Chapter 3
Nonmetric Methods

3.1 Introduction

With nonmetric (i.e., categorical) data, we have lists of attributes as features rather than real numbers. For example, a fruit may be described as {(color=) red, (texture=), shiny, (taste=) sweet, (size=) large} or a segment of DNA as a sequence of base pairs, such as "GACTTAGATTCCA." These are discrete data, and they are conveniently addressed by decision trees, rule-based classifiers, and syntactic (grammar-based) methods.

3.2 Decision Tree Classifier

A decision tree is a simple classifier in the form of a hierarchical tree structure, which performs supervised classification using a *divide-and-conquer* strategy. It comprises a directed branching structure with a series of questions (Fig. 3.1), like the *Twenty Questions* game. The questions are placed at *decision nodes*; each tests the value of a particular attribute (feature) of the pattern (object) and provides a binary or multi-way split. The starting node is known as the *root node*, which is considered the parent of every other node. The branches correspond to the possible answers. Successive decision nodes are visited until a terminal or *leaf node* is reached, where the class (category) is read (assigned). (The decision tree is an upside-down tree, with the root at the top and the leaves at the bottom!). Classification is performed by routing from the root node until arriving at a leaf node. The tree structure is not fixed a priori but the tree grows and branches during learning depending on the complexity of the problem.

Figure 3.2 is an example of a three-level decision tree, used to decide what to do on a Saturday morning. Suppose, for example, that our parents haven't turned up and the sun is shining; the decision tree tells us to go off and play tennis. Note that the decision tree covers all eventualities: there are no values that the weather, our

G. Dougherty, *Pattern Recognition and Classification: An Introduction*,
DOI 10.1007/978-1-4614-5323-9_3,

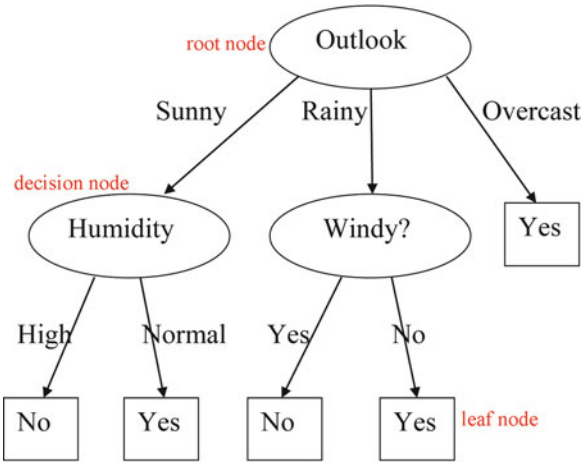

Fig. 3.1 A (two-level) decision tree for determining whether to play tennis. We have used *elliptical shapes* for the decision nodes (including the root node) and *rectangular shapes* for the leaf nodes

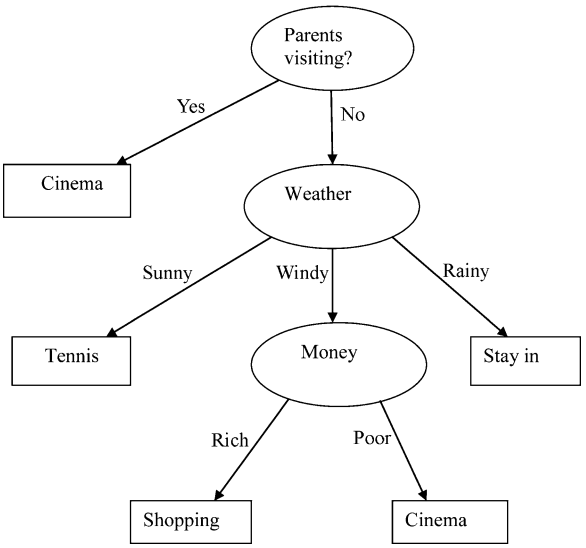

Fig. 3.2 A three-level decision tree for determining what to do on a Saturday morning

parents turning up or not, and our financial situation can take which aren't catered for in the decision tree.

Decision trees are more general than representations of decision-making processes. By rephrasing the questions, they can be applied to classification problems. An advantage of the decision tree classifier is that it can be used with nonmetric/categorical data, including nominal data with no natural ordering (although it can also be adapted to use quantitative data). Another benefit is its clear interpretability, providing a natural way to incorporate prior knowledge (and it is straightforward to convert the tests into logical expressions). Decision trees, once constructed, are very fast since they require very little computation.

The decision tree is easy to use; the more interesting question is how to construct the tree from training data (records), after having chosen a set of discriminating features. In principle, there are exponentially many decision trees that can be constructed from a given set of features. While some of the trees will be more accurate than others, finding the optimal tree is not computationally feasible. Nevertheless, a number of efficient algorithms have been developed to create or "grow" a reasonably accurate, albeit suboptimal, decision tree in a reasonable amount of time. These algorithms usually employ a *greedy* strategy that grows the tree using the *most informative* attribute (feature) at each step and does not allow backtracking. The most informative attribute will be the one which splits the set arriving at the node into the most homogeneous subsets.

3.2.1 Information, Entropy, and Impurity

Information can be thought of as the reduction of uncertainty, and informative attributes will be the ones that result in the largest reduction of uncertainty. The information content of a single message state in units of information is given by:

$$I(E) = \log \frac{1}{P(E)} = -\log P(E) \tag{3.1}$$

where $P(E)$ is the prior probability of occurrence of the message. Intuitively, the amount of information carried by a message is inversely related to the probability of its occurrence. Messages with a high probability of occurring carry little information, and conversely, messages that are least expected carry most information. If only two events are possible (0 and 1), the base of the logarithm in (3.1) is 2, and the resulting unit of information is the *bit*. If the two events are equally likely [$P_1(E) = P_2(E) = ½$] then $I(E_1) = I(E_2) = -\log_2(½) = 1$ bit, i.e., 1 bit of information is conveyed when one of the two possible equally likely events occurs. However, if the two possible events are not equally likely [for example, $P_1(E) = ¼$ and $P_2(E) = ¾$], then the information conveyed by the less common event [$I(E_1) = -\log_2(¼) = 2$] is greater than that conveyed by the more common event [$I(E_2) = -\log_2(¾) = 0.415$]. (Taking logs to the base 2 is less familiar to us, but remember that $\log_2 N = \log_{10} N/\log_{10} 2$).

Entropy is a measure of the disorder or unpredictability in a system. (It is used for discrete variables, whereas variance would be the metric for continuous variables). Given a binary (two-class) classification, C, and a set of examples, S, the class distribution at any node can be written as (p_0, p_1), where $p_1 = 1 - p_0$, and the entropy, H, of S is the sum of the information:

$$H(S) = -p_0 \log_2 p_0 - p_1 \log_2 p_1 \tag{3.2}$$

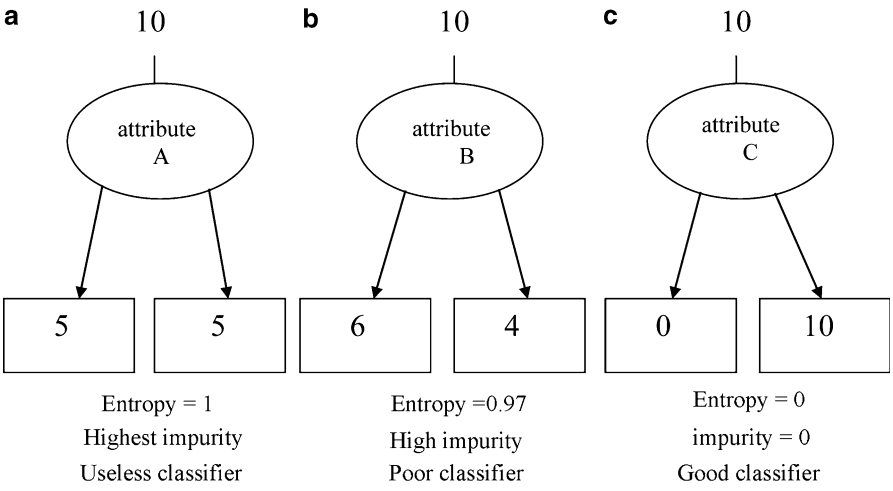

Fig. 3.3 Comparison of three decision nodes based on different attributes

If the attribute results in a classification that separates the examples into (0.5, 0.5) (as in Fig. 3.3a), the entropy (uncertainty) of that feature is at a maximum (equal to 1.0). This is *not* a useful attribute. If another attribute splits the examples into (0.6, 0.4), the entropy relative to this new classification is $-0.6 \log_2 0.6 - 0.4 \log_2 0.4 = 0.97$ (Fig. 3.3b). If all the test examples of a third attribute are of the same class [i.e., the split is (0, 1) or (1, 0) as in Fig. 3.3c], then the entropy (uncertainty) of that feature is zero and it provides good classification.

Entropy can be thought of as describing the amount of *impurity* in a set of features at a node. The smaller the degree of impurity, the more skewed the class distribution (and the more useful is the node). For example, a node with class distribution (0, 1) has zero impurity (and zero entropy) and is a good classifier; whereas a node with uniform class distribution (0.5, 0.5) has the highest impurity (and entropy = 1) and is a useless classifier.

In the general case, the target attribute can take on c different values (viz., a multi-way split) and the entropy of S relative to this c-wise classification is given by

$$H(p) = -\sum_{i=1}^{c} p_i \log_2 p_i \tag{3.3}$$

where p_i is the proportion of S belonging to class i. Note that the base of the logarithm is still 2, since we are continuing to measure the entropy in bits; and note that the maximum possible entropy relative to this attribute is $\log_2 c$.

Other impurity measures, which can be used to determine the best way to split a series of records include the *Gini impurity* and the *classification error*:

$$\text{Gini}(p) = 1 - \sum_{i} {p_i}^2 \tag{3.4}$$

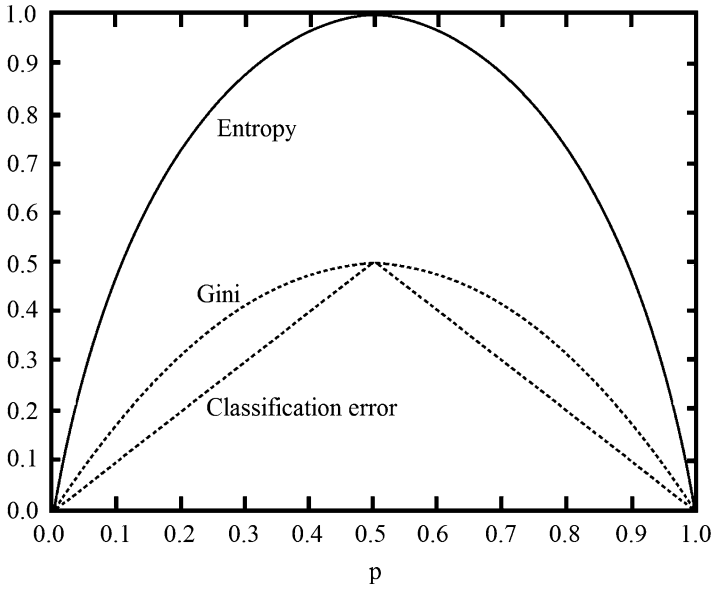

Fig. 3.4 Impurity measures for binary classification

$$\text{classification error}(p) = 1 - \max(p_i) \tag{3.5}$$

(the Gini impurity is actually the expected error rate if the class label is selected randomly from the class distribution present).

The values of these impurity measures for binary classification are shown in Fig. 3.4. All three measures attain a maximum value for a uniform distribution ($p = 0.5$), and a minimum when all the examples belong to the same class ($p = 0$ or 1). A disadvantage of the classification error is that it has a discontinuous derivative, which may be a problem when searching for an optimal decision over a continuous parameter space.

3.2.2 Information Gain

We now return to the problem of trying to determine the best attribute to choose for each decision node of the tree. The decision mode will receive a mixed bag of instances, and the best attribute will be the one that best separates them into homogeneous subsets (Fig. 3.5). The measure we will use is the *gain*, which is the expected reduction in impurity caused by partitioning the examples according to this attribute. More precisely, the gain, Gain(S, A), of an attribute A, relative to a collection of samples S, is defined as:

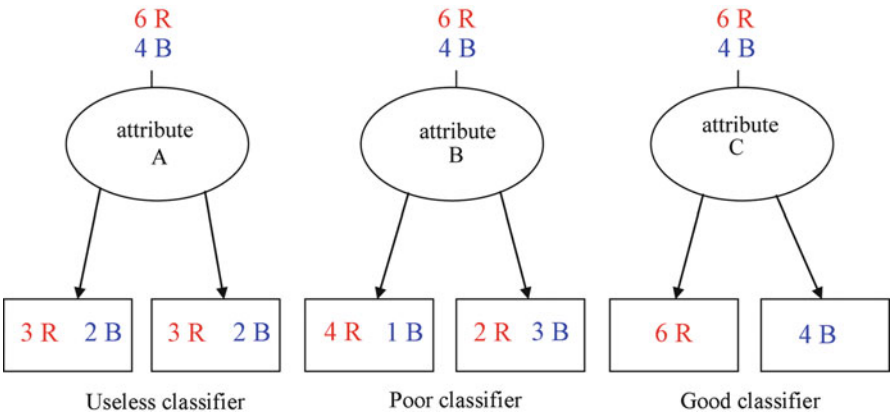

Fig. 3.5 Different attributes splitting a mixture of instances. Attribute C provides the purest split and is the best attribute to use for classification

$$\text{Gain}(S, A) = \text{Impurity}(S) - \sum_{i=1}^{k} \frac{|S_{vi}|}{|S|} \text{Impurity}(S_{vi}) \tag{3.6}$$

where the attribute A has a set of values $\{v_1, v_2, v_3 \ldots v_k\}$, and the number of examples within S with the value v_i is $|S_{vi}|$. The first term is just the impurity of the original collection S and the second term is the expected value of the impurity after S is partitioned using attribute A. The second term is simply the sum of the impurities of each subset S_{vi}, weighted by the fraction of examples that belong to S_{vi}. If entropy is used as the measure of impurity, then the gain is known as the *information gain*.

Example 3.1 Using the Gini index to find the gain

With reference to Fig. 3.6, there are two attributes, A and B, which can be used to split the data (comprising 12 instances) into smaller subsets. Before splitting (i.e., the parent node), the Gini index is 0.5 since there is an equal number of cases from both classes.

If attribute A is chosen to split the data, the Gini index for node N_1 is 0.4898 (i.e., $1 - [(4/7)^2 + (3/7)^2] = 24/49$) and for node N_2 it is 0.480 (i.e., $1 - [(2/5)^2 + (3/5)^2] = 12/25$). The weighted average for the two descendant nodes is $(7/12) \times 0.4898 + (5/12) \times 0.480 = 0.486$.

Similarly, if we use attribute B, the weighted average of the Gini index for attribute B is 0.375.

Since the subsets for attribute B have a smaller Gini index (i.e., smaller impurity), it is preferred to attribute A. [Or the gain in using attribute B is larger $(0.5 - 0.375 = 0.125)$ than the gain in using attribute A $(0.5 - 0.486 = 0.014)$.]

(continued)

(continued)

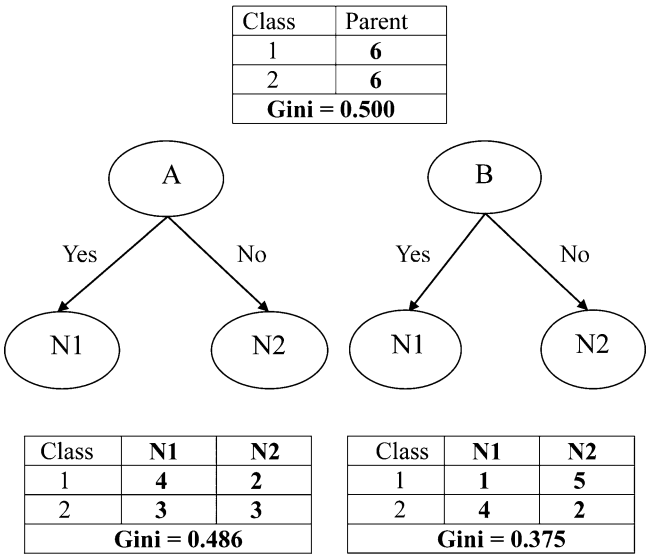

Fig. 3.6 Splitting binary attributes

This basic algorithm, the ID3 algorithm (Quinlan 1986), employs a top-down, greedy search through the space of possible decision trees. (The name ID3 was given because it was the third in a series of "interactive dichotomizer" procedures.)

Example 3.2 Using the ID3 algorithm to build a decision tree.

Suppose we want to train a decision tree using the examples (instances) in Table 3.1.

Table 3.1 Examples of decisions made over the past ten weekends

Examples	Weather	Parents visiting?	Money	Decision (category)
1	Sunny	Yes	Rich	Cinema
2	Sunny	No	Rich	Tennis
3	Windy	Yes	Rich	Cinema
4	Rainy	Yes	Poor	Cinema
5	Rainy	No	Rich	Stay in
6	Rainy	Yes	Poor	Cinema
7	Windy	No	Poor	Cinema
8	Windy	No	Rich	Shopping
9	Windy	Yes	Rich	Cinema
10	Sunny	No	Rich	Tennis

(continued)

(continued)

The first thing is to find the attribute for the root node. To do this, we need to calculate the entropy, $H(S)$, before any splitting. Using (3.3), this comes to $S = 1.571$ [viz., $-0.6 \log_2 0.6 - 0.2 \log_2 0.2 - 2 \times (0.1 \log_2 0.1)$].

Then we need to determine the values of Gain(S, parents), Gain(S, weather), and Gain(S, money), using (3.6):

$$\begin{aligned}\text{Gain}(S, \text{parents}) &= 1.571 - (|S_{\text{yes}}|/10) \times \text{Entropy}(S_{\text{yes}}) - (|S_{\text{no}}|/10) \\ &\quad \times \text{Entropy}(S_{\text{no}}) = 1.571 - (0.5) \times 0 - (0.5) \times (1.922) = 0.61\end{aligned}$$

If "parents coming?" is the node, then five instances will go down the "Yes" branch (and then will all be class "cinema," with an entropy of zero: this would be a terminal node) and five will go down the "No" branch [and they will comprise two "tennis," one "stay-in," one "cinema," and one "shopping": the entropy of this node is $-0.4 \log_2 0.4 - 3 \times (0.2 \log_2 0.2)$, which is 1.922].

$$\begin{aligned}\text{Gain}(S, \text{weather}) &= 1.571 - (|S_{\text{sun}}|/10) \times \text{Entropy}(S_{\text{sun}}) - (|S_{\text{wind}}|/10) \\ &\quad \times \text{Entropy}(S_{\text{wind}}) - (|S_{\text{rain}}|/10) \times \text{Entropy}(S_{\text{rain}}) \\ &= 1.571 - (0.3) \times (0.918) - (0.4) \times (0.8113) - (0.3) \\ &\quad \times (0.918) = 0.70\end{aligned}$$

$$\begin{aligned}\text{Gain}(S, \text{money}) &= 1.571 - (|S_{\text{rich}}|/10) \times \text{Entropy}(S_{\text{rich}}) \\ &\quad - (|S_{\text{poor}}|/10) \times \text{Entropy}(S_{\text{poor}}) \\ &= 1.571 - (0.7) \times (1.842) - (0.3) \times 0 = 0.2816\end{aligned}$$

This means that the weather attribute, with its three branches, should be the first (root) node (Fig. 3.7a).

Now we look at the branches. For the sunny branch, $S_{\text{sunny}} = \{1, 2, 10\}$. Since the classes (cinema, tennis, and tennis respectively) are not the same, we will need another decision node here (Fig. 3.7b). The same situation occurs for the other two branches (where $S_{\text{windy}} = \{3, 7, 8, 9\}$, and $S_{\text{rainy}} = \{4, 5, 6\}$).

Returning to the sunny branch, we are only interested in the three examples $\{1, 2, 10\}$ and we set S to be S_{sunny}, from which $H(S)$ turns out to be 0.918 (viz., $-0.1 \log_2 0.1 - 0.2 \log_2 0.2$), since two examples end up together (as "tennis") and one ends up on its own (as "cinema"). We now need to calculate Gain (S_{sunny}, parents) and Gain(S_{sunny}, money):

$$\begin{aligned}\text{Gain}(S_{\text{sunny}}, \text{parents}) &= 0.918 - (|S_{\text{yes}}|/|S|) \times \text{Entropy}(S_{\text{yes}}) - (|S_{\text{no}}|/|S|) \\ &\quad \times \text{Entropy}(S_{\text{no}}) \\ &= 0.918 - (1/3) \times 0 - (2/3) \times 0 = 0.918\end{aligned}$$

(continued)

(continued)

$$\begin{aligned}\text{Gain}\ (S_{\text{sunny}}, \text{money}) &= 0.918 - (|S_{\text{rich}}|/|S|) \times \text{Entropy}(S_{\text{rich}}) \\ &\quad - (|S_{\text{poor}}|/|S|) \times \text{Entropy}(S_{\text{poor}}) \\ &= 0.918 - (3/3) \times 0.918 - (0/3) \times 0 = 0\end{aligned}$$

Note that Entropy(S_{yes}) and Entropy(S_{no}) are both zero, because both S_{yes} and S_{no} contain examples which are all in the same category (cinema and tennis, respectively). Hence, the parents attribute should be chosen next. It has two branches for "Yes" and "No": the "Yes" branch contains a single example {1}, and the "No" branch contains two examples, {2, 10} which are in the same class. Hence both these branches end at leaf nodes (Fig. 3.7c).

The rest of the tree is left as an exercise for the reader!

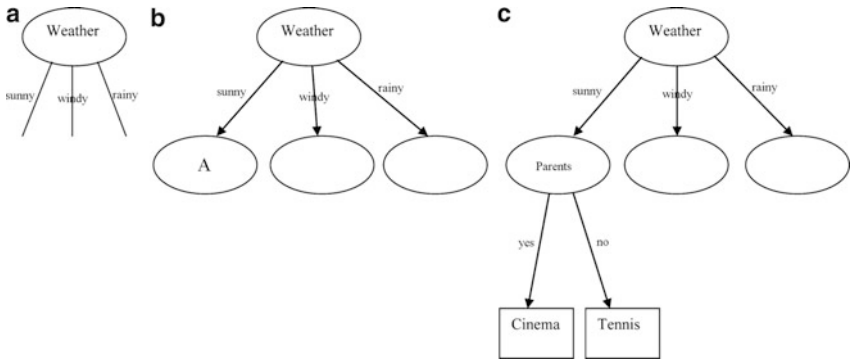

Fig. 3.7 Stages in building a decision tree

3.2.3 *Decision Tree Issues*

There are a number of issues arising with decision trees

- Decision trees partition feature space into disjoint regions, with decision boundaries that are rectilinear and parallel to the feature axes (Fig. 3.8).
  Oblique decision trees can be used to overcome this limitation by using test conditions such as $w_{11}x_1 + w_{12}x_2 + w_{10} > 0$, where x_1 and x_2 are the features, w_{11} and w_{12} are weights, and w_{10} is a *bias* or threshold value (Fig. 3.9). The decision tree is now a *linear*, *multivariate* tree. If these test conditions occur near the top of the tree, and the training set is large, then training can be slow.
- When should we stop splitting during training? If we continue splitting too far, the data will be *overfit*. In the extreme, each terminal (or leaf) node will correspond to a single training point and the full tree is merely a look-up table, which cannot be expected to generalize well to (noisy) test data. Conversely, if splitting is stopped too early, then the error on the training data is not sufficiently low and performance with test data will suffer.

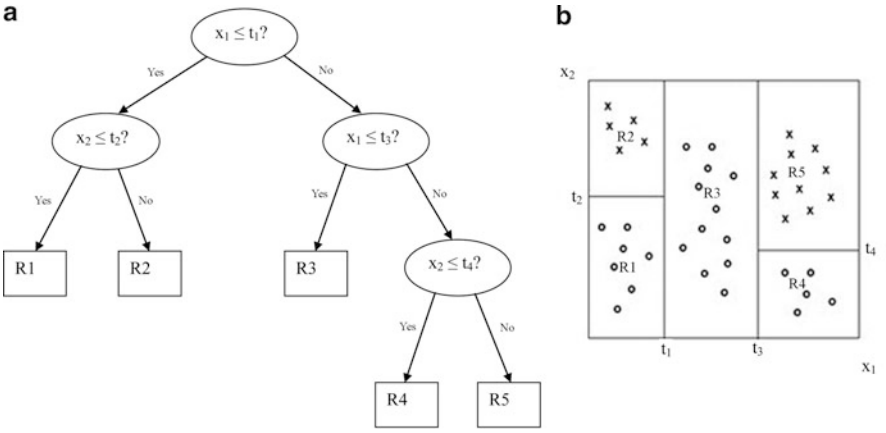

Fig. 3.8 (**a**) A decision tree and (**b**) the resulting decision boundaries in feature space

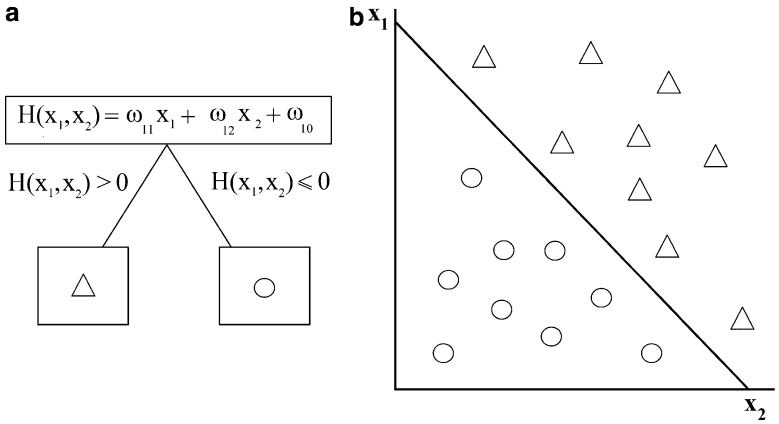

Fig. 3.9 (**a**) An oblique decision tree and (**b**) the resulting decision boundary in feature space

The training and test error rates are large when the tree is very small. This situation is known as *underfitting*: the model has yet to learn the true structure of the data. As the number of nodes in the tree increases, it will have fewer training and test errors (Fig. 3.10). However, once the tree becomes large, its test error rate begins to increase even though its training error rate continues to fall. This is known as *overfitting*, where the tree contains some nodes that accidentally fit noisy points in the training data but do not generalize to the test data.

One way to decide when to stop splitting is by using *validation* or *cross-validation*. In validation, the tree is trained using a subset of the data (e.g., 90%) with the remaining (10%) kept as a validation set. We continue splitting until the error in the validation set is minimized. (In cross-validation several independently chosen subsets are used.)

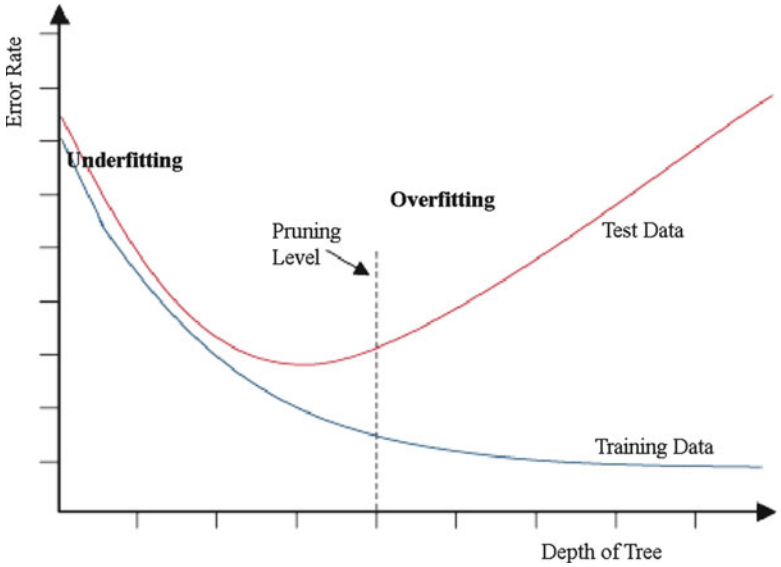

Fig. 3.10 Training and test error rates: a typical example. To avoid overfitting, the tree can be post-pruned; an appropriate pruning level is indicated

Another method is to stop splitting when the reduction in impurity falls below a certain threshold, or when a node represents less than a certain number of points (e.g., 5% of the total training set).

If the tree is grown too large, it can be *pruned* by trimming in a bottom-up fashion (Fig. 3.10). All pairs of neighboring leaf nodes (i.e., ones linked to a common parent) are considered for elimination. Any pair whose elimination results in a satisfactory (small) increase in impurity is eliminated, and the common parent node becomes a leaf node. (This node could then itself be pruned.) Pruning is incorporated in a successor to the ID3 algorithm known as C4.5 (Quinlan 1993), and in a modification of that, the J48 algorithm (Witten and Frank 2005).

- In certain cases, the available data may be missing values for some attributes. 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 the gain is to be calculated at a node in the decision tree to evaluate whether the attribute A is the best attribute to test at this decision node, and that the value $A(x)$ is unknown for one of the training samples. One strategy would be to assign it the value that is most common among training examples at this node. Alternatively, we might assign it the most common value among examples at this node that have the same classification.

  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 the particular node. For example, given a Boolean attribute A, if the node 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 the 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 incorporated in the C4.5 (Quinlan 1993) and J48 (Witten and Frank 2005) algorithms.

- When comparing binary splits with multi-way splits, the multi-way split is inherently favored. To avoid this, the gains should be normalized by dividing by $-\Sigma\, p_i \log_2 p_i$ (summed over the number of splits, k): if each attribute has the same number of cases, then $p_i = 1/k$ and the normalizing factor is $\log_2 k$. With multi-way splits, it is important to ensure that there are sufficient numbers of cases associated with each choice in order to make reliable predictions.
- With binary splits of continuous data, we have to decide the best value of the threshold at which the split is to occur, i.e., the threshold value that produces the greatest information gain. This could be achieved by histogramming the data (in which case a bin width has to be chosen), computing the Gini index for all the possible splits, and choosing the split that produces the smallest Gini index.

3.2.4 Strengths and Weaknesses

Decision trees are easy to understand, do not require much computation when performing classification, and provide a clear indication of which attributes (fields) are most important. They are most useful for categorical data but can be adapted to handle continuous data.

On the other hand, they are prone to errors when there are many classes and a relatively small number of training examples. They can be computationally expensive to train, and pruning is also computationally expensive. They tend to give rectangular decision regions, unless more computationally expensive strategies are used.

3.3 Rule-Based Classifier

In problems where classes can be characterized by general relationships, rather than just by examples (instances), it becomes attractive to build classifiers based on rules. Humans generally like explanations for most decisions, and sometimes there are legal and ethical requirements for this (e.g., rejection of a credit card application, medical diagnosis).

It is possible to extract rules from a decision tree. Each path from root to a leaf can be written down as a series of IF. . .THEN rules. For example, the left-most path in Fig. 3.1 would lead to the rule **IF** (outlook = sunny) **AND** (humidity = high) **THEN** do not play tennis. When more than one leaf is labeled with the same class, then the paths can be combined with a logical OR. It may be possible to prune the resulting rules, although they may not be able to be written back as a tree after that.

Another way is to learn the rules directly from the data. Such *rule induction* is similar to decision tree induction except that rule induction does a depth-first search and generates one rule (path) at a time, whereas tree induction goes breadth-first and generates all paths simultaneously.

Rules are learned one at a time. Each rule is a combination of conditions, which are added one at a time to minimize some criterion (such as entropy). Once a rule is grown and pruned, it is added to the *rule base* and all the training examples covered by that rule are removed from the training set: and the process continues until enough rules are generated. There is an outer loop of adding one rule at a time to the rule base and an inner loop of adding one condition at a time to the current rule. Both steps are greedy, and do not guarantee optimality; and both loops incorporate pruning for better generalization. An example of a rule-induction algorithm is *Ripper* (Cohen 1995), which is based on an earlier algorithm *Irep* (Furnkranz and Widmer 1994). It is well suited for handling datasets with imbalanced class distributions.

Rule-based classifiers give comparable performance to the decision tree classifier. They create rectilinear partitions similar to those created by (non-oblique) decision trees. Nevertheless, if the rule-based classifier allows multiple rules to be triggered for a given example, then a more complex decision boundary can be constructed.

3.4 Other Methods

Ordered sequences or strings of discrete items, as in a sequence of letters in an English word or a DNA sequence, such as "AGCTTGGCATC" (where A, G, C, and T stand for the nucleic acids adenine, guanine, cytosine, and thymine, respectively), are nominal elements. The strings can be of different lengths and there is no obvious distance metric. *String matching* involves finding whether a sub-string appears in a string for a particular *shift* of characters. An *edit distance*, based on the nearest neighbor distance (Chap. 5), can be invoked to measure the similarity or difference between two strings. Such an edit distance describes how many fundamental operations (substitution, insertion, or deletion of a character) are required to transform one string into another.

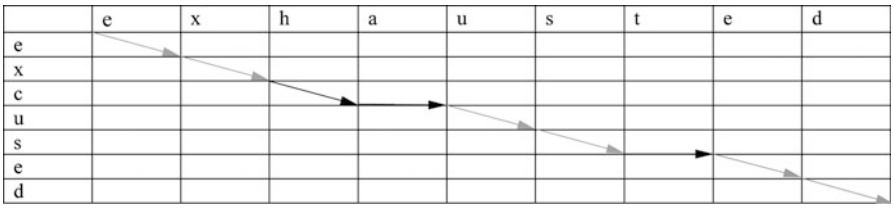

Fig. 3.11 The edit distance calculation for strings x and y can be illustrated in a table. (A *gray arrow* indicates no change. A *black diagonal arrow* indicates substitution, and a *black horizontal arrow* an insertion)

For example, the string $\mathbf{x}$ = "excused" can be transformed into the string $\mathbf{y}$ = " exhausted" using one substitution and two insertions (Fig. 3.11). If these operations are equally costly, then the edit distance is 3.

The sequence of characters may be generated according to particular structural rules, viz., grammar. An example would be valid telephone numbers, with international, national, and local codes. Often this structure is hierarchical, with "noun" and "verb" phrases. Grammatical methods can be used to provide constraints and improve accuracy. For example, an optical character recognition (OCR) system that recognizes and interprets mathematical equations based on a scanned pixel image can have particular "slots" which can be filled by only a limited set of symbols.

The string generated by a set of rules is referred to as a *sentence*; and the rules are specified by a *grammar*. In pattern recognition, we are given a sentence and a grammar and seek to determine whether the sentence was generated by the grammar. The general process is known as *parsing*. Many parsing methods depend on the model underlying the grammar. One popular such model is *finite-state machines*.

3.5 Exercises

1. Suppose that the probability of five events are $P(1) = 0.5$, and $P(2) = P(3) = P(4) = P(5) = 0.125$. Calculate the entropy. Explain in words what this means.
2. Three binary nodes, N_1, N_2, and N_3, split examples into (0, 6), (1,5), and (3,3), respectively. For each node, calculate its entropy, Gini impurity, and classification error.
3. Build a decision tree that computes the logical AND function.
4. Imagine that there are four things that you like to do in the evening: going to a pub, watching TV, going to a party, or studying (!). The choice is sometimes made for you—if you have an assignment due the next day, you need to study, if you're feeling lazy then the pub isn't for you, and if there isn't a party then you can't go to it. You are looking for a decision tree which will help you decide what to do each evening. Here is a list of everything you've done in the past 10 days.

Deadline?	Is there a party?	Lazy?	Activity
Urgent	Yes	Yes	Party
Urgent	No	Yes	Study
Near	Yes	Yes	Party
None	Yes	No	Party
None	No	Yes	Pub
None	Yes	No	Party
Near	No	No	Study
Near	No	Yes	TV
Near	Yes	Yes	Party
Urgent	No	No	Study

(The first thing to do is to work out which feature to use as the starting (root) node. For this you need to compute the entropy, and then find out which feature has the maximal information gain).

5. Write out the steps in the ID3 algorithm in pseudo code.
6. Consider the training data from the alphabet $A = \{a, b, c\}$:

$\omega 1$	$\omega 2$	$\omega 3$
aabbc	Bccba	caaaa
ababcc	Bbbca	cbcaab
babbcc	cbbaaaa	baaca

Use the edit distance to classify each of the following strings [if there are ambiguities in the classification, state which two (or all three) categories are candidates]: "abacc," "abca," "ccbba," "bbaaac"

References

Cohen, W.: Fast effective rule induction. In: Prieditis, A., Russell, S.J. (eds.) Twelfth International Conference on Machine Learning, pp. 115–123. Morgan Kaufmann, San Mateo, CA (1995)

Furnkranz, J., Widmer, G.: Incremental reduced error pruning. In: Cohen, W., Hirsch, H. (eds.) Eleventh International Conference on Machine Learning, pp. 70–77. Morgan Kaufmann, San Mateo, CA (1994)

Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**, 81–106 (1986)

Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA (1993)

Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Mateo, CA (2005)