Table 9.2. The predicted maximum errors were derived from a statistical model of ID3's behavior in the domain. For further analysis and details see Quinlan (1983).

These results are supported by further empirical studies and by anecdotal results from further applications. Variations of ID3 have been developed to deal with such problems as noise and excessively large training sets. For more details, see Quinlan (1986*a*, *b*).

Size of Training Set	Percentage of Whole Universe	Errors in 10,000 Trials	Predicted Maximum Errors
200	0.01	199	728
1,000	0.07	33	146
5,000	0.36	8	29
25,000	1.79	6	7
125,000	8.93	2	1

Table 9.2 The evaluation of ID3

9.3.4 Decision Tree Data Issues: Bagging, Boosting

Quinlan (1983) was the first to suggest the use of information theory to produce subtrees in decision tree learning and his work was the basis for our presentation. Our examples were clean, however, and their use straightforward. There are a number of issues that we did not address, each of which often occurs in a large data set:

1. The data is bad. This can happen when two (or more) identical attribute sets give different results. What can we do if we have no a priori reason to get rid of data?

2. Data from some attribute sets is missing, perhaps because it is too expensive to obtain. Do we extrapolate? Can we create a new value "unknown?" How can we smooth over this irregularity?

3. Some of the attribute sets are continuous. We handled this by breaking the continuous value "income" into convenient subsets of values, and then used these groupings. Are there better approaches?

4. The data set may be too large for the learning algorithm. How do you handle this?

Addressing these issues produced new generations of decision tree learning algorithms after ID3. The most notable of these is C4.5 (Quinlan 1996). These issues also led to techniques such as bagging and boosting. Since the data for classifier learning systems are attribute-value vectors or instances, it is tempting to manipulate the data to see if different classifiers are produced.

Bagging produces replicate training sets by sampling with replacement from the training instances. *Boosting* uses all instances at each replication, but maintains a weight for each instance in the training set. This weight is intended to reflect that vector's importance. When the weights are adjusted, different classifiers are produced, since the weights cause the learner to focus on different instances. In either case, the multiple classifiers produced are combined by voting to form a composite classifier. In bagging, each component classifier has the same vote, while boosting assigns different voting strengths to component classifiers on the basis of their accuracy.

In working with very large sets of data, it is common to divide the data into subsets, build the decision tree on one subset, and then test its accuracy on other subsets. The literature on decision tree learning is now quite extensive, with a number of data sets on-line, and a number of empirical results published showing the results of using various versions of decision tree algorithms on this data.

Finally, it is straightforward to convert a decision tree into a comparable rule set. What we do is make each possible path through the decision tree into a single rule. The pattern for the rule, its left-hand side (Chapter 5), consists of the decisions leading to the leaf node. The action or right-hand side is the leaf node or outcome of the tree. This rule set may be further customized to capture subtrees within the decision tree. This rule set may then be run to classify new data.

9.4 Inductive Bias and Learnability

So far, our discussion has emphasized the use of empirical data to guide generalization. However, successful induction also depends upon prior knowledge and assumptions about the nature of the concepts being learned. *Inductive bias* refers to any criteria a learner uses to constrain the concept space or to select concepts within that space. In the next section, we examine the need for bias and the types of biases that learning programs typically