Module-2

DECISION TREE LEARNING

INTRODUCTION

Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree. Learned trees can also be re-represented as sets of if-then rules to improve human readability. These learning methods are among the most popular of inductive inference algorithms and have been successfully applied to a broad range of tasks from learning to diagnose medical cases to learning to assess credit risk of loan applicants.

DECISION TREE REPRESENTATION

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute. An

instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.

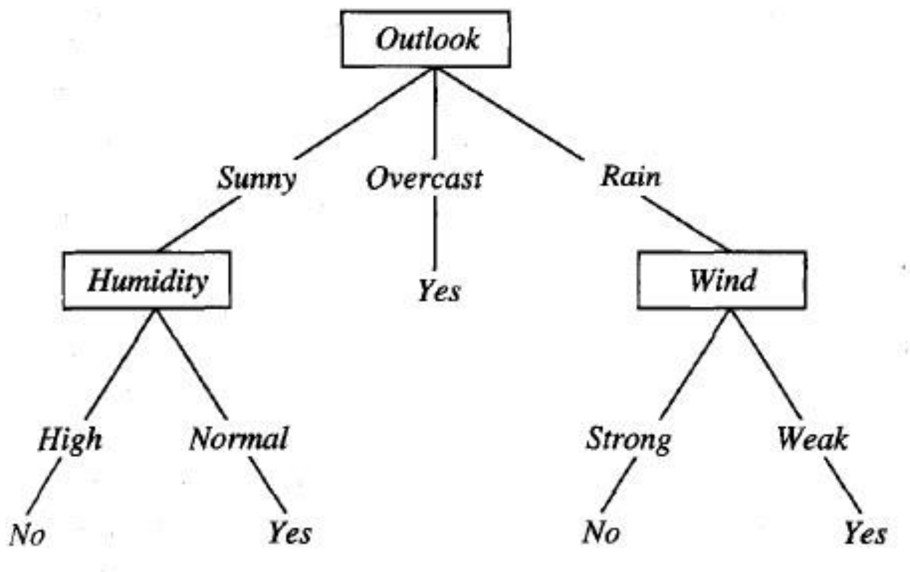


FIGURE 3.1
A decision tree for the concept *PlayTennis*. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf (in this case, *Yes* or *No*). This tree classifies Saturday mornings according to whether or not they are suitable for playing tennis.

Figure ***3.1*** illustrates a typical learned decision tree. This decision tree classifies Saturday mornings according to whether they are suitable for playing tennis. For example, the instance

(Outlook = ***Sunny, Temperature*** = ***Hot, Humidity*** = ***High, Wind*** = ***Strong)***

would be sorted down the leftmost branch of this decision tree and would therefore be classified as a negative instance (i.e., the tree predicts that ***PlayTennis*** = ***no).*** This tree and the example used in Table to illustrate the ID3 learning algorithm are adapted from (Quinlan ***1986).*** In general, decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances. Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions. For example, the decision tree shown in Figure 3.1 corresponds to the expression

(Outlook = ***Sunny*** **A** ***Humidity*** = ***Normal)***
V ***(Outlook*** = ***Overcast)***
v ***(Outlook*** = ***Rain A Wind*** = ***Weak)***

APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING

Although a variety of decision tree learning methods have been developed with somewhat differing capabilities and requirements, decision tree learning is generally best suited to problems with the following characteristics:

- ***Instances are represented by attribute-value pairs.*** Instances are described by a fixed set of attributes (e.g., ***Temperature)*** and their values (e.g., ***Hot).*** The easiest situation for decision tree learning is when each attribute takes on small number of disjoint possible values (e.g., ***Hot, Mild, Cold).*** However, extensions to the basic algorithm (discussed in Section 3.7.2) allow handling real-valued attributes as well (e.g., representing ***Temperature*** numerically).
- ***The targetfunction has discrete output values.*** The decision tree in Figure 3.1assigns a boolean classification (e.g., ***yes*** or ***no)*** to each example. Decision tree methods easily extend to learning functions with more than two possible output values. A more substantial extension allows learning target functions with real-valued outputs, though the application of decision trees in this setting is less common.
- ***Disjunctive descriptions may be required.*** As noted above, decision trees naturally represent disjunctive expressions.
- ***The training data may contain errors.*** Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples.

- ***The training data may contain missing attribute values.*** Decision tree methods can be used even when some training examples have unknown values (e.g., if the ***Humidity*** of the day is known for only some of the training

  examples). This issue is discussed in Section 3.7.4.

Many practical problems have been found to fit these characteristics. Decision tree learning has therefore been applied to problems such as learning to classify medical patients by their disease, equipment malfunctions by their cause, and loan applicants by their likelihood of defaulting on payments. Such problems, in which the task is to classify examples into one of a discrete set of possible categories, are often referred to as ***classification problems.***

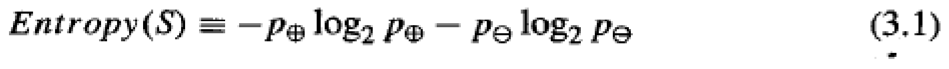
THE BASIC DECISION TREE LEARNING ALGORITHM

Most algorithms that have been developed for learning decision trees are variations on a core algorithm that employs a top-down, greedy search through the space of possible decision trees. This approach is exemplified by the ID3 algorithm

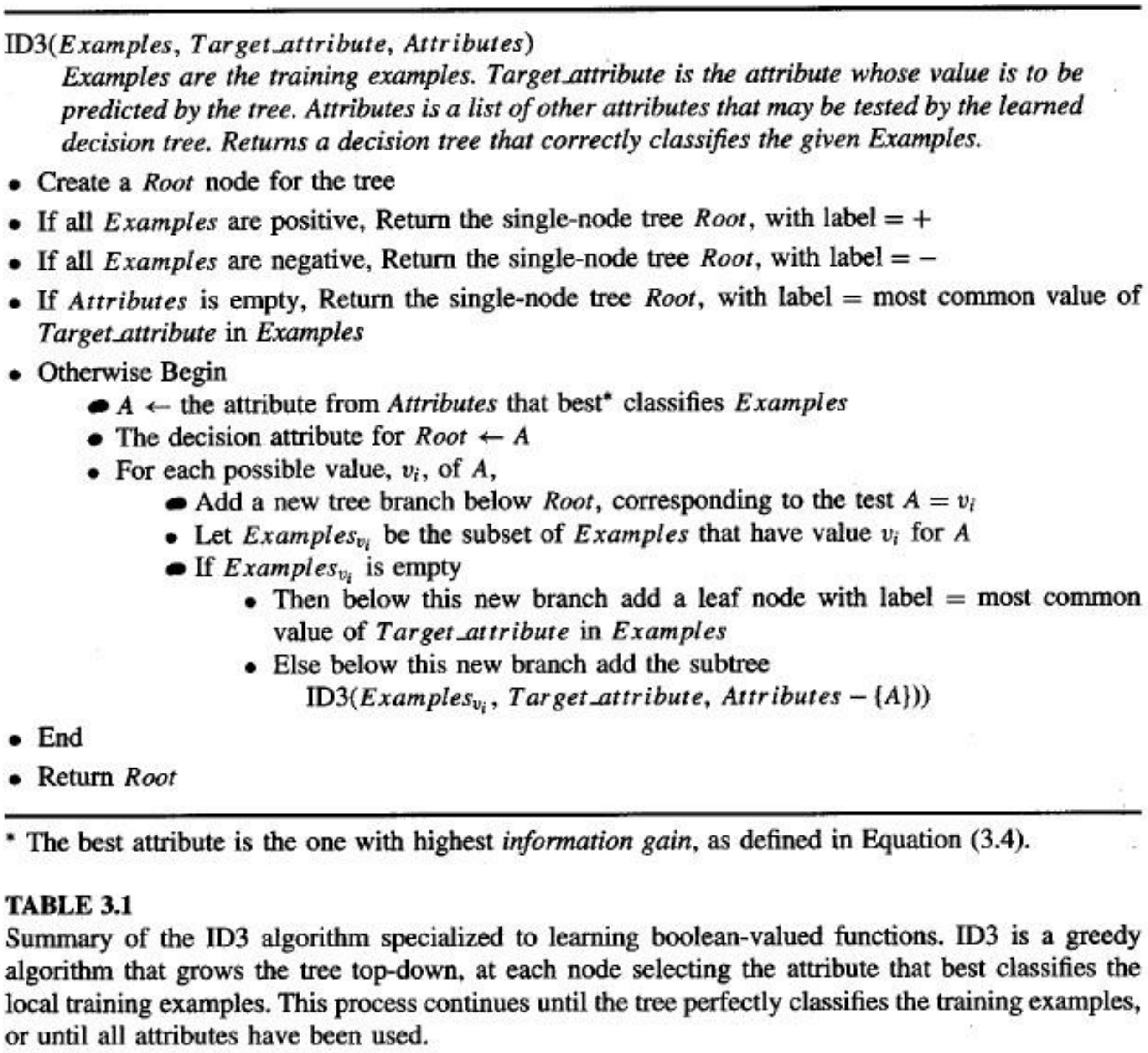
(Quinlan 1986) and its successor C4.5 (Quinlan 1993), which form the primary focus of our discussion here. In this section we present the basic algorithm for decision tree learning, corresponding approximately to the ID3 algorithm. In Section 3.7 we consider a number of extensions to this basic algorithm, including extensions incorporated into C4.5 and other more recent algorithms for decision tree learning. Our basic algorithm, ID3, learns decision trees by constructing them topdown, beginning with the question "which attribute should be tested at the root of the tree?'To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected and used as the test at the root node of the tree. A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this attribute). The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.This forms a greedy search for an acceptable decision tree, in which the algorithm never backtracks to reconsider earlier choices. **A** simplified version of the algorithm, specialized to learning boolean-valued functions (i.e., concept learning), is described in Table 3.1.

Which Attribute Is the Best Classifier?

The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree. We would like to select the attribute that is most useful for classifying examples. What is a good quantitative measure of the worth of an attribute? We will define a statistical property, called ***information gain,*** that measures how well a given attribute separates the training examples according to their target classification. ID3 uses this information gain measure to select among the candidate attributes at each step while growing the tree.



$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus} \qquad (3.1)$$



```
ID3(Examples, Target_attribute, Attributes)
    Examples are the training examples. Target_attribute is the attribute whose value is to be
    predicted by the tree. Attributes is a list of other attributes that may be tested by the learned
    decision tree. Returns a decision tree that correctly classifies the given Examples.
• Create a Root node for the tree
• If all Examples are positive, Return the single-node tree Root, with label = +
• If all Examples are negative, Return the single-node tree Root, with label = −
• If Attributes is empty, Return the single-node tree Root, with label = most common value of
  Target_attribute in Examples
• Otherwise Begin
      • $A \leftarrow$ the attribute from Attributes that best* classifies Examples
      • The decision attribute for Root $\leftarrow A$
      • For each possible value, $v_i$, of $A$,
            • Add a new tree branch below Root, corresponding to the test $A = v_i$
            • Let $Examples_{v_i}$ be the subset of Examples that have value $v_i$ for $A$
            • If $Examples_{v_i}$ is empty
                  • Then below this new branch add a leaf node with label = most common
                    value of Target_attribute in Examples
                  • Else below this new branch add the subtree
                        ID3($Examples_{v_i}$, Target_attribute, Attributes − {A}))
• End
• Return Root
```

* The best attribute is the one with highest *information gain*, as defined in Equation (3.4).

TABLE 3.1
Summary of the ID3 algorithm specialized to learning boolean-valued functions. ID3 is a greedy algorithm that grows the tree top-down, at each node selecting the attribute that best classifies the local training examples. This process continues until the tree perfectly classifies the training examples, or until all attributes have been used.

ENTROPY MEASURES HOMOGENEITY OF EXAMPLES

In order to define information gain precisely, we begin by defining a measure commonly used in information theory, called ***entropy.***

Definition of entropy:

It characterizes the (im)purity of an arbitrary collection of examples. Given a collection S(samples), containing positive and negative examples of some target concept, the entropy of S relative to this boolean classification is where p+, is the proportion of positive examples in S and p-, is the proportion of negative examples in S.

To illustrate, suppose S is a collection of 14 examples of some Boolean concept, including 9 positive and 5 negative examples (we adopt the notation [9+, 5-] to summarize such a sample of data). Then the entropy of S relative to this boolean classification is Notice that the entropy is 0 if all members of S belong to the same class. For example, if all members are positive (pe = I), then p, is 0, and Entropy(S) = -1 . $\log_2(1)$ - 0 . $\log_2 0$ = -1 . 0 - 0 . $\log_2 0$ = 0. Note the entropy is 1 when the collection contains an equal number of positive and negative examples. If the collection contains unequal numbers of positive and negative examples, the entropy is between **0** and 1. Figure 3.2 shows the form of the entropy function relative to a boolean classification, as p+**,** varies between **0** and 1.

One interpretation of entropy from information theory is that it specifies the minimum number of bits of information needed to encode the classification of an arbitrary member of ***S*** (i.e., a member of ***S*** drawn at random with uniform probability). For example, if ***p,*** is 1, the receiver knows the drawn example will be positive, so no message need be sent, and the entropy is zero. On the other hand, if ***p+*** is 0.5, one bit is required to indicate whether the drawn example is positive or negative. If ***p+*** is 0.8, then a collection of messages can be encoded using on average less than 1 bit per message by assigning shorter codes to collections of positive examples and longer codes to less likely negative

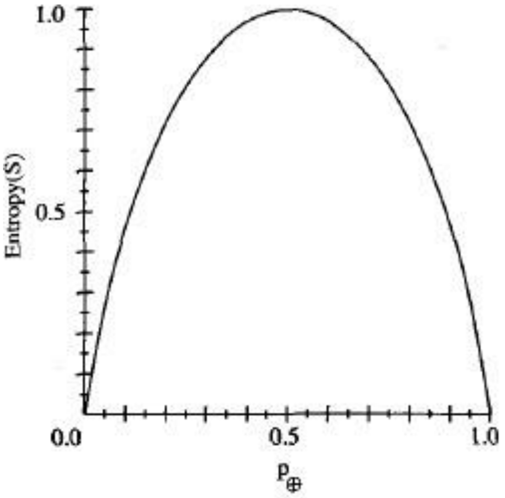


FIGURE 3.2
The entropy function relative to a boolean classification, as the proportion, $p_\oplus$, of positive examples varies between 0 and 1.

examples. Thus far we have discussed entropy in the special case where the target classification is boolean. More generally, if the target attribute can take on c different values, then the entropy of ***S*** relative to this c-wise classification is defined as

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i \qquad (3.3)$$

where ***pi*** is the proportion of ***S*** belonging to class ***i.*** Note the logarithm is still base 2 because entropy is a measure of the expected encoding length measured in ***bits.*** Note also that if the target attribute can take on c possible values, the entropy can be as large as log, ***c.***

INFORMATION GAIN MEASURES THE EXPECTED REDUCTION IN ENTROPY

Given entropy as a measure of the impurity in a collection of training examples, we can now define a measure of the effectiveness of an attribute in classifying the training data. The measure we will use, called ***information gain,*** is simply the expected reduction in entropy caused by partitioning the examples according to this attribute. More precisely, the information gain, ***Gain(S, A)*** of ***an*** attribute **A,** relative to a collection of examples ***S,*** is defined as

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \qquad (3.4)$$

where ***Values(A)*** is the set of all possible values for attribute A, and ***S,*** is the subset of S for which attribute A has value v (i.e., $S_v = \{s \in S | A(s) = v\}$)..

Note the first term in Equation ***(3.4)*** is just the entropy of the original collection ***S,*** and the second term is the expected value of the entropy after ***S*** is partitioned using attribute ***A.*** The expected entropy described by this second term is simply the sum of the entropies of each subset ***S,,*** weighted by the fraction of examples that belong to ***S,. Gain(S, A)*** is therefore the expected reduction in entropy caused by knowing the value of attribute A. Put another way, ***Gain(S, A)*** is the information provided about the ***target &action value,*** given the value of some other attribute ***A.*** The value of ***Gain(S, A)*** is the number of bits saved when encoding the target value of an arbitrary member of ***S,*** by knowing the value of attribute ***A.***

For example, suppose ***S*** is a collection of training-example days described by attributes including ***Wind,*** which can have the values ***Weak*** or ***Strong.*** As before, assume ***S*** is a collection containing ***14*** examples, [9+, 5-]. Of these 14 examples, suppose 6 of the positive and 2 of the negative examples have ***Wind*** = ***Weak,*** and the remainder have ***Wind*** = ***Strong.*** The information gain due to sorting the original ***14*** examples by the attribute ***Wind*** may then be calculated as

$$\begin{aligned}
Values(Wind) &= Weak, Strong \\
S &= [9+, 5-] \\
S_{Weak} &\leftarrow [6+, 2-] \\
S_{Strong} &\leftarrow [3+, 3-] \\
Gain(S, Wind) &= Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v) \\
&= Entropy(S) - (8/14) Entropy(S_{Weak}) \\
&\quad - (6/14) Entropy(S_{Strong}) \\
&= 0.940 - (8/14)0.811 - (6/14)1.00 \\
&= 0.048
\end{aligned}$$

Information gain is precisely the measure used by ID3 to select the best attribute at each step in growing the tree. The use of information gain to evaluate the relevance of attributes is summarized in Figure ***3.3.*** In this figure the information gain of two different attributes, ***Humidity*** and ***Wind,*** is computed in order to determine which is the better attribute for classifying the training examples shown in Table ***3.2.***

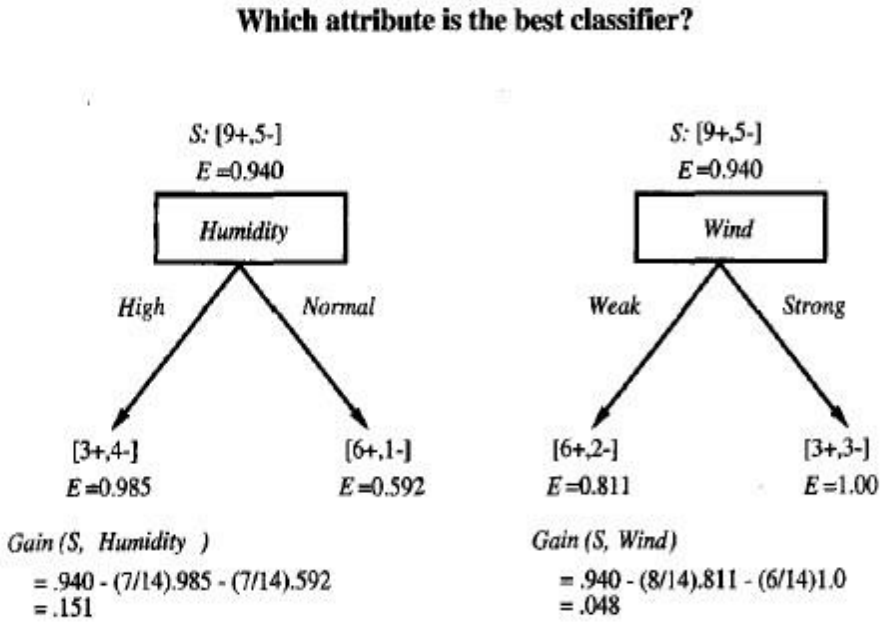


FIGURE 3.3

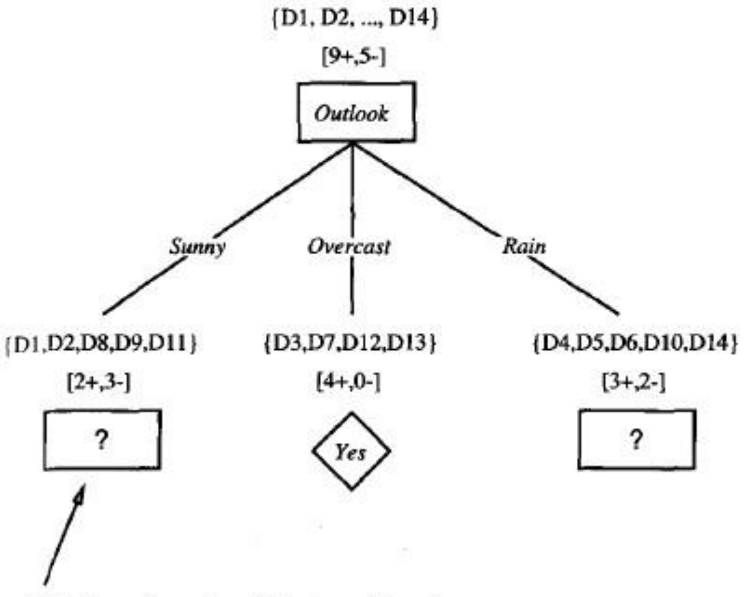
Humidity provides greater information gain than *Wind*, relative to the target classification. Here, E stands for entropy and S for the original collection of examples. Given an initial collection S of 9 positive and 5 negative examples, [9+, 5−], sorting these by their *Humidity* produces collections of [3+, 4−] ($Humidity = High$) and [6+, 1−] ($Humidity = Normal$). The information gained by this partitioning is .151, compared to a gain of only .048 for the attribute *Wind*.

An Illustrative Example

To illustrate the operation of ID3, consider the learning task represented by the training examples of Table 3.2. Here the target attribute ***PlayTennis,*** which can have values ***yes*** or ***no*** for different Saturday mornings, is to be predicted based on other attributes of the morning in question. Consider the first step through the algorithm, in which the topmost node of the decision tree is created. Which attribute should be tested first in the tree? ID3 determines the information gain for each candidate attribute (i.e., ***Outlook, Temperature, Humidity,*** and ***Wind),*** then gain for two of these attributes is shown in Figure 3.3.

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

TABLE 3.2
Training examples for the target concept *PlayTennis*.



Which attribute should be tested here?

S_{sunny} = {D1,D2,D8,D9,D11}

$Gain\,(S_{sunny}, Humidity)$ = .970 - (3/5) 0.0 - (2/5) 0.0 = .970

$Gain\,(S_{sunny}, Temperature)$ = .970 - (2/5) 0.0 - (2/5) 1.0 - (1/5) 0.0 = .570

$Gain\,(S_{sunny}, Wind)$ = .970 - (2/5) 1.0 - (3/5) .918 = .019

FIGURE 3.4
The partially learned decision tree resulting from the first step of ID3. The training examples are sorted to the corresponding descendant nodes. The *Overcast* descendant has only positive examples and therefore becomes a leaf node with classification *Yes*. The other two nodes will be further expanded, by selecting the attribute with highest information gain relative to the new subsets of examples.

The information gain values for all four attributes are

Gain(S, Outlook) = 0.246

Gain(S, Humidity) = 0.151

Gain(S, Wind) = 0.048

Gain(S, Temperature) = 0.029

where ***S*** denotes the collection of training examples from Table 3.2. According to the information gain measure, the ***Outlook*** attribute provides the best prediction of the target attribute, ***PlayTennis,*** over the training examples. Therefore, ***Outlook*** is selected as the decision attribute for the root node, and branches are created below the root for each of its possible values (i.e.***Sunny, Overcast,*** and ***Rain).*** The resulting partial decision tree is shown in Figure 3.4, along with the training examples sorted to each new descendant node. Note that every example for which ***Outlook*** = ***Overcast*** is also a positive example of ***PlayTennis.*** Therefore, this node of the tree becomes a leaf node with the classification ***PlayTennis*** = ***Yes.*** In contrast, the descendants corresponding to ***Outlook*** = ***Sunny*** and ***Outlook*** = ***Rain*** still have nonzero entropy, and the decision tree will be further elaborated below these nodes. The process of selecting a new attribute and partitioning the training examples is now repeated for each nontenninal descendant node, this time using only the training examples associated with that node. Attributes that have been incorporated higher in the tree are excluded, so that any given attribute can appear at most once along any path through the tree. This process continues for each new leaf node until either of two conditions is met: (1) every attribute has already been included along this path through the tree, or (2) the training examples associated with this leaf node all have the same target attribute value (i.e., their entropy is zero). Figure 3.4 illustrates the computations of information gain for the next step in growing the decision tree. The final decision tree learned by ID3 from the 14 training examples of Table 3.2 is shown in Figure 3.1.

HYPOTHESIS SPACE SEARCH IN DECISION TREE LEARNING

As with other inductive learning methods, ID3 can be characterized as searching a space of hypotheses for one that fits the training examples. The hypothesis space searched by ID3 is the set of possible decision trees. ID3 performs a simple-to complex, hill-climbing search through this hypothesis space, beginning with the empty tree, then considering progressively more elaborate hypotheses in search of a decision tree that correctly classifies the training data. The evaluation function that guides this hill-climbing search is the information gain measure. This search is depicted in Figure 3.5.

By viewing ID3 in terms of its search space and search strategy, we can get some insight into its capabilities and limitations.

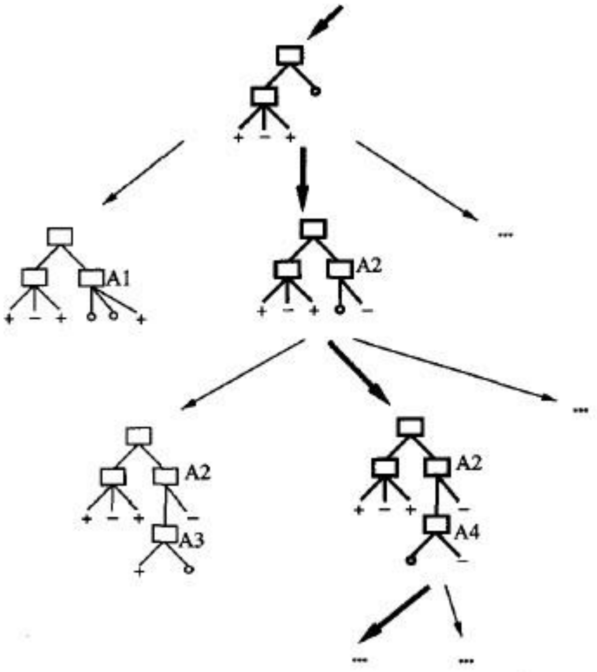


FIGURE 3.5
Hypothesis space search by ID3. ID3 searches through the space of possible decision trees from simplest to increasingly complex, guided by the information gain heuristic.

- **ID3's** hypothesis space of all decision trees is a ***complete*** space of finite discrete-valued functions, relative to the available attributes. Because every finite discrete-valued function can be represented by some decision tree, ID3
  avoids one of the major risks of methods that search incomplete hypothesis spaces (such as methods that consider only conjunctive hypotheses): that the hypothesis space might not contain the target function.
- **ID3** maintains only a single current hypothesis as it searches through the space of decision trees. This contrasts, for example, with the earlier version space candidate elimination method, which maintains the set of ***all*** hypotheses consistent with the available training examples. By determining only a single hypothesis, ID^ loses the capabilities that follow from explicitly representing all consistent hypotheses. For example, it does not have the ability to determine how many alternative decision trees are consistent with the available training data, or to pose new instance queries that optimally resolve among these competing hypotheses.
- **ID3** in its pure form performs no backtracking in its search. Once it, selects an attribute to test at a particular level in the tree, it never backtracks to reconsider this choice. Therefore, it is susceptible to the usual risks of hill-climbing search without backtracking: converging to locally optimal solutions that are not globally optimal. In the case of **ID3,** a locally optimal solution corresponds to the decision tree it selects along the single search path it explores. However, this locally optimal solution may be less desirable than trees that would have been encountered along a different branch of the search. Below we discuss an extension that adds a form of backtracking (post-pruning the decision tree).

- **ID3** uses all training examples at each step in the search to make statistically based decisions regarding how to refine its current hypothesis. This contrasts with methods that make decisions incrementally, based on individual training examples (e.g., FIND-S or CANDIDATE-ELIMINATOION).One advantage of using statistical properties of all the examples (e.g., information gain) is that the resulting search is much less sensitive to errors in individual training examples. **ID3** can be easily extended to handle noisy training data by modifying its termination criterion to accept hypotheses that imperfectly fit the training data.

INDUCTIVE BIAS IN DECISION TREE LEARNING

What is the policy by which ID3 generalizes from observed training examples to classify unseen instances? In other words, what is its inductive bias? Recall from Chapter 2 that inductive bias is the set of assumptions that, together with the training data, deductively justify the classifications assigned by the learner to future instances.

Given a collection of training examples, there are typically many decision trees consistent with these examples. Describing the inductive bias of ID3 therefore consists of describing the basis by which it chooses one of these consistent

hypotheses over the others. Which of these decision trees does ID3 choose? It chooses the first acceptable tree it encounters in its simple-to-complex, hill-climbing search through the space of possible trees. Roughly speaking, then, the ID3 search strategy (a) selects in favour of shorter trees over longer ones, and (b) selects trees that place the attributes with highest information gain closest to the root. Because of the subtle interaction between the attribute selection heuristic used by ID3 and the particular training examples it encounters, it is difficult to characterize precisely the inductive bias exhibited by ID3. However, we can approximately characterize its bias as a preference for short decision trees over complex trees.

Approximate inductive bias of ID3: Shorter trees are preferred over larger trees.

In fact, one could imagine an algorithm similar to ID3 that exhibits precisely this inductive bias. Consider an algorithm that begins with the empty tree and searches ***breadth First*** through progressively more complex trees, first considering

all trees of depth 1, then all trees of depth 2, etc. Once it finds a decision tree consistent with the training data, it returns the smallest consistent tree at that search depth (e.g., the tree with the fewest nodes). Let us call this breadth-first search algorithm BFS-ID3. BFS-ID3 finds a shortest decision tree and thus exhibits precisely the bias "shorter trees are preferred over longer trees." ID3 can be viewed as an efficient approximation to BFS-ID3, using a greedy heuristic search to attempt to find the shortest tree without conducting the entire breadth-first search through the hypothesis space.

Because ID3 uses the information gain heuristic and a hill climbing strategy, it exhibits a more complex bias than BFS-ID3. In particular, it does not always find the shortest consistent tree, and it is biased to favor trees that place attributes with high information gain closest to the root.

> **A closer approximation to the inductive bias of ID3:** Shorter trees are preferred over longer trees. Trees that place high information gain attributes close to the root are preferred over those that do not.

Restriction Biases and Preference Biases

There is an interesting difference between the types of inductive bias exhibited by ID3 and by the CANDIDATE-ELIMINATION algorithm discussed in Chapter 2.

Consider the difference between the hypothesis space search in these two approaches:

- ID3 searches a complete hypothesis space (i.e., one capable of expressing any finite discrete-valued function). It searches incompletely through this space, from simple to complex hypotheses, until its termination condition is met (e.g., until it finds a hypothesis consistent with the data). Its inductive bias is solely a consequence of the ordering of hypotheses by its search strategy. Its hypothesis space introduces no additional bias.
- The version space CANDIDATE-ELIMINATION algorithm searches an incomplete hypothesis space (i.e., one that can express only a subset of the potentially teachable concepts). It searches this space completely, finding every hypothesis consistent with the training data. Its inductive bias is solely a consequence of the expressive power of its hypothesis representation. Its search strategy introduces no additional bias.

In brief, the inductive bias of ID3 follows from its search strategy, whereas the inductive bias of the CANDIDATE-ELIMINATION algorithm follows from the definition of its search space.

The inductive bias of ID3 is thus a preference for certain hypotheses over others (e.g., for shorter hypotheses), with no hard restriction on the hypotheses that can be eventually enumerated. This form of bias is typically called a preference bias (or, alternatively, a search bias). In contrast, the bias of the CANDIDATEELIMINATION algorithm is in the form of a categorical restriction on the set of hypotheses considered. This form of bias is typically called a restriction bias (or, alternatively, a language bias).

Given that some form of inductive bias is required in order to generalize beyond the training data (see Chapter 2), which type of inductive bias shall we prefer; a preference bias or restriction bias?

Typically, a preference bias is more desirable than a restriction bias, because it allows the learner to work within a complete hypothesis space that is assured to contain the unknown target function. In contrast, a restriction bias that strictly limits the set of potential hypotheses is generally less desirable, because it introduces the possibility of excluding the unknown target function altogether.

Whereas ID3 exhibits a purely preference bias and CANDIDATE-ELIMINATION a purely restriction bias, some learning systems combine both.

Why Prefer Short Hypotheses?

Is ID3's inductive bias favouring shorter decision trees a sound basis for generalizing beyond the training data? Philosophers and others have debated this question for centuries, and the debate remains unresolved to this day. William of Occam was one of the first to discuss the question, around the year 1320, so this bias often goes by the name of Occam's razor.

Occam's razor: Prefer the simplest hypothesis that fits the data.

Of course giving an inductive bias a name does not justify it. Why should one prefer simpler hypotheses? Notice that scientists sometimes appear to follow this inductive bias. Physicists, for example, prefer simple explanations for the motions of the planets, over more complex explanations. Why? One argument is that because there are fewer short hypotheses than long ones (based on straightforward combinatorial arguments), it is less likely that one will find a short hypothesis that coincidentally fits the training data. In contrast there are often many very complex hypotheses that fit the current training data but fail to generalize correctly to subsequent data. Consider decision tree hypotheses, for example. There are many more 500-node decision trees than 5-node decision trees. Given a small set of 20 training examples, we might expect to be able to find many 500-node decision trees consistent with these, whereas we would be more surprised if a 5-node decision tree could perfectly fit this data. We might therefore believe the 5-node tree is less likely to be a statistical coincidence and prefer this hypothesis over the 500-node hypothesis.

Upon closer examination, it turns out there is a major difficulty with the above argument. By the same reasoning we could have argued that one should prefer decision trees containing exactly 17 leaf nodes with 11 nonleaf nodes, that use the decision attribute $\mathbf{A}_1$ at the root, and test attributes $\mathbf{A}_2$ through $\mathbf{A}_{ll}$**,** in numerical order. There are relatively few such trees, and we might argue (by the same reasoning as above) that our a priori chance of finding one consistent with an arbitrary set of data is therefore small. The difficulty here is that there are very many small sets of hypotheses that one can define-most of them rather arcane. Why should we believe that the small set of hypotheses consisting of decision trees with ***short descriptions*** should be any more relevant than the multitude of other small sets of hypotheses that we might define?

A second problem with the above argument for Occam's razor is that the size of a hypothesis is determined by the particular representation used ***internally*** by the learner. Two learners using different internal representations could therefore arrive at different hypotheses, both justifying their contradictory conclusions by Occam's razor! For example, the function represented by the learned decision tree in Figure 3.1 could be represented as a tree with just one decision node, by a learner that uses the boolean attribute XYZ, where we define the attribute XYZ to be true for instances that are classified positive by the decision tree in Figure 3.1 and false otherwise. Thus, two learners, both applying Occam's razor, would generalize in different ways if one used the XYZ attribute to describe its examples and the other used only the attributes ***Outlook, Temperature, Humidity,*** and ***Wind.***

This last argument shows that Occam's razor will produce two different hypotheses from the same training examples when it is applied by two learners basis we might be tempted to reject Occam's razor altogether. However, consider the following scenario that examines the question of which internal representations might arise from a process of evolution and natural selection. Imagine a population of artificial learning agents created by a simulated evolutionary process involving reproduction, mutation, and natural selection of these agents. Let us assume that this evolutionary process can alter the perceptual systems of these agents from generation to generation, thereby changing the internal attributes by which they perceive their world. For the sake of argument, let us also assume that the learning agents employ a fixed learning algorithm (say ID3) that cannot be altered by evolution. It is reasonable to assume that over time evolution will produce internal representation that make these agents increasingly successful within their environment. Assuming that the success of an agent depends highly on its ability to generalize accurately, we would therefore expect evolution to develop internal representations that work well with whatever learning algorithm and inductive bias is present. If the species of agents employs a learning algorithm whose inductive bias is Occam's razor, then we expect evolution to produce internal representations for which Occam's razor is a successful strategy. The essence of the
argument here is that evolution will create internal representations that make the learning algorithm's inductive bias a self-fulfilling prophecy, simply because it can alter the representation easier than it can alter the learning algorithm.

ISSUES IN DECISION TREE LEARNING

Practical issues in learning decision trees include determining how deeply to grow the decision tree, handling continuous attributes, choosing an appropriate attribute selection measure, handling training data with missing attribute values, handling attributes with differing costs, and improving computational efficiency. Below we discuss each of these issues and extensions to the basic ID3 algorithm that address them. ID3 has itself been extended to address most of these issues, with the resulting system renamed C4.5 (Quinlan 1993).

Avoiding Overfitting the Data

The algorithm described in Table 3.1 grows each branch of the tree just deeply enough to perfectly classify the training examples. While this is sometimes a reasonable strategy, in fact it can lead to difficulties when there is noise in the data,

or when the number of training examples is too small to produce a representative sample of the true target function. In either of these cases, this simple algorithm can produce trees that ***overfit*** the training examples.

We will say that a hypothesis overfits the training examples if some other hypothesis that fits the training examples less well actually performs better over the entire distribution of instances (i.e., including instances beyond the training set).

Definition: **Given a hypothesis space H, a hypothesis *h ε* H is said to overlit the training data if there exists some alternative hypothesis *h' ε* H, such that *h* has smaller error than *h'* over the training examples, but *h'* has a smaller error than *h* over the entire distribution of instances.**

Figure 3.6 illustrates the impact of overfitting in a typical application of decision tree learning. In this case, the ID3 algorithm is applied to the task of learning which medical patients have a form of diabetes. The horizontal axis of this plot indicates the total number of nodes in the decision tree, as the tree is being constructed. The vertical axis indicates the accuracy of predictions made by the tree. The solid line shows the accuracy of the decision tree over the training examples, whereas the broken line shows accuracy measured over an independent set of test examples (not included in the training set). Predictably, the accuracy of the tree over the training examples increases monotonically as the tree is grown. However, the accuracy measured over the independent test examples first increases, then decreases. As can be seen, once the tree size exceeds approximately *25* nodes, further elaboration of the tree decreases its accuracy over the test examples despite increasing its accuracy on the training examples.

How can it be possible for tree h to fit the training examples better than h', but for it to perform more poorly over subsequent examples? One way this can occur is when the training examples contain random errors or noise. To illustrate, consider the effect of adding the following positive training example, incorrectly labeled as negative, to the (otherwise correct) examples in Table 3.2.

(Outlook = Sunny, Temperature = Hot, Humidity = Normal, Wind = Strong, PlayTennis = No)

Given the original error-free data, ID3 produces the decision tree shown in Figure 3.1. However, the addition of this incorrect example will now cause ID3 to construct a more complex tree. In particular, the new example will be sorted into the second leaf node from the left in the learned tree of Figure 3.1, along with the previous positive examples D9 and Dl 1. Because the new example is labeled as a negative example, ID3 will search for further refinements to the tree below this node. Of course as long as the new erroneous example differs in some arbitrary way from the other examples affiliated with this node, ID3 will succeed in finding a new decision attribute to separate out this new example from the two previous positive examples at this tree node. The result is that ID3 will output a decision tree (h) that is more complex than the original tree from Figure 3.1 (h'). Of course h will fit the collection of training examples perfectly, whereas the simpler h' will not. However, given that the new decision node is simply a consequence of fitting the noisy training example, we expect h to outperform h' over subsequent data drawn from the same instance distribution.

The above example illustrates how random noise in the training examples can lead to overfitting. In fact, overfitting is possible even when the training data are noise-free, especially when small numbers of examples are associated with leaf nodes. In this case, it is quite possible for coincidental regularities to occur, in which some attribute happens to partition the examples very well, despite being unrelated to the actual target function. Whenever such coincidental regularities exist, there is a risk of overfitting.

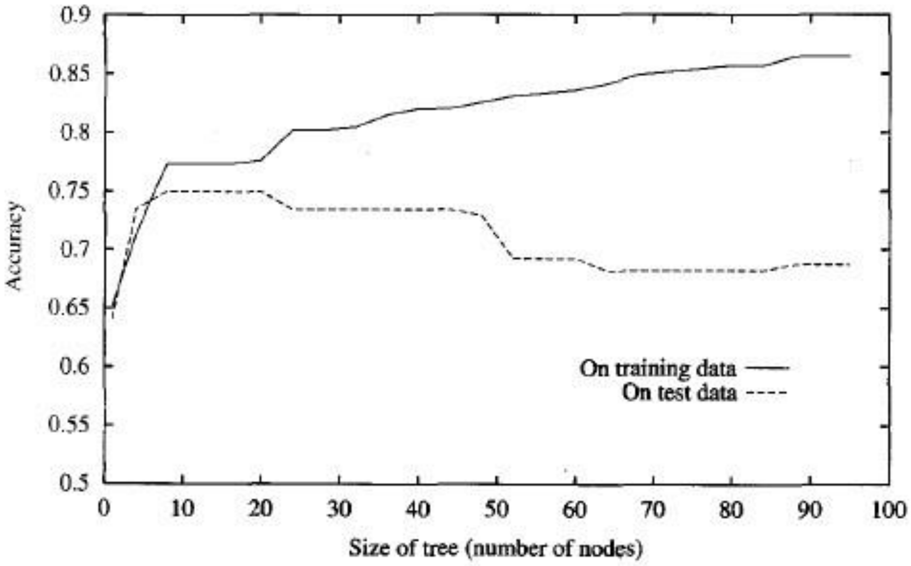


FIGURE 3.6
Overfitting in decision tree learning. As ID3 adds new nodes to grow the decision tree, the accuracy of the tree measured over the training examples increases monotonically. However, when measured over a set of test examples independent of the training examples, accuracy first increases, then decreases. Software and data for experimenting with variations on this plot are available on the World Wide Web at http://www.cs.cmu.edu/~tom/mlbook.html.

Overfitting is a significant practical difficulty for decision tree learning and many other learning methods. For example, in one experimental study of ID3 involving five different learning tasks with noisy, nondeterministic data (Mingers 1989b), overfitting was found to decrease the accuracy of learned decision trees by 10-25% on most problems.

There are several approaches to avoiding overfitting in decision tree learning.

These can be grouped into two classes:

- approaches that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data,
- approaches that allow the tree to overfit the data, and then post-prune the tree.

Although the first of these approaches might seem more direct, the second approach of post-pruning overfit trees has been found to be more successful in practice. This is due to the difficulty in the first approach of estimating precisely
when to stop growing the tree.

Regardless of whether the correct tree size is found by stopping early or by post-pruning, a key question is what criterion is to be used to determine the correct final tree size. Approaches include:

- Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree.
- Use all the available data for training, but apply a statistical test to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set. For example, Quinlan (1986) uses a chi-square test to estimate whether further expanding a node is likely to improve performance over the entire instance distribution, or only on the current sample of training data.
- Use an explicit measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized. This approach, based on a heuristic called the Minimum Description Length principle, is discussed further in Chapter 6, as well as in Quinlan and Rivest (1989) and Mehta et al. (1995).

The first of the above approaches is the most common and is often referred to as a training and validation set approach. We discuss the two main variants of this approach below. In this approach, the available data are separated into two sets of examples: a training set, which is used to form the learned hypothesis, and a separate validation set, which is used to evaluate the accuracy of this hypothesis over subsequent data and, in particular, to evaluate the impact of pruning this hypothesis. The motivation is this: Even though the learner may be misled by random errors and coincidental regularities within the training set, the validation set is unlikely to exhibit the same random fluctuations. Therefore, the validation set can be expected to provide a safety check against overfitting the spurious characteristics of the training set. Of course, it is important that the validation set be large enough to itself provide a statistically significant sample of the instances. One common heuristic is to withhold one-third of the available examples for the validation set, using the other two-thirds for training.

REDUCED ERROR PRUNING

How exactly might we use a validation set to prevent overfitting? One approach, called reduced-error pruning (Quinlan 1987), is to consider each of the decision nodes in the.tree to be candidates for pruning. Pruning a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node. Nodes are removed only if the resulting pruned tree performs no worse than-the original over the validation set. This has the effect that any leaf node added due to coincidental regularities in the training set is likely to be pruned because these same coincidences are unlikely to occur in the validation set. Nodes are pruned iteratively, always choosing the node whose removal most increases the decision tree accuracy over the validation set. Pruning of nodes continues until further pruning is harmful (i.e., decreases accuracy of the tree over the validation set).

The impact of reduced-error pruning on the accuracy of the decision tree is illustrated in Figure **3.7.** As in Figure **3.6,** the accuracy of the tree is shown measured over both training examples and test examples. The additional line in Figure **3.7** shows accuracy over the test examples as the tree is pruned. When pruning begins, the tree is at its maximum size and lowest accuracy over the test set. As pruning proceeds, the number of nodes is reduced and accuracy over the test set increases. Here, the available data has been split into three subsets: the training examples, the validation examples used for pruning the tree, and a set of test examples used to provide an unbiased estimate of accuracy over future unseen examples. The plot shows accuracy over the training and test sets. Accuracy over the validation set used for pruning is not shown.

Using a separate set of data to guide pruning is an effective approach provided a large amount of data is available. The major drawback of this approach is that when data is limited, withholding part of it for the validation set reduces even further the number of examples available for training. The following section presents an alternative approach to pruning that has been found useful in many practical situations where data is limited. Many additional techniques have been proposed **as** well, involving partitioning the available data several different times in multiple ways, then averaging the results. Empirical evaluations of alternative tree pruning methods are reported by Mingers (1989b) and by Malerba et al. (1995).

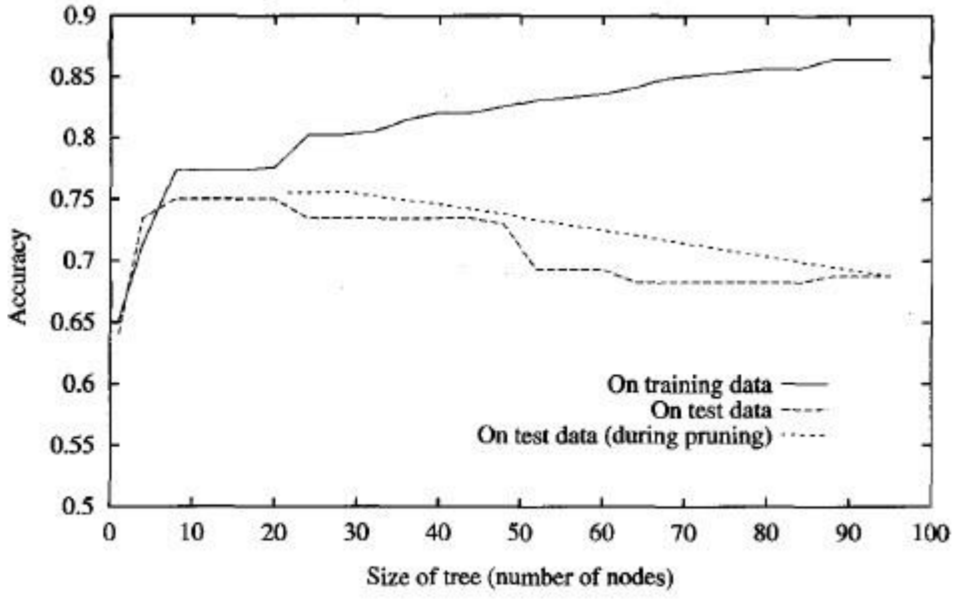


FIGURE 3.7
Effect of reduced-error pruning in decision tree learning. This plot shows the same curves of training and test set accuracy as in Figure 3.6. In addition, it shows the impact of reduced error pruning of the tree produced by ID3. Notice the increase in accuracy over the test set as nodes are pruned from the tree. Here, the validation set used for pruning is distinct from both the training and test sets.

RULE POST-PRUNING

In practice, one quite successful method for finding high accuracy hypotheses is a technique we shall call rule post-pruning. A variant of this pruning method is used by C4.5 (Quinlan 1993), which is an outgrowth of the original ID3 algorithm. Rule post-pruning involves the following steps:

1. Infer the decision tree from the training set, growing the tree until the training data is fit as well as possible and allowing overfitting to occur.

2. Convert the learned tree into an equivalent set of rules by creating one rule for each path from the root node to a leaf node.

3. Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.

4. Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.

To illustrate, consider again the decision tree in Figure 3.1. In rule postpruning, one rule is generated for each leaf node in the tree. Each attribute test along the path from the root to the leaf becomes a rule antecedent (precondition) and the classification at the leaf node becomes the rule consequent (postcondition). For example, the leftmost path of the tree in Figure 3.1 is translated into the rule

IF (Outlook = Sunny) A (Humidity = High) THEN PlayTennis = No

Next, each such rule is pruned by removing any antecedent, or precondition, whose removal does not worsen its estimated accuracy. Given the above rule, for example, rule post-pruning would consider removing the preconditions

(Outlook = Sunny) and (Humidity = High). It would select whichever of these pruning steps produced the greatest improvement in estimated rule accuracy, then consider pruning the second precondition as a further pruning step. No pruning step is performed if it reduces the estimated rule accuracy.

As noted above, one method to estimate rule accuracy is to use a validation set of examples disjoint from the training set. Another method, used by C4.5, is to evaluate performance based on the training set itself, using a pessimistic estimate to make up for the fact that the training data gives an estimate biased in favour of the rules. More precisely, C4.5 calculates its pessimistic estimate by calculating the rule accuracy over the training examples to which it applies, then calculating the standard deviation in this estimated accuracy assuming a binomial distribution. For a given confidence level, the lower-bound estimate is then taken as the measure of rule performance (e.g., for a 95% confidence interval, rule accuracy is pessimistically estimated by the observed accuracy over the training set, minus 1.96 times the estimated standard deviation). The net effect is that for large data sets, the pessimistic estimate is very close to the observed accuracy (e.g., the standard deviation is very small), whereas it grows further from the observed accuracy as the size of the data set decreases. Although this heuristic method is not statistically valid, it has nevertheless been found useful in practice. See Chapter 5 for a discussion of statistically valid approaches to estimating means and confidence intervals.

Why convert the decision tree to rules before pruning? There are three main advantages.

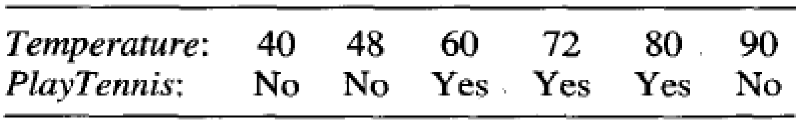
- Converting to rules allows distinguishing among the different contexts in which a decision node is used. Because each distinct path through the decision tree node produces a distinct rule, the pruning decision regarding that attribute test can be made differently for each path. In contrast, if the tree itself were pruned, the only two choices would be to remove the decision node completely, or to retain it in its original form.
- Converting to rules removes the distinction between attribute tests that occur near the root of the tree and those that occur near the leaves. Thus, we avoid messy bookkeeping issues such as how to reorganize the tree if the root node is pruned while retaining part of the subtree below this test.
- Converting to rules improves readability. Rules are often easier for people to understand.

Incorporating Continuous-Valued Attributes

Our initial definition of ID3 is restricted to attributes that take on a discrete set of values. First, the target attribute whose value is predicted by the learned tree must be discrete valued. Second, the attributes tested in the decision nodes of

the tree must also be discrete valued. This second restriction can easily be removed so that continuous-valued decision attributes can be incorporated into the learned tree. This can be accomplished by dynamically defining new discrete valued attributes that partition the continuous attribute value into a discrete set of intervals. In particular, for an attribute A that is continuous-valued, the algorithm can dynamically create a new boolean attribute A_c that is true if A < c
and false otherwise. The only question is how to select the best value for the threshold c.

As an example, suppose we wish to include the continuous-valued attribute ***Temperature*** in describing the training example days in the learning task of Table 3.2. Suppose further that the training examples associated with a particular node in the decision tree have the following values for ***Temperature*** and the target attribute ***PlayTennis.***



Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

What threshold-based boolean attribute should be defined based on Temperature? Clearly, we would like to pick a threshold, ***c,*** that produces the greatest information gain. By sorting the examples according to the continuous attribute
A, then identifying adjacent examples that differ in their target classification, we can generate a set of candidate thresholds midway between the corresponding values of ***A.*** It can be shown that the value of ***c*** that maximizes information gain must always lie at such a boundary (Fayyad 1991). These candidate thresholds can then be evaluated by computing the information gain associated with each. In the current example, there are two candidate thresholds, corresponding to the values of Temperature at which the value of PlayTennis changes: (48 + 60)/2, and (80 + 90)/2.
The information gain can then be computed for each of the candidate attributes, Temperature>54 and Temperature>85 the best can be selected (Temperature>54). This dynamically created boolean attribute can then compete with the other discrete-valued candidate attributes available for growing the decision tree. Fayyad and Irani (1993) discuss an extension to this approach that splits the continuous attribute into multiple intervals rather than just two intervals based on a single threshold. Utgoff and Brodley (1991) and Murthy et al. (1994) discuss approaches that define features by thresholding linear combinations of several continuous-valued attributes.

Alternative Measures for Selecting Attributes

There is a natural bias in the information gain measure that favors attributes with many values over those with few values. As an extreme example, consider the attribute Date, which has a very large number of possible values (e.g., March 4, 1979). If we were to add this attribute to the data in Table 3.2, it would have the highest information gain of any of the attributes. This is because Date alone perfectly predicts the target attribute over the training data. Thus, it would be selected as the decision attribute for the root node of the tree and lead to a (quite broad) tree of depth one, which perfectly classifies the training data. Of course, this decision tree would fare poorly on subsequent examples, because it is not a useful predictor despite the fact that it perfectly separates the training data.

What is wrong with the attribute Date? Simply put, it has so many possible values that it is bound to separate the training examples into very small subsets. Because of this, it will have a very high information gain relative to the training examples, despite being a very poor predictor of the target function over unseen instances.

One way to avoid this difficulty is to select decision attributes based on some measure other than information gain. One alternative measure that has been used successfully is the gain ratio (Quinlan 1986). The gain ratio measure penalizes attributes such as Date by incorporating a term, called split information, that is sensitive to how broadly and uniformly the attribute splits the data:

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|} \qquad (3.5)$$

where S1 through S, are the ***c*** subsets of examples resulting from partitioning S by the c-valued attribute A. Note that Split Information is actually the entropy of S with respect to the values of attribute A. This is in contrast to our previous
uses of entropy, in which we considered only the entropy of S with respect to the target attribute whose value is to be predicted by the learned tree.

The Gain Ratio measure is defined in terms of the earlier Gain measure, as well as this Split information, as follows

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)} \qquad (3.6)$$

Notice that the Split information term discourages the selection of attributes with many uniformly distributed values. For example, consider a collection of n examples that are completely separated by attribute A (e.g., Date). In this case, the Split information value will be log, n. In contrast, a boolean attribute B that splits the same n examples exactly in half will have Split information of 1. If attributes A and B produce the same information gain, then clearly B will score higher according to the Gain Ratio measure.

One practical issue that arises in using GainRatio in place of Gain to select attributes is that the denominator can be zero or very small when |Si | = |S | for one of the Si. This either makes the GainRatio undefined or very large for

attributes that happen to have the same value for nearly all members of S. To avoid selecting attributes purely on this basis, we can adopt some heuristic such as first calculating the Gain of each attribute, then applying the GainRatio test
only considering those attributes with above average Gain (Quinlan 1986).

An alternative to the GainRatio, designed to directly address the above difficulty, is a distance-based measure introduced by Lopez de Mantaras (1991). This measure is based on defining a distance metric between partitions of 'the data. Each attribute is evaluated based on the distance between the data partition it creates and the perfect partition (i.e., the partition that perfectly classifies the training data). The attribute whose partition is closest to the perfect partition is chosen. Lopez de Mantaras (1991) defines this distance measure, proves that it is not biased toward attributes with large numbers of values, and reports experimental studies indicating that the predictive accuracy of the induced trees is not significantly different from that obtained with the Gain and Gain Ratio measures. However, this distance measure avoids the practical difficulties associated with the GainRatio measure, and in his experiments it produces significantly smaller trees in the case of data sets whose attributes have very different numbers of values.

A variety of other selection measures have been proposed as well (e.g., see Breiman et al. 1984; Mingers 1989a; Kearns and Mansour 1996; Dietterich et al. 1996). Mingers (1989a) provides an experimental analysis of the relative
effectiveness of several selection measures over a variety of problems. He reports significant differences in the sizes of the unpruned trees produced by the different selection measures. However, in his experimental domains the choice of attribute selection measure appears to have a smaller impact on final accuracy than does the extent and method of post-pruning.

Handling Training Examples with Missing Attribute Values

In certain cases, the available data may be missing values for some attributes. For example, in a medical domain in which we wish to predict patient outcome based on various laboratory tests, it may be that the lab test Blood-Test-Result is available only for a subset of the patients. In such cases, it is common to estimate the missing attribute value based on other examples for which this attribute has a known value.

Consider the situation in which ***Gain(S, A)*** is to be calculated at node ***n*** in the decision tree to evaluate whether the attribute ***A*** is the best attribute to test at this decision node. Suppose that ***(x, c(x))*** is one of the training examples in ***S***
and that the value ***A(x)*** is unknown.

One strategy for dealing with the missing attribute value is to assign it the value that is most common among training examples at node ***n.*** Alternatively, we might assign it the most common value among examples at node ***n*** that have the classification ***c(x).*** The elaborated training example using this estimated value for ***A(x)*** can then be used directly by the existing decision tree learning algorithm. This strategy is examined by Mingers (1989a).

A second, more complex procedure is to assign a probability to each of the possible values of ***A*** rather than simply assigning the most common value to ***A(x).*** These probabilities can be estimated again based on the observed frequencies of the various values for ***A*** among the examples at node ***n.*** For example, given a boolean attribute ***A,*** if node ***n*** contains six known examples with ***A*** = 1 and four with ***A*** = 0, then we would say the probability that ***A(x)*** = 1 is 0.6, and the probability that ***A(x)*** = 0 is 0.4. A fractional 0.6 of instance ***x*** is now distributed down the branch for ***A*** = 1, and a fractional 0.4 of ***x*** down the other tree branch. These fractional examples are used for the purpose of computing information ***Gain*** and can be further subdivided at subsequent branches of the tree if a second missing attribute value must be tested. This same fractioning of examples can also be applied after learning, to classify new instances whose attribute values are unknown. In this case, the classification of the new instance is simply the most probable classification, computed by summing the weights of the instance fragments classified in different ways at the leaf nodes of the tree. This method for handling missing attribute values is used in C4.5 (Quinlan 1993).

Handling Attributes with Differing Costs

In some learning tasks the instance attributes may have associated costs. For example, in learning to classify medical diseases we might describe patients in terms of attributes such **as** Temperature, BiopsyResult, Pulse, BloodTestResults, etc. These attributes vary significantly in their costs, both in terms of monetary cost and cost to patient comfort. In such tasks, we would prefer decision trees that use low-cost attributes where possible, relying on high-cost attributes only when needed to produce reliable classifications.

ID3 can be modified to take into account attribute costs by introducing a cost term into the attribute selection measure. For example, we might divide the ***Gain*** by the cost of the attribute, so that lower-cost attributes would be preferred. While such cost-sensitive measures do not guarantee finding an optimal cost-sensitive decision tree, they do bias the search in favor of low-cost attributes.

Tan and Schlimmer (1990) and Tan (1993) describe one such approach and apply it to a robot perception task in which the robot must learn to classify different objects according to how they can be grasped by the robot's manipulator.

In this case the attributes correspond to different sensor readings obtained by movable sonar on the robot. Attribute cost is measured by the number of seconds required to obtain the attribute value by positioning and operating the sonar. They demonstrate that more efficient recognition strategies are learned, without sacrificing classification accuracy, by replacing the information gain attribute selection measure by the following measure

$$\frac{Gain^2(S, A)}{Cost(A)}$$

Nunez (1988) describes a related approach and its application to learning medical diagnosis rules. Here the attributes are different symptoms and laboratory tests with differing costs. His system uses a somewhat different attribute selection

Measure

$$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}$$

where ***w* belongs** [0, 1] is a constant that determines the relative importance of cost versus information gain. Nunez (1991) presents an empirical comparison of these two approaches over a range of tasks.