Chapter 18 (AIMA)
Learning From Examples

Decision Tree

A decision tree

- Represents a function that takes as input a vector of attribute values
- Returns a “decision”—a single output value.
- The input and output values can be discrete or continuous.
- For now we will concentrate on problems where
  - Inputs have discrete values
  - Output has exactly two possible values; this is Boolean classification, where each example input will be classified as true (a **positive** example) or false (a **negative** example).

Decision Tree

In a decision tree:

- Each internal node in the tree corresponds to a test of the value of one of the input attributes,

- The branches from the node are labeled with the possible values of the attribute, .

- Each leaf node in the tree specifies a value to be returned by the function.

Decision Tree: Example

Goal:

- WillWait - to decide whether to wait for a table at a restaurant.
- Goal is binary valued (i.e., binary classification task)

Decision Tree: Example

Attributes:

1. Alternate: whether there is a suitable alternative restaurant nearby.
2. Bar : whether the restaurant has a comfortable bar area to wait in.
3. Fri◁Sat: true on Fridays and Saturdays.
4. Hungry: whether we are hungry.
5. Patrons: how many people are in the restaurant (values are None, Some, and Full).
6. Price: the restaurant’s price range ($, $$, $$$).

Decision Tree Example

Attributes:

7. Raining: whether it is raining outside.
8. Reservation: whether we made a reservation.
9. Type: the kind of restaurant (French, Italian, Thai, or burger).
10. WaitEstimate: the wait estimated by the host (0–10 minutes, 10–30, 30–60, or $>$60).

- Note that every variable has a small set of possible values; the value of WaitEstimate, for example, is not an integer, rather it is one of the four discrete values 0–10, 10–30, 30–60, or $>$60.

Decision Tree Example

An example decision tree:

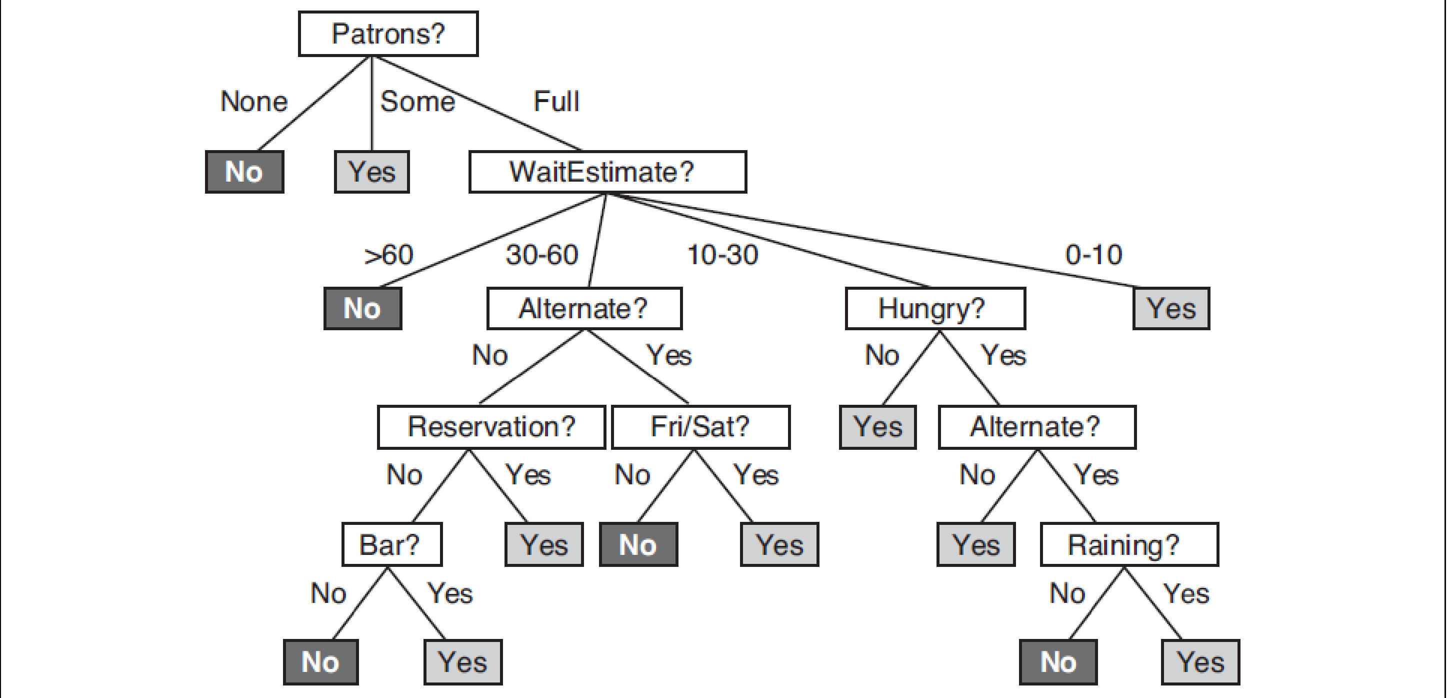


Figure 18.2 A decision tree for deciding whether to wait for a table.

Expressiveness of Decision Trees

Expressiveness of decision trees:

- 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.
- Decision trees are good for some kinds of functions and bad for others

Expressiveness of Decision Trees

- In a Boolean decision tree, the goal attribute is true if and only if the input attributes satisfy one of the paths leading to a leaf with value true.

- Goal $\Leftrightarrow$ (Path1 $\vee$ Path2 $\vee \cdots$), where each Path is a conjunction of attribute-value tests required to follow that path.

- Thus, the whole expression is equivalent to disjunctive normal form, which means that any function in propositional logic can be expressed as a decision tree.

Expressiveness of Decision Trees

- A truth table over n attributes has 2^n rows, one for each combination of values of the attributes.

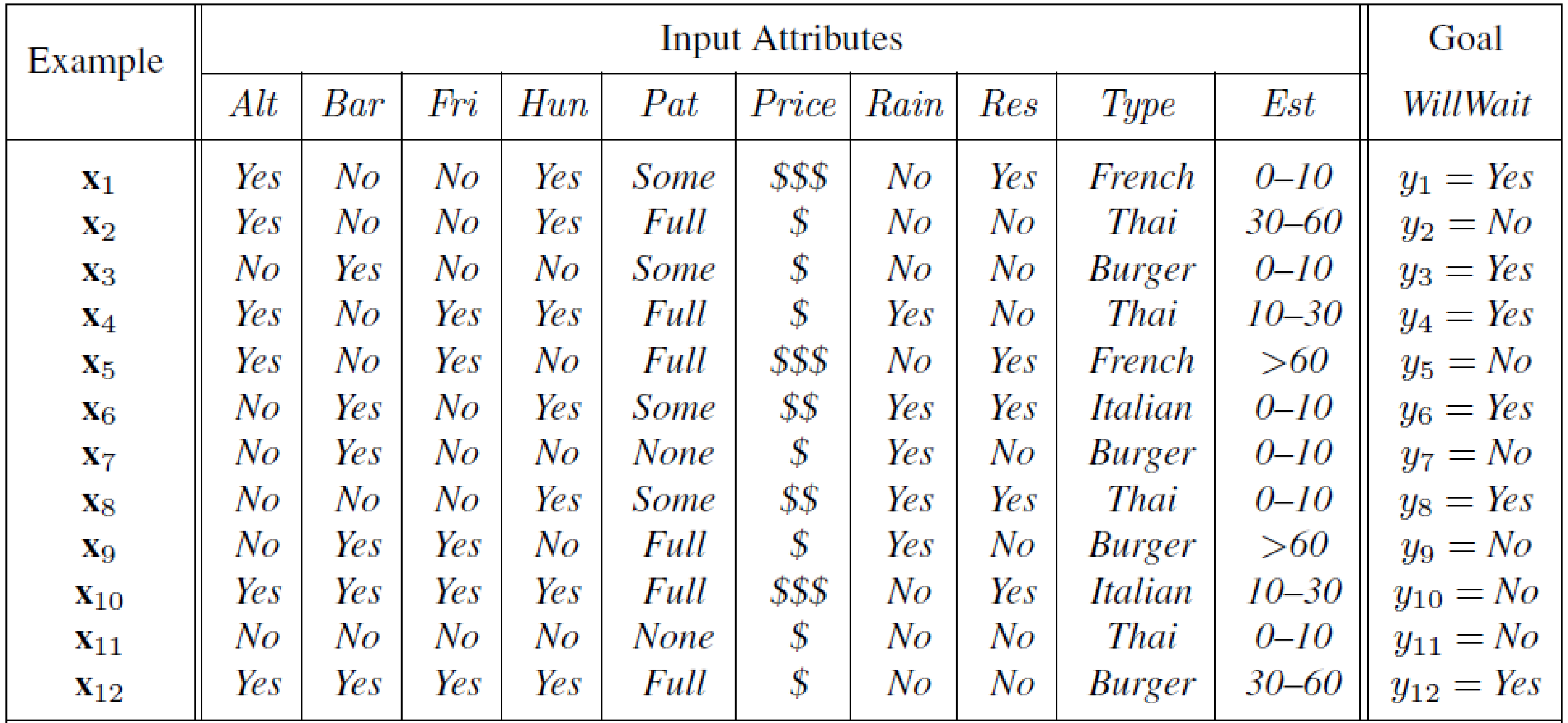
- We can consider the "answer" column of the table as a 2^n-bit number that defines the function. That means there are 2^(2^n) different functions (and there will be more than that number of trees, since more than one tree can compute the same function).

- 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.

Inducing Decision Trees

An example for a Boolean decision tree consists of an $(\mathbf{x}, y)$ pair

- where **x** is a vector of values for the input attributes, $y$is a single Boolean out



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.

Inducing Decision Trees

- We want a tree that is
  - Consistent with the examples
  - 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, we can find a good approximate solution: a small (but not smallest) consistent tree.

Decision Tree Learning

- The decision tree learning algorithm adopts a greedy divide-and-conquer strategy:
    - Always test the most important attribute first.
        - 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.

    - The test divides the problem up into smaller subproblems that can then be solved recursively.

Decision Tree Learning

Which attribute to test at root? Type vs Patron?

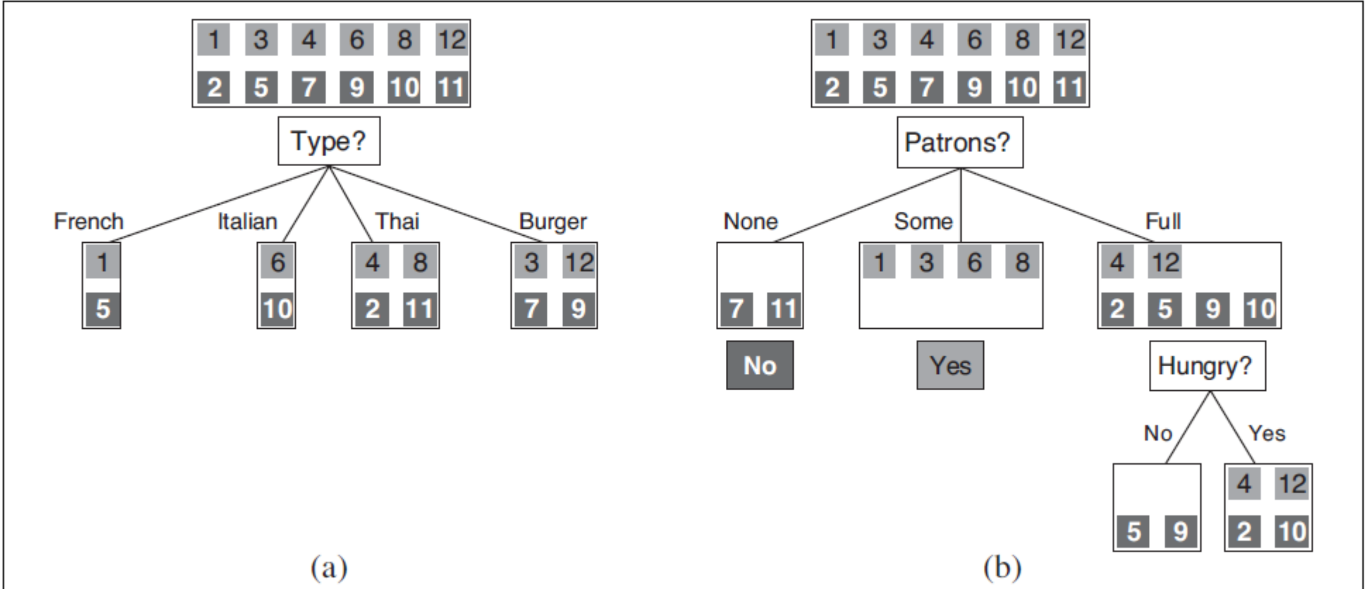


Figure 18.4 Splitting the examples by testing on attributes. At each node we show the positive (light boxes) and negative (dark boxes) examples remaining. (a) Splitting on *Type* brings us no nearer to distinguishing between positive and negative examples. (b) Splitting on *Patrons* does a good job of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

Decision Tree Learning Algorithm

Algorithm steps:

1. Test an attribute at each node.

2. Partition the examples according to values and create child nodes with relevant examples.

3. Now consider child node for further tests of attributes except the one which have already been tested in the hierarchy (recursive operation)

.

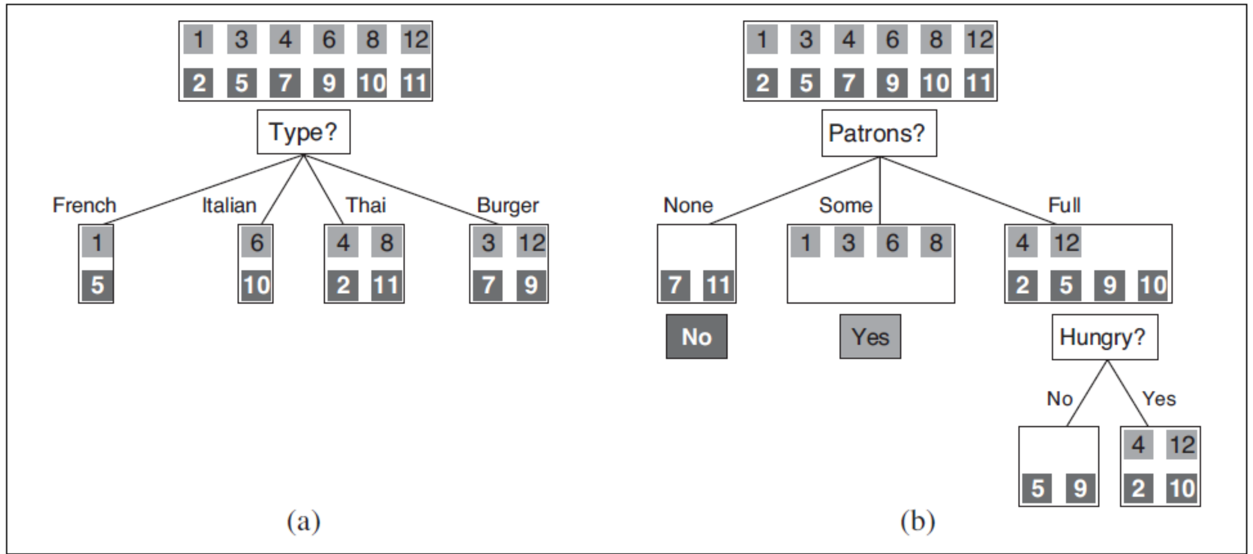


Figure 18.4 Splitting the examples by testing on attributes. At each node we show the positive (light boxes) and negative (dark boxes) examples remaining. (a) Splitting on *Type* brings us no nearer to distinguishing between positive and negative examples. (b) Splitting on *Patrons* does a good job of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

Decision Tree Learning Algorithm

There are four cases to consider at a child node:

1. If the examples are all positiv (or all negative), then we are dor we can answer Yes or No.

2. If there are some positive and some negative examples, then choose the best attribute to split (recursive operation).

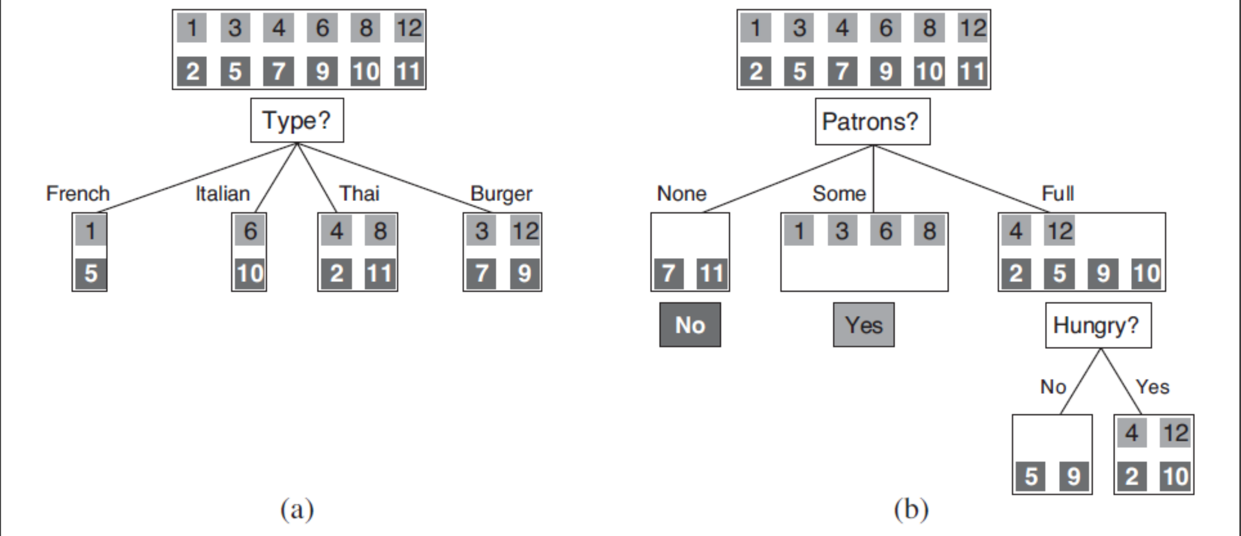


Figure 18.4 Splitting the examples by testing on attributes. At each node we show the positive (light boxes) and negative (dark boxes) examples remaining. (a) Splitting on *Type* brings us no nearer to distinguishing between positive and negative examples. (b) Splitting on *Patrons* does a good job of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

Decision Tree Learning Algorithm

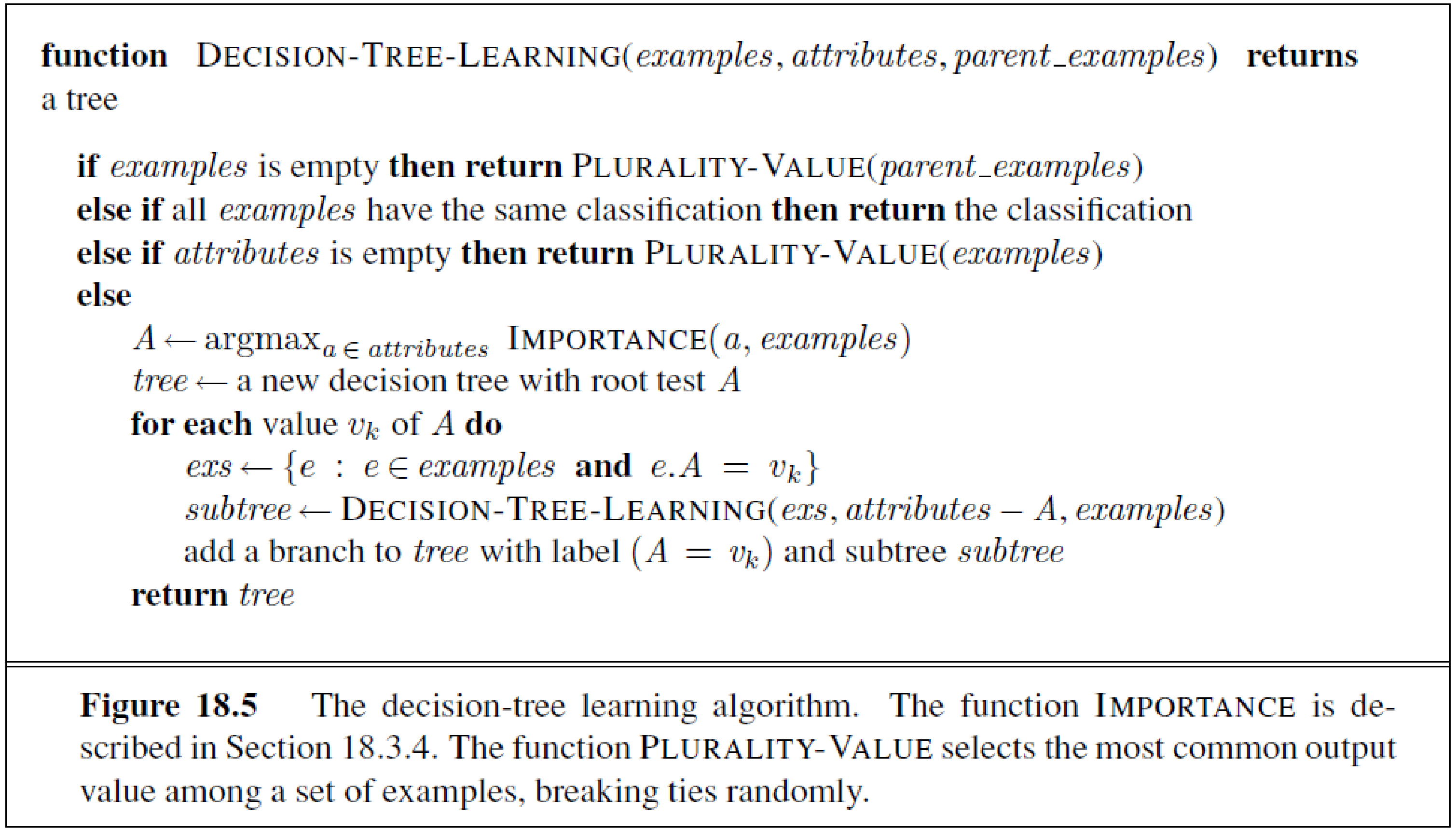
There are four cases to consider at a child node:

3. If there are no examples left, it means that no example has been observed for this combination of attribute values, and we return a default value (*plurality of parent node*)

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. *In this case return the plurality classification of the remaining examples*.

Decision Tree Learning Algorithm

Algorithm pseudocode



```
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
     $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE($a, examples$)
     $tree \leftarrow$ a new decision tree with root test $A$
     for each value $v_k$ of $A$ do
        $exs \leftarrow \{e : e \in examples$ and $e.A = v_k\}$
        $subtree \leftarrow$ 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.

Choosing Attribute Tests

- The greedy search used in decision tree learning is designed to approximately minimize the depth of the final tree.

- **Entropy:** 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 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.

Choosing Attribute Tests

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 positive.

Choosing Attribute Tests

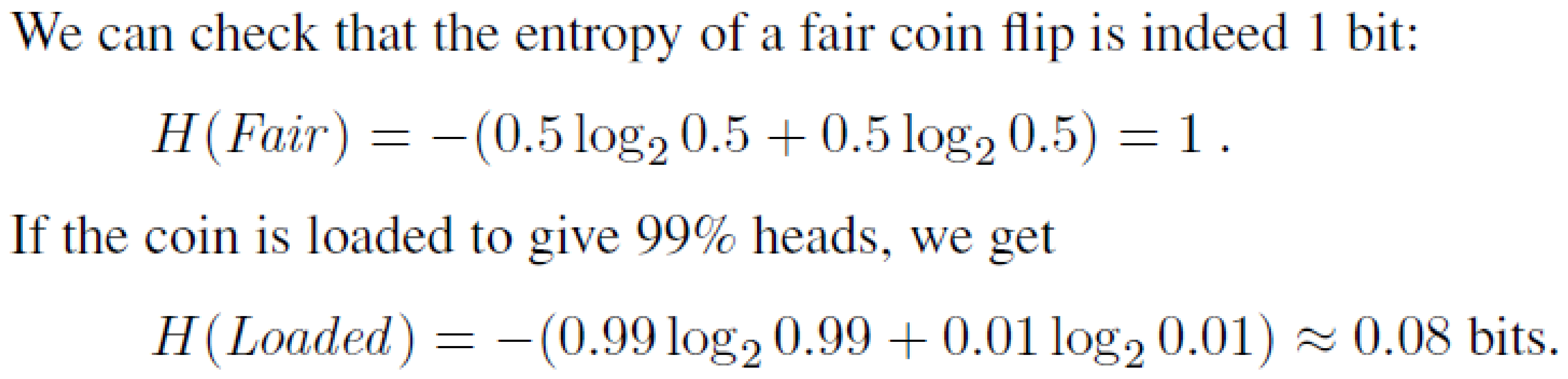
Entropy:

- In general, the entropy of a random variable $V$with values , each with probability is defined as :

$$\text{Entropy:} \quad H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k) \ .$$

Choosing Attribute Tests

Entropy:



We can check that the entropy of a fair coin flip is indeed 1 bit:

$$H(Fair) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1\,.$$

If the coin is loaded to give 99% heads, we get

$$H(Loaded) = -(0.99 \log_2 0.99 + 0.01 \log_2 0.01) \approx 0.08 \text{ bits.}$$

Choosing Attribute Tests

Entropy:

- Define as the entropy of a Boolean random variable that is true with probability

$$B(q) = -(q \log_2 q + (1 - q) \log_2 (1 - q)) \ .$$

Choosing Attribute Tests

Entropy:

- If a training set contains positive examples and negative examples, then the entropy of the goal attribute on the whole set is

$$H(Goal) = B\left(\frac{p}{p+n}\right) .$$

Choosing Attribute Tests

Entropy:

- An attribute $A$with distinct values divides the training set into subsets ᶜ ▷▷▷ᶜ.

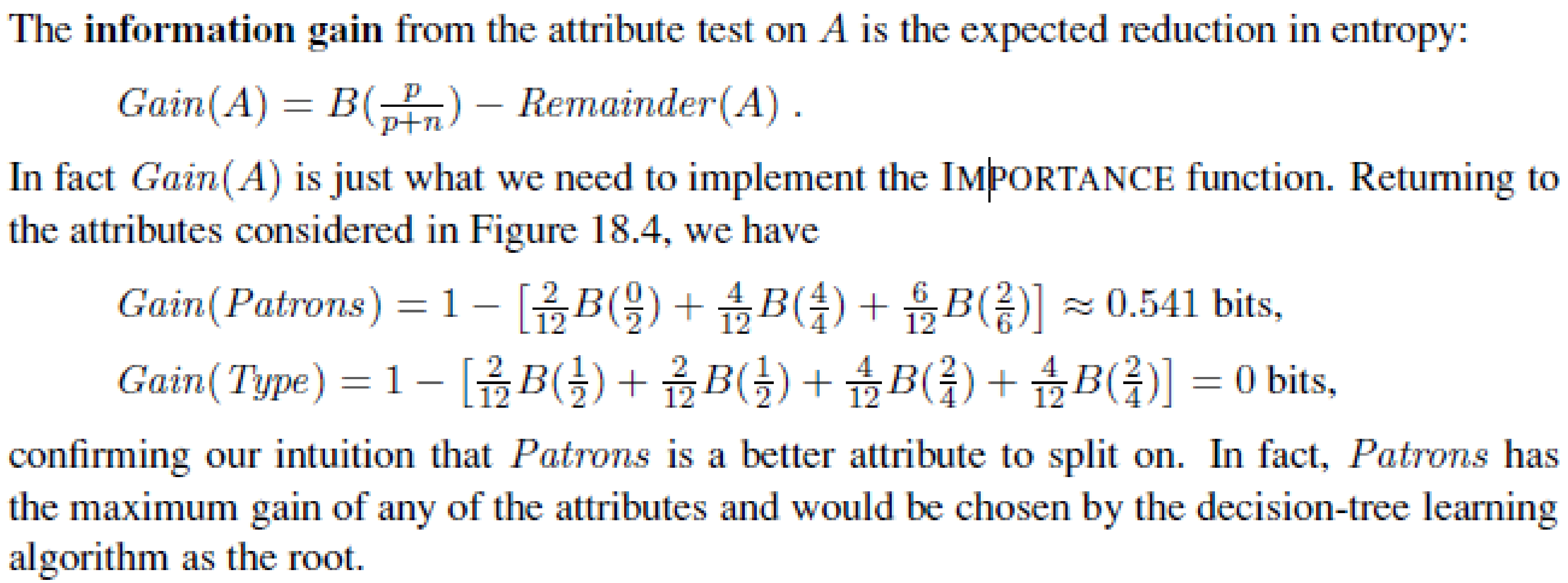
- Each subset has positive examples and negative examples, with entropy of bits of information.

- A randomly chosen example from the training set has the kth value for the attribute with probability so the expected entropy of the subsets after testing attribute $A$is:

$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right) .$$

Choosing Attribute Tests

Information Gain:



The **information gain** from the attribute test on A is the expected reduction in entropy:

$$Gain(A) = B(\tfrac{p}{p+n}) - Remainder(A)\ .$$

In fact $Gain(A)$ is just what we need to implement the IMPORTANCE function. Returning to the attributes considered in Figure 18.4, we have

$$Gain(Patrons) = 1 - [\tfrac{2}{12}B(\tfrac{0}{2}) + \tfrac{4}{12}B(\tfrac{4}{4}) + \tfrac{6}{12}B(\tfrac{2}{6})] \approx 0.541 \text{ bits},$$

$$Gain(Type) = 1 - [\tfrac{2}{12}B(\tfrac{1}{2}) + \tfrac{2}{12}B(\tfrac{1}{2}) + \tfrac{4}{12}B(\tfrac{2}{4}) + \tfrac{4}{12}B(\tfrac{2}{4})] = 0 \text{ bits},$$

confirming our intuition that *Patrons* is a better attribute to split on. In fact, *Patrons* has the maximum gain of any of the attributes and would be chosen by the decision-tree learning algorithm as the root.

Decision Tree

Final decision tree from 12 examples

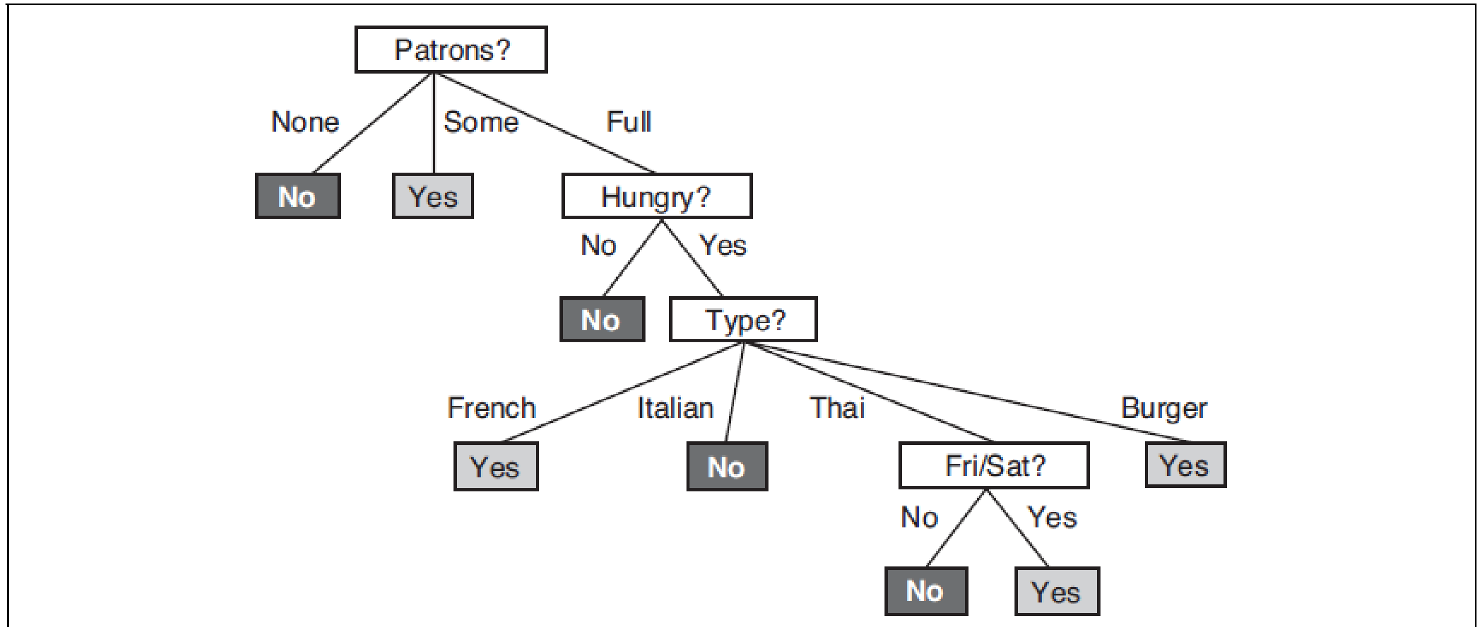


Figure 18.6 The decision tree induced from the 12-example training set.

Decision Tree Learning Curve

- 100 examples
- Split into train and test
  (e.g., 99 and 1)
- Random split 20 times a
  report average accurac
  on test set
- As the training size grow
  accuracy increase.

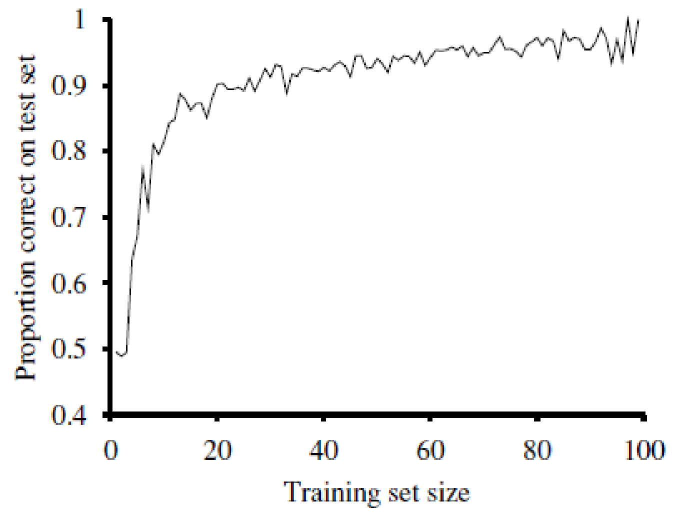


Figure 18.7 A learning curve for the decision tree learning algorithm on 100 randomly generated examples in the restaurant domain. Each data point is the average of 20 trials.