that any function in propositional logic can be expressed as a decision tree. As an example, the rightmost path in Figure 18.2 is

$$Path = (Patrons = Full \wedge WaitEstimate = 0\text{–}10)\ .$$

For a wide variety of problems, the decision tree format yields a nice, concise result. But some functions cannot be represented concisely. For example, the majority function, which returns true if and only if more than half of the inputs are true, requires an exponentially large decision tree. In other words, decision trees are good for some kinds of functions and bad for others. Is there *any* kind of representation that is efficient for *all* kinds of functions? Unfortunately, the answer is no. We can show this in a general way. Consider the set of all Boolean functions on n attributes. How many different functions are in this set? This is just the number of different truth tables that we can write down, because the function is defined by its truth table. A truth table over n attributes has 2^n rows, one for each combination of values of the attributes. We can consider the "answer" column of the table as a 2^n-bit number that defines the function. That means there are 2^{2^n} different functions (and there will be more than that number of trees, since more than one tree can compute the same function). This is a scary number. For example, with just the ten Boolean attributes of our restaurant problem there are 2^{1024} or about 10^{308} different functions to choose from, and for 20 attributes there are over $10^{300{,}000}$. We will need some ingenious algorithms to find good hypotheses in such a large space.

18.3.3 Inducing decision trees from examples

An example for a Boolean decision tree consists of an $(\mathbf{x}, y)$ pair, where $\mathbf{x}$ is a vector of values for the input attributes, and y is a single Boolean output value. A training set of 12 examples

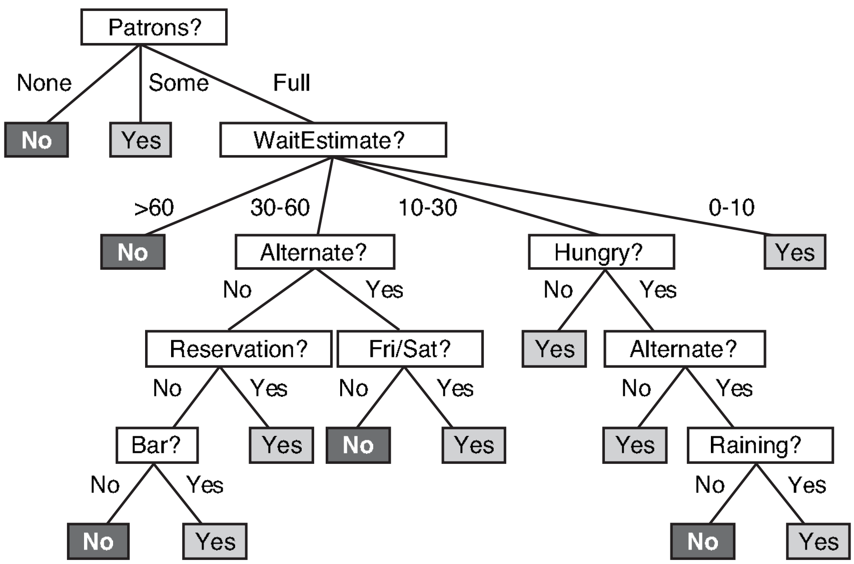

Figure 18.2 A decision tree for deciding whether to wait for a table.

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Figure 18.3 Examples for the restaurant domain.

is shown in Figure 18.3. The positive examples are the ones in which the goal *WillWait* is true $(\mathbf{x}_1, \mathbf{x}_3, \ldots)$; the negative examples are the ones in which it is false $(\mathbf{x}_2, \mathbf{x}_5, \ldots)$.

We want a tree that is consistent with the examples and is as small as possible. Unfortunately, no matter how we measure size, it is an intractable problem to find the smallest consistent tree; there is no way to efficiently search through the 2^{2^n} trees. With some simple heuristics, however, we can find a good approximate solution: a small (but not smallest) consistent tree. The DECISION-TREE-LEARNING algorithm adopts a greedy divide-and-conquer strategy: always test the most important attribute first. This test divides the problem up into smaller subproblems that can then be solved recursively. By "most important attribute," we mean the one that makes the most difference to the classification of an example. That way, we hope to get to the correct classification with a small number of tests, meaning that all paths in the tree will be short and the tree as a whole will be shallow.

Figure 18.4(a) shows that *Type* is a poor attribute, because it leaves us with four possible outcomes, each of which has the same number of positive as negative examples. On the other hand, in (b) we see that *Patrons* is a fairly important attribute, because if the value is *None* or *Some*, then we are left with example sets for which we can answer definitively (*No* and *Yes*, respectively). If the value is *Full*, we are left with a mixed set of examples. In general, after the first attribute test splits up the examples, each outcome is a new decision tree learning problem in itself, with fewer examples and one less attribute. There are four cases to consider for these recursive problems:

1. If the remaining examples are all positive (or all negative), then we are done: we can answer *Yes* or *No*. Figure 18.4(b) shows examples of this happening in the *None* and *Some* branches.
2. If there are some positive and some negative examples, then choose the best attribute to split them. Figure 18.4(b) shows *Hungry* being used to split the remaining examples.
3. If there are no examples left, it means that no example has been observed for this com-

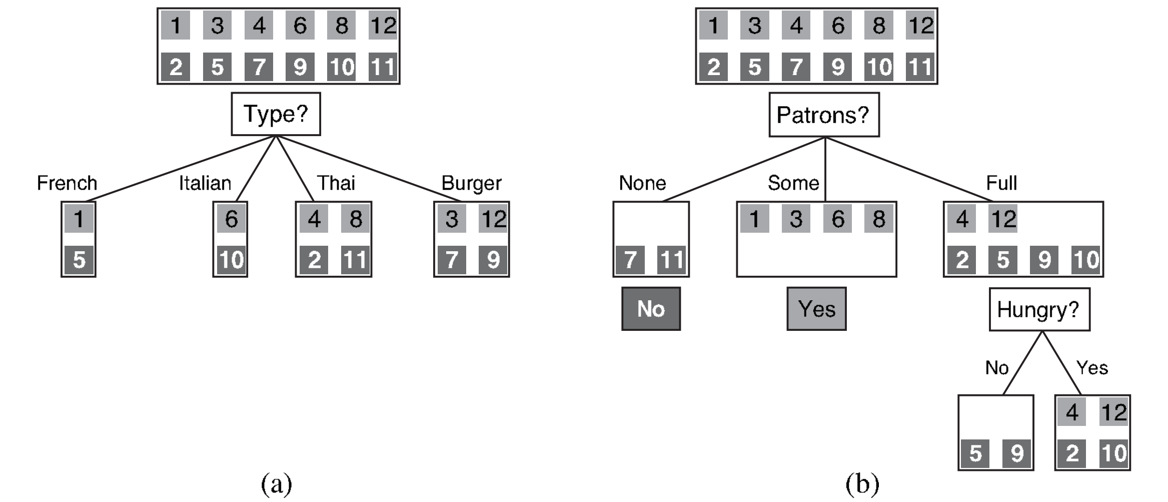

Figure 18.4 Splitting the examples by testing on attributes. At each node we show the positive (light boxes) and negative (dark boxes) examples remaining. (a) Splitting on *Type* brings us no nearer to distinguishing between positive and negative examples. (b) Splitting on *Patrons* does a good job of separating positive and negative examples. After splitting on *Patrons*, *Hungry* is a fairly good second test.

bination of attribute values, and we return a default value calculated from the plurality classification of all the examples that were used in constructing the node's parent. These are passed along in the variable $parent\_examples$.

4. If there are no attributes left, but both positive and negative examples, it means that these examples have exactly the same description, but different classifications. This can happen because there is an error or **noise** in the data; because the domain is nondeterministic; or because we can't observe an attribute that would distinguish the examples. The best we can do is return the plurality classification of the remaining examples.

NOISE

The DECISION-TREE-LEARNING algorithm is shown in Figure 18.5. Note that the set of examples is crucial for *constructing* the tree, but nowhere do the examples appear in the tree itself. A tree consists of just tests on attributes in the interior nodes, values of attributes on the branches, and output values on the leaf nodes. The details of the IMPORTANCE function are given in Section 18.3.4. The output of the learning algorithm on our sample training set is shown in Figure 18.6. The tree is clearly different from the original tree shown in Figure 18.2. One might conclude that the learning algorithm is not doing a very good job of learning the correct function. This would be the wrong conclusion to draw, however. The learning algorithm looks at the *examples*, not at the correct function, and in fact, its hypothesis (see Figure 18.6) not only is consistent with all the examples, but is considerably simpler than the original tree! The learning algorithm has no reason to include tests for $Raining$ and $Reservation$, because it can classify all the examples without them. It has also detected an interesting and previously unsuspected pattern: the first author will wait for Thai food on weekends. It is also bound to make some mistakes for cases where it has seen no examples. For example, it has never seen a case where the wait is 0–10 minutes but the restaurant is full.

```
function DECISION-TREE-LEARNING(examples, attributes, parent_examples) returns
a tree

  if examples is empty then return PLURALITY-VALUE(parent_examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY-VALUE(examples)
  else
     A ← argmax_{a ∈ attributes} IMPORTANCE(a, examples)
     tree ← a new decision tree with root test A
     for each value v_k of A do
        exs ← {e : e ∈ examples and e.A = v_k}
        subtree ← DECISION-TREE-LEARNING(exs, attributes − A, examples)
        add a branch to tree with label (A = v_k) and subtree subtree
     return tree
```

Figure 18.5 The decision-tree learning algorithm. The function IMPORTANCE is described in Section 18.3.4. The function PLURALITY-VALUE selects the most common output value among a set of examples, breaking ties randomly.

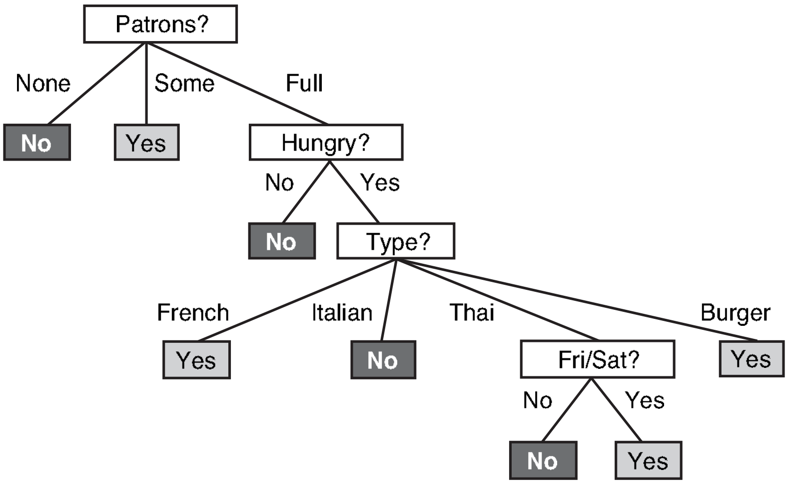

Figure 18.6 The decision tree induced from the 12-example training set.

In that case it says not to wait when *Hungry* is false, but I (SR) would certainly wait. With more training examples the learning program could correct this mistake.

We note there is a danger of over-interpreting the tree that the algorithm selects. When there are several variables of similar importance, the choice between them is somewhat arbitrary: with slightly different input examples, a different variable would be chosen to split on first, and the whole tree would look completely different. The function computed by the tree would still be similar, but the structure of the tree can vary widely.

LEARNING CURVE

We can evaluate the accuracy of a learning algorithm with a **learning curve**, as shown in Figure 18.7. We have 100 examples at our disposal, which we split into a training set and

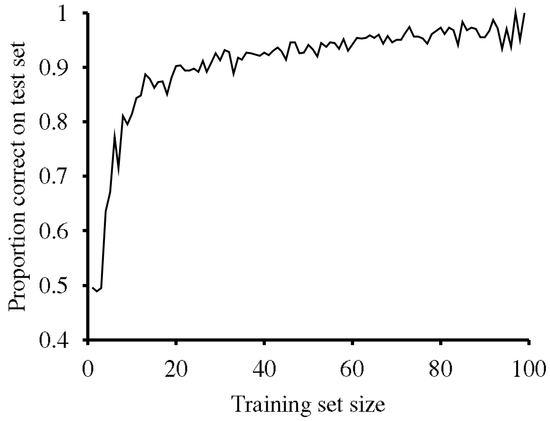

Figure 18.7 A learning curve for the decision tree learning algorithm on 100 randomly generated examples in the restaurant domain. Each data point is the average of 20 trials.

a test set. We learn a hypothesis h with the training set and measure its accuracy with the test set. We do this starting with a training set of size 1 and increasing one at a time up to size 99. For each size we actually repeat the process of randomly splitting 20 times, and average the results of the 20 trials. The curve shows that as the training set size grows, the accuracy increases. (For this reason, learning curves are also called **happy graphs**.) In this graph we reach 95% accuracy, and it looks like the curve might continue to increase with more data.

18.3.4 Choosing attribute tests

The greedy search used in decision tree learning is designed to approximately minimize the depth of the final tree. The idea is to pick the attribute that goes as far as possible toward providing an exact classification of the examples. A perfect attribute divides the examples into sets, each of which are all positive or all negative and thus will be leaves of the tree. The *Patrons* attribute is not perfect, but it is fairly good. A really useless attribute, such as *Type*, leaves the example sets with roughly the same proportion of positive and negative examples as the original set.

All we need, then, is a formal measure of "fairly good" and "really useless" and we can implement the IMPORTANCE function of Figure 18.5. We will use the notion of information gain, which is defined in terms of **entropy**, the fundamental quantity in information theory (Shannon and Weaver, 1949).

ENTROPY

Entropy is a measure of the uncertainty of a random variable; acquisition of information corresponds to a reduction in entropy. A random variable with only one value—a coin that always comes up heads—has no uncertainty and thus its entropy is defined as zero; thus, we gain no information by observing its value. A flip of a fair coin is equally likely to come up heads or tails, 0 or 1, and we will soon show that this counts as "1 bit" of entropy. The roll of a fair *four*-sided die has 2 bits of entropy, because it takes two bits to describe one of four equally probable choices. Now consider an unfair coin that comes up heads 99% of the time. Intuitively, this coin has less uncertainty than the fair coin—if we guess heads we'll be wrong only 1% of the time—so we would like it to have an entropy measure that is close to zero, but