Suppose we have $n$ samples of feature vectors $\vec{x}_i$, where each is comprised of $m$ features, $\vec{x}_i = (x_{i,1},\ x_{i,2},\ \ldots,\ x_{i,k},\ \ldots,\ x_{i,m})$. (In the case of the text data, the features were stopword counts.) These can be stacked into an $n \times m$ matrix with entries $M_{ij} = x_{i,j}$:

$$M = \begin{pmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vdots \\ \vec{x}_n \end{pmatrix}$$

If we're interested in visualizing or clustering these $n$ samples, we need to measure distances between them. Although it is difficult to visualize directly in $m$ dimensions, the distances are encapsulated in the pairwise dot products $\vec{x}_i \cdot \vec{x}_j = \sum_{k=1}^{m} x_{i,k}\, x_{j,k}$ . The idea behind dimensional reduction is to find some lower dimensional representation $\vec{y}_i$ for the $\vec{x}_i$ that encapsulates the same pairwise relations.

Notice that these relations are naturally encoded in the matrix product $MM^T$ (where $M^T$ is the transpose of $M$, $M^T_{ij} = M_{ji}$):

$$(MM^T)_{ij} = \vec{x}_i \cdot \vec{x}_j$$

By construction, $MM^T$ is a symmetric matrix. A fundamental property of symmetric matrices (known as eigenvalue decomposition) is they can be written in the form

$$MM^T = ULU^T \tag{1}$$

where $U$ is an "orthogonal" matrix, with $U^T = U^{-1}$ (so that $UU^T = U^TU = \mathbf{1}$, with $\mathbf{1}$ the $n \times n$ identity matrix diag(1,1,...,1), i.e., 1's along the diagonal and 0 elsewhere), and $L$ is a diagonal matrix of the form

$$L = \begin{pmatrix} s_1^2 & & & \\ & s_2^2 & & \\ & & \ddots & \\ & & & s_n^2 \end{pmatrix} .$$

The $U^T$ matrix acts as a rotation to a frame in which the matrix looks as simple as possible, stretching or shrinking the $i^{\text{th}}$ axis by the eigenvalue $s_i^2$, and $U$ then rotates back to the original frame.

The above decomposition is unique, up to reordering of the eigenvalues, and by convention they're in descending order with $s_1^2$ the largest. For a general matrix, the eigenvalues can be negative, but a matrix of the form $MM^T$ will automatically have positive eigenvalues $\ell_i$, so they're written as $\ell_i = s_i^2$ in the above. This means we can also write $L = S^2$

where $S = \mathrm{diag}(s_1, \ldots, s_n)$, and then $MM^T = (US)(US)^T$, where $US$ is the matrix $U$ with all elements of the $k^{\text{th}}$ column scaled by $s_k$.

Suppose we wish to approximate the matrix $MM^T$ by a matrix with fewer parameters, i.e., to implement a "dimensional reduction". We can approximate the matrix by ignoring the smallest eigenvalues, i.e., successively zeroing them out. So if it turns out for example that the first $\ell$ are much larger than the rest, then we can set the rest to zero and substitute $S_\ell = \mathrm{diag}(s_1, s_2, \ldots, s_\ell,\ 0, \ldots, 0)$ to approximate $MM^T$ as $(US_\ell)(US_\ell)^T$. The mathematical statement is that this is the best approximation to $MM^T$ with $\ell$ such parameters.

Consider the case ($\ell = 2$) when only the first two $s_i^2$ are substantial. Then $MM^T$ is approximated by

$$(MM^T)_{\text{approx}} = US_2S_2U^T = \begin{pmatrix} s_1u_{11} & s_2u_{12} & \\ s_1u_{21} & s_2u_{22} & \\ \vdots & \vdots & \Large 0 \\ s_1u_{n1} & s_2u_{n2} & \end{pmatrix} \begin{pmatrix} s_1u_{11} & \cdots & s_1u_{n1} \\ s_2u_{12} & \cdots & s_2u_{n2} \\ & & \\ & \Large 0 & \\ & & \end{pmatrix}$$

This tells us that the $\vec{y}_i$ defined as the rows of $US_2$:

$$\begin{aligned} \vec{y}_1 &= (s_1u_{11}, s_2u_{12}) \\ &\vdots \\ \vec{y}_i &= (s_1u_{i1}, s_2u_{i2}) \qquad (2) \\ &\vdots \\ \vec{y}_n &= (s_1u_{n1}, s_2u_{n2}) \end{aligned}$$

satisfy $(MM^T)_{\text{approx}} = \vec{y}_i \cdot \vec{y}_j$. The $\vec{y}_i$ are thus the desired dimensionally reduced versions of the original $\vec{x}_i$'s, best capturing their pairwise dot products, $\vec{x}_i \cdot \vec{x}_j \approx \vec{y}_i \cdot \vec{y}_j$, with just $\ell = 2$ dimensional vectors. This permits us to plot the $\vec{y}_i$ in two dimensions to see how the original data is clustered.

The python realization of eqn. (2) in the assignment notebook used

U,S,Vt = linalg.svd(M)  # decomposes $M = USV^T$

to calculate directly* the matrices $U$ and $S$. The $\vec{y}_i$ of eqn. (2) are given by truncating rows of $US$, in python as US[:,:2] (or US[:,:$\ell$] for $\ell > 2$), where the matrix product $US$ is

* The SVD="singular value decomposition" of an $n \times m$ matrix $M$ calculates as well an $m \times m$ matrix $V$, which like the $n \times n$ matrix $U$ satisfies $VV^T = \mathbf{1}$, and $S$ is diagonal with at most $\min(n, m)$ non-zero elements $s_i$.

given by

US = U[:,:len(S)].dot(diag(S)) .  # len(S) in case $n > m$

Equivalently, the eigenvalues of the symmetric $n \times n$ matrix $MM^T$ can be calculated as in eqn. (1) using

L,O = linalg.eigh(M.dot(M.T))

where O and L will agree with U and S*S, respectively, up to reordering. These can be compared using some random $M$, e.g., M=random.random([18,50]), but since linalg.eigh() happens to return eigenvalues in *ascending* order, the comparison requires first setting L=L[::-1] and O=O[:,::-1] to reverse the order. Then the sequences of values in S and sqrt(L) will agree (as easily confirmed by checking that norm(S-sqrt(L)) vanishes).

For circumstances in which there's a restricted set of alternatives, labelled say $i = 1, \ldots, d$, and each of which can be assigned a probability $p_i$, there is a way to quantify the "information uncertainty" (due to Shannon), in bits of information:

$$H = \sum_{i=1}^{d} p_i \log_2 \frac{1}{p_i} = -\sum_{i=1}^{d} p_i \log_2 p_i \ . \tag{1}$$

For example, in the case of flipping a single fair coin, we would have $p_1 = p_2 = 1/2$, so

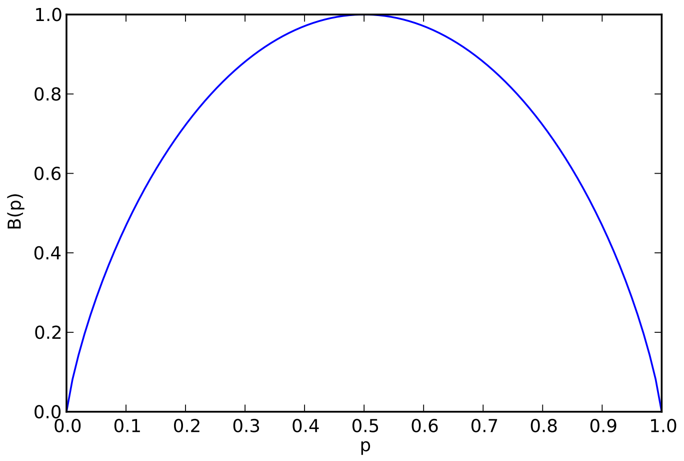
$$H = \frac{1}{2}\log_2 2 + \frac{1}{2}\log_2 2 = \frac{1}{2} + \frac{1}{2} = 1 \text{ bit} \ ,$$

which is the amount of information conveyed by a single H/T (heads/tails) result. Similarly, the amount of information in 2 fair coin flips ($p_i = 1/4$) is 2 bits, and the amount in 3 fair coin flips ($p_i = 1/8$) is 3 bits.

In general, if there are $n$ equally likely possibilities, then the formula reduces to

$$H = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} = \sum_{i=1}^{n} \frac{1}{n} \log_2 n = \log_2 n \ \text{ bits} \ ,$$

so the amount of information conveyed by rolling a fair die ($p_i = 1/6$) is $\log_2 6 \approx 2.6$, intermediate between 2 and 3 bits. The information in eqn. (1) is maximized in this equiprobability case: when the $p_i$ are not all equal (but of course still sum to 1), then the information uncertainty is always less than $\log_2 n$. For example, a coin that has a 99% probability of coming up heads has $H = -(99/100)\log_2(99/100) - (1/100)\log_2(1/100) \approx .08$ bits of information, where much less than 1 bit of information is acquired since there was already a large likelihood the result would be heads. The general result for the coin that comes up H with probability $p \in [0,1]$ is given by $B(p) = -p\log_2 p - (1-p)\log_2(1-p)$, and has maximum of 1 bit for $p = 1/2$:



```
def B(q):
   if q==0 or q==1:
      return 0.
   else:
      return -q*math.log(q,2)-(1-q)*math.log(1-q,2)
p=linspace(0,1,101)
H=map(B,p)      # or H=[B(q) for q in p]
xticks(linspace(0,1,11))
xlabel('p')
ylabel('B(p)')
plot(p,H)
savefig('Bp.pdf')
```

The formula (1) has the property that the information is additive when combining independent systems, as in the cases of the multiple coin flips above. This is easy to see in the special case of two systems with $m$ and $n$ equiprobability possibilities, respectively, where the information satisfies $H = \log_2 mn = \log_2 m + \log_2 n$ and is hence the sum of the information uncertainties for the two subsystems (this is ultimately the reason for the logarithm). For example, in the case of flipping a coin and rolling a die, there are 12 possibilities with $p_i = 1/12$ (coin is H, die is 1, etc.) and the total information is $\log_2 12 = \log_2 2 + \log_2 6 \approx 3.6$.

Shannon (1948) showed that eqn.(1) is the unique measure of information (up to overall normalization, which we choose to measure in bits) which has the properties: additive as described above (and independent of the order in which the system is divided into parts), continuous in all the $p_i$, symmetric in the $p_i$ (i.e., independent of their order), is maximized when all possibilities are equally likely, and increases with that total number of possibilities.

To get more intuition into what it means to have a non-integer number of bits of information, consider a stream of letters $ABAAcBABd\ldots$ generated by the probability distribution: $p_A = 1/2$, $p_B = 1/4$, $p_c = 1/8$, $p_d = 1/8$. The information content is

$$H = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{4}\log\frac{1}{4} - \frac{1}{8}\log\frac{1}{8} - \frac{1}{8}\log\frac{1}{8} = \frac{1}{2} + \frac{1}{2} + \frac{3}{8} + \frac{3}{8} = 1\frac{3}{4}\ .$$

In order to transmit the stream of information, naively it would take 2 bits/character to represent each of four possibilities. But can we somehow encode a stream of unequal probability alternatives using fewer bits/character on average? The idea is to reserve the smallest number of bits for the most frequent characters to reduce the average usage. So we represent the most probable $A$ by a single 0, and use 1 as a signal that we need to look at the next bit (known as a "prefix code"). Then we can represent $B$ by 10, and use 11 to signal that we need to look at a third bit, so $c$ can be encoded as 110 and $d$ as 111. The sequence $ABAAcBABd$ is then encoded as 0 10 0 0 110 10 0 10 111, and using the rules we can see that the sequence 0100011010010111 can be unambiguously decoded as $ABAAcBABd$. What is the average number of bits per character used in this scheme? Since $A$ occurs half the time and needs 1 bit, $B$ occurs 1/4 of the time and needs 2 bits, $c$ and $d$ each occur 1/8 of the time and need 3 bits, on average this means

$$\frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 2 + \frac{1}{8}\cdot 3 + \frac{1}{8}\cdot 3 = 1\frac{3}{4} \text{ bits per character.}$$

In general, the Shannon information gives the number of bits per character for an optimal encoding scheme (if it exists), resulting in the optimal compression ratio.

Shannon also experimented with estimating the bits/character of English text. Since the probabilities of the 26 letters are far from equal, we expect somewhat less than the maximum $\log_2 26 \approx 3.7$ bits/char. The uncertainty per character was first estimated by giving people a section of text and asking them to guess the next letter (in principle they could employ 1- and higher gram probability distributions of letter co-occurrences to aid them, now this is easier with computers ...). The typical estimates are in the range 1–1.5 bits/char, which is why it's possible to compress text files (using gz, zip, or equivalent).

With the notion of information in hand, there's a related notion of *mutual information* shared by two random variables. Roughly speaking, it quantifies the extent to which they are dependent (so that independent random variables have zero mutual information). It is fun to describe this metaphorically. Consider that the world $W$ consists of a set of states $w \in W$ with some probability distribution $\sum_{w \in W} p(w) = 1$, and associated information uncertainty $H[W] = -\sum_{w \in W} p(w) \log_2 p(w)$. Now imagine there are certain types of data $d \in D$ that we can measure, which as well come with some probability distribution $\sum_{d \in D} p(d) = 1$, and associated information uncertainty $H[D] = -\sum_{d \in D} p(d) \log_2 p(d)$.

Let's ask *on average* how much information we can expect to obtain about the world by making these measurements. After measuring some $d$, the new probability distribution for the world is $p(w|d) = p(w,d)/p(d)$, i.e., conditioned on having measured $d$. The new information uncertainty is $H[W|d] = -\sum_{w \in W} p(w|d) \log_2 p(w|d)$, and if lower than $H[W]$ their difference represents the amount of information about the world obtained from having measured $d$. On average, the expected information uncertainty after measuring is thus $\sum_{d \in D} p(d) H[W|d]$. The mutual information $I[W;D]$ between the world $W$ and data $D$ is defined as the expected information gain from making the measurements: $I[W;D] = H[W] - \sum_{d \in D} p(d) H[W|d]$. From the definitions, we find that it satisfies

$$\begin{aligned}
I[W;D] &= H[W] - \sum_{d \in D} p(d) H[W|d] \\
&= -\sum_{w \in W} p(w) \log_2 p(w) + \sum_{d \in D} p(d) \sum_{w \in W} p(w|d) \log_2 p(w|d) \\
&= -\sum_{w \in W, d \in D} p(w,d) \log_2 p(w) + \sum_{w \in W, d \in D} p(w,d) \log_2 p(w,d)/p(d) \\
&= \sum_{w \in W, d \in D} p(w,d) \log_2 \frac{p(w,d)}{p(w)p(d)} \ .
\end{aligned}$$

Perhaps surprisingly, the result is symmetric in $W$ and $D$: on average we learn the same amount about the world from measuring the data, as we learn about the probability distribution of likely data from knowing about the world. If $W$ and $D$ are independent, i.e., $p(w,d) = p(w)p(d)$ for all $w \in W, d \in D$, then the argument of the logarithm is always 1 and $I[W;D] = 0$. More generally, the mutual information satisfies* $I[W;D] \geq 0$, and vanishes only when the events are independent. Note also: $I[W;D] = H[W] + H[D] - H[W,D]$, where $H[W,D]$ is the information uncertainty of the joint distribution $p(w,d)$.

* This can easily be proved using the inequality $\ln x \leq x - 1$.

Decision Trees (From Russell/Norvig Chpt 18)

First recall the definition of Supervised Learning: given a training set $(x_1, y_1)$, …, $(x_N, y_N)$ generated by some unknown function $y = f(x)$, learn some approximation $h$ to $f$ which agrees as well as possible on the training set. If $y$ takes some discrete set of values, then the problem is called *classification* (e.g., the Boolean case of Spam/non-Spam). If $y$ is a continuous variable, then estimating its functional form is called *regression*.

Inferring the structure of a decision tree is one popular way of guessing a function that takes a set of attribute values, and returns an output value as a single 'decision'. In the below, we'll consider the Boolean case in which the decision is a yes/no value. It is determined by a sequence of tests, applied to the attributes, and branching according to their values, until reaching an end point (leaf node) of the tree associated to an output value. An example would be an auto-repair manual, structured as a series of troubleshooting questions and branching according to the condition.

Another example is a Boolean decision tree, outputting a yes/no answer, for deciding whether to wait at a restaurant, with feature vector consisting of answers to attributes: Alternatives available?, Bar?, Fri/Sat?, Hungry?, Patrons? (none, some, full), Price?, Raining?, Reservation?, Type?, estimated wait time? depicted here:

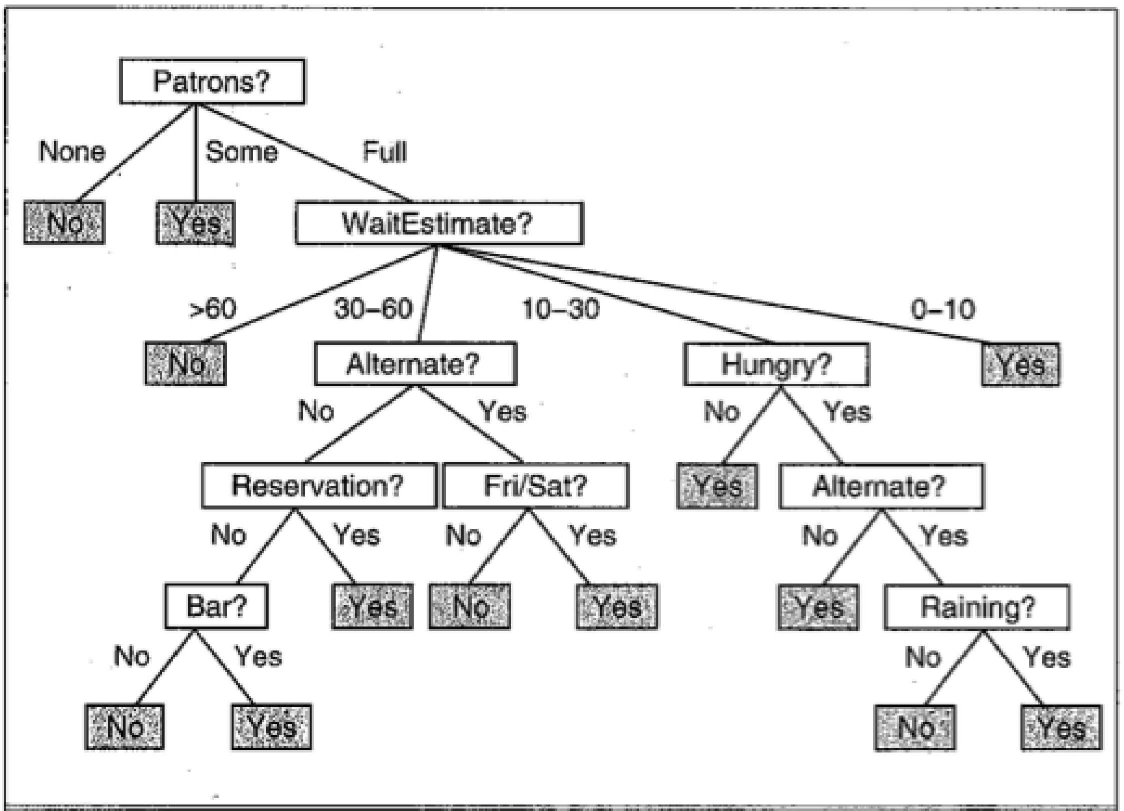
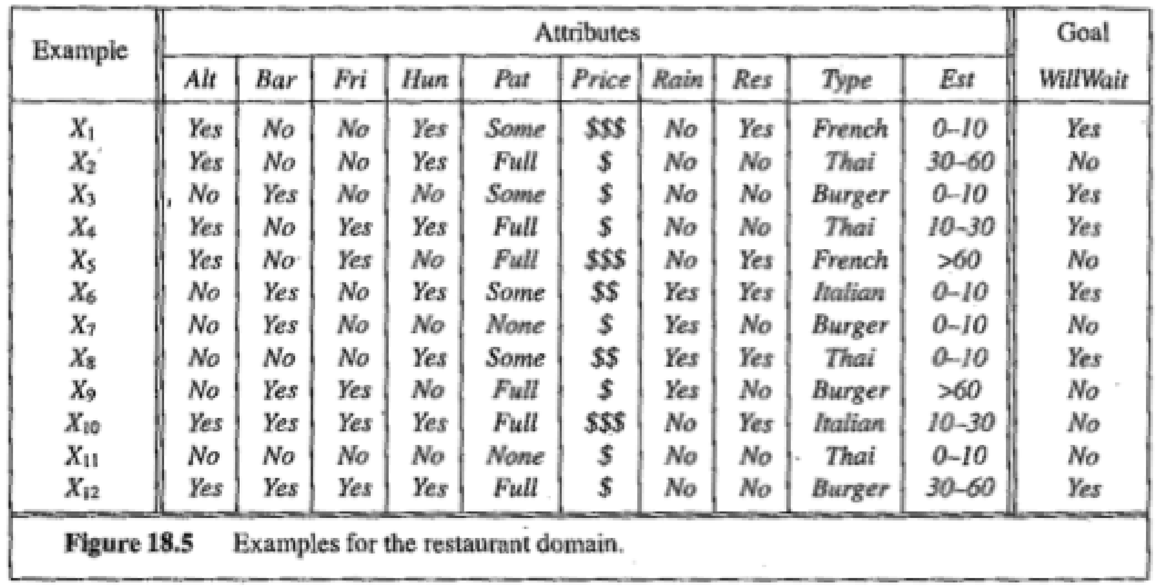


**Figure 18.4** A decision tree for deciding whether to wait for a table.

Suppose we do not know the underlying structure of this tree, but instead have just some set of examples generated by this tree, in the form of $(\vec{x},y)$ pairs:



| Example | Attributes | | | | | | | | | | Goal |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Alt* | *Bar* | *Fri* | *Hun* | *Pat* | *Price* | *Rain* | *Res* | *Type* | *Est* | *WillWait* |
| $X_1$ | Yes | No | No | Yes | Some | \$\$\$ | No | Yes | French | 0–10 | Yes |
| $X_2$ | Yes | No | No | Yes | Full | \$ | No | No | Thai | 30–60 | No |
| $X_3$ | No | Yes | No | No | Some | \$ | No | No | Burger | 0–10 | Yes |
| $X_4$ | Yes | No | Yes | Yes | Full | \$ | No | No | Thai | 10–30 | Yes |
| $X_5$ | Yes | No | Yes | No | Full | \$\$\$ | No | Yes | French | >60 | No |
| $X_6$ | No | Yes | No | Yes | Some | \$\$ | Yes | Yes | Italian | 0–10 | Yes |
| $X_7$ | No | Yes | No | No | None | \$ | Yes | No | Burger | 0–10 | No |
| $X_8$ | No | No | No | Yes | Some | \$\$ | Yes | Yes | Thai | 0–10 | Yes |
| $X_9$ | No | Yes | Yes | No | Full | \$ | Yes | No | Burger | >60 | No |
| $X_{10}$ | Yes | Yes | Yes | Yes | Full | \$\$\$ | No | Yes | Italian | 10–30 | No |
| $X_{11}$ | No | No | No | No | None | \$ | No | No | Thai | 0–10 | No |
| $X_{12}$ | Yes | Yes | Yes | Yes | Full | \$ | No | No | Burger | 30–60 | Yes |

**Figure 18.5** Examples for the restaurant domain.

The challenge is to infer a decision tree consistent with above examples. In general there will be many such trees, and the object is to find the smallest one possible, having the minimal depth (smallest number of vertical levels). This is computationally impossible in general, but a greedy algorithm based on an information theoretic quantity gives a good approximation.

The twelve examples above have six Yes and six No outcomes, and hence the information uncertainty at the outset is $B(6/12) = 1$ bit. The first step is to find the most important attribute, understood as the one that gives the most information in the Shannon sense.

Consider the Type attribute and the outcomes associated to its four values: French (1 Yes, 1 No), Italian (1 Yes, 1 No), Thai (2 Yes, 2 No), and Burger (2 Yes, 2 No). For each of the four values, there's an equal number of Yes/No values and no information is gained regardless its value.

Now consider the Patrons attribute and the outcomes for its three values: None (2 No), Some (4 Yes), Full (2 Yes, 4 No). The first two values determine the outcome and the third value gives partial information (more likely No). The latter case (Patrons = Full) can be further discriminated by examining another attribute: e.g., if Hungry is No, then WillWait is No, leaving (2 Yes, 2 No) if Hungry is Yes.

To quantify the information gain associated to the attribute Patrons, we assign to each of its values a probability proportional to the number of times it occurs in the table. Two of twelve times, it is None, and both times WillWait is No, giving a contribution of $(2/12)B(0/2) = 0$. Similarly the four times it is Some the resulting WillWait is Yes, hence a contribution of $(4/12)B(4/4)$. Overall branching according to its values leaves an expected information uncertainty of

$$\frac{2}{12}B(0/2) + \frac{4}{12}B(4/4) + \frac{6}{12}B(2/6) \ ,$$

and hence the information gain is

$$1 - \Big[\frac{2}{12}B(0/2) + \frac{4}{12}B(4/4) + \frac{6}{12}B(2/6)\Big] = 1 - \frac{1}{2}B(1/3) \approx .541 \ .$$

By contrast, the information uncertainty after examining the four possibilities for the Type attribute is $(2/12)B(1/2) + (2/12)B(1/2) + (4/12)B(2/4) + (4/12)B(2/4) = 1$ (since $B(1/2) = 1$), but we started with 1 bit of uncertainty, so the information gain is $1 - 1 = 0$.

Suppose we start with a total of $p$ positive (Yes) and $n$ negative (No) results in the table of examples, a beginning uncertainty of $B(\frac{p}{p+n})$, measuring the total info needed to reach the goal from the examples. In general, if we then test some attribute with $d$ possibilities and whose $k^{\text{th}}$ value has $p_k$ positive and $n_k$ negative results among the examples, then on that branch $B(p_k/(p_k + n_k))$ additional bits of info are required to answer the question, and that branch contributes $\frac{p_k+n_k}{p+n}B(\frac{p_k}{p_k+n_k})$ to the expected remaining information uncertainty. The total remaining information uncertainty after testing the attribute is the sum over these values, so the information gained is

$$\text{Gain} = B\Big(\frac{p}{p+n}\Big) - \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B\Big(\frac{p_k}{p_k + n_k}\Big) \ .$$

Testing all of the attributes in turn, it turns out that the information gain from examining the Patrons attribute at the outset is the largest, and hence Patrons is the best initial discriminator. Testing all the other attributes in the remaining indeterminate case Patrons=Full gives Hungry as the best discriminator at that level. In the case Hungry=yes, then the remaining four possibilities (2 Yes, 2 No) are best split by considering the Type (which recall was not at all useful at the top level), and the one remaining ambiguous case (Thai: 1 Yes, 1 No) is determined by considering whether it's Fri/Sat. The minimal depth decision tree trained on the above examples is thus as depicted here:

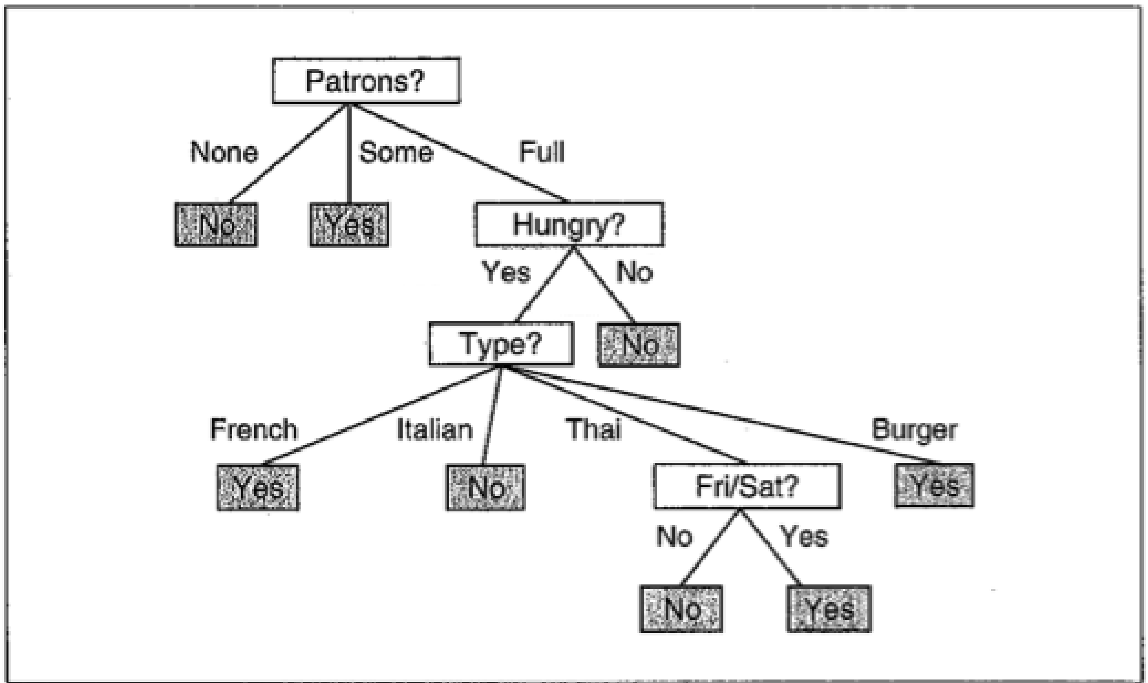


**Figure 18.8** The decision tree induced from the 12-example training set.

This tree is simpler than the initial tree used to generate the twelve examples, but agrees in result for all of them. Among other things, we see that neither the Rain nor Reservation attributes are needed to match this small set of examples. A larger set of examples drawn from the original tree might lead to inferring a more complex tree. A smaller set of examples might undetermine the tree, i.e., we could reach a decision case still ambiguous but with no remaining attributes to test. (In that case a plurality rule could be used to make the decision.) It is also possible that the examples are intrinsically inconsistent (a form of noise), in which case the learning algorithm would have to detect outliers by giving the correct result on the largest possible number of examples.