

Decision Trees

Basic Process

Decision trees are a popular class of machine learning methods. Taking binary classification as an example, we can regard the task as deciding the answer to the question “Is this instance positive?” As the name suggests, a decision tree makes decisions based on tree structures, which is also a common decision-making mechanism used by humans. For example, in order to answer the question “Is this watermelon ripe?” we usually go through a series of judgments or sub-decisions: we first consider “What is the color?” If it is green then “What is the shape of root?” If it is curly then “What is the knocking sound?” Finally, based on the observations, we decide whether the watermelon is ripe or not. Such a decision process is illustrated in ■ Figure 4.1.

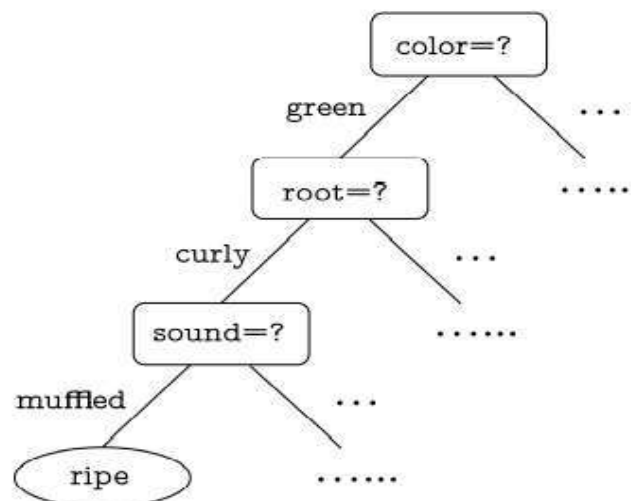


Fig. 4.1 A decision tree of the watermelon problem

The conclusions at the end of the decision process correspond to the possible classifications, e.g., ripe or unripe. Every question asked in the decision process is a test on one feature, e.g., color =? or root =?. Every test leads to either the conclusion or an additional test conditioned on the current answer. For example, if the current decision is color = green, the next test root =? considers only green watermelons.

Typically, a decision tree consists of one root node, multiple internal nodes, and multiple leaf nodes. The leaf nodes correspond to the decision outcomes, and every other node corresponds to a feature test. The samples in each node are divided into child nodes according to the splitting results of features. Each path from the root node to the leaf node is a decision sequence. The goal is to produce a tree that can generalize to predict unseen samples. The construction of decision trees follows the *divide-and-conquer* strategy, as shown in ■ Algorithm 4.1.

Algorithm 4.1 Decision Tree Learning

Input: Training set $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;

Feature set $A = \{a_1, a_2, \dots, a_d\}$.

Process: Function TreeGenerate(D, A)

<pre> 1: Generate node i; 2: if All samples in D belong to the same class C then 3: Mark node i as a class C leaf node; return 4: end if 5: if $A = \emptyset$ OR all samples in D take the same value on A then 6: Mark node i as a leaf node, and its class label is the majority class in D; return 7: end if 8: Select the optimal splitting feature a_* from A; 9: for each value a_*^v in a_* do 10: Generate a branch for node i; Let D_v be the subset of samples taking value a_*^v on a_*; 11: if D_v is empty then 12: Mark this child node as a leaf node, and label it with the major- ity class in D; return 13: else 14: Use TreeGenerate($D_v, A \setminus \{a_*\}$) as the child node. 15: end if 16: end for </pre>	<p>Recursive return, case (1).</p> <p>Recursive return, case (2).</p> <p>We will discuss the optimal split selection in the next section.</p> <p>Recursive return, case (3).</p> <p>Exclude a_* from A.</p>
--	---

Output: A decision tree with root node i .

As shown in ■ Algorithm 4.1, the tree is generated recursively, and the recursion stops in any of the following three cases: (1) all samples in the current node belong to the same class, that is, no further splitting is needed; (2) the current feature set is empty, or all samples have the same feature values, that is, not splittable; (3) there is no sample in the current node, that is, not splittable.

In case (2), we mark the current node as a leaf node and set its label to the majority class of its samples. In case (3), we mark the current node as a leaf node but set its label to the majority class of the samples in its parent node. Note that the two cases are different: case (2) uses the posterior probability of the current node, whereas case (3) uses the class probability of the parent node as the prior probability of the current node.

Split Selection

The core of the decision tree learning algorithm is the line 8 of **Algorithm 4.1**, that is, selecting the optimal splitting feature. Generally speaking, as the splitting process proceeds, we wish more samples within each node to belong to a single class, that is, increasing the *purity* of each node.

2.1 Information Gain

One of the most commonly used measures for purity is *information entropy*, or simply *entropy*. Let p_k denotes the proportion of the k th class in the current data set D , where $k = 1, 2, \dots, |\mathcal{Y}|$. Then, the entropy is defined as

$$\text{Ent}(D) = - \sum_{k=1}^{|\mathcal{Y}|} p_k \log_2 p_k. \quad (4.1)$$

The lower the $\text{Ent}(D)$, the higher the purity of D .

Suppose that the discrete feature a has V possible values $\{a^1, a^2, \dots, a^V\}$. Then, splitting the data set D by feature a produces V child nodes, where the v th child node D^v includes all samples in D taking the value a^v for feature a . Then, the entropy of D^v can be calculated using (4.1). Since there are different numbers of samples in the child nodes, a weight $|D^v| / |D|$ is assigned to reflect the importance of each node, that is, the greater the number of samples, the greater the impact of the branch node. Then, the *information gain* of splitting the data set D with feature a is calculated as

$$\text{Gain}(D, a) = \text{Ent}(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} \text{Ent}(D^v).$$

In general, the higher the information gain, the more purity improvement we can expect by splitting D with feature a . Therefore, information gain can be used for split selection, that is, using $a_* = \arg \max_{a \in A} \text{Gain}(D, a)$ as the splitting feature on the line 8

of ■ Algorithm 4.1. The well-known decision tree algorithm ID3 Quinlan (1986) takes information gain as the guideline for selecting the splitting features.

Let us see a more concrete example with the watermelon data set 2.0 in ■ Table 4.1. This data set includes 17 training samples, which are used to train a decision tree classifier for predicting the ripeness of uncut watermelons, where $|\mathcal{Y}| = 2$. In the beginning, the root node includes all samples in D , where $p_1 = \frac{8}{17}$ of them are positive and $p_2 = \frac{9}{17}$ of them are negative. According to (4.1), the entropy of the root node is

$$\text{Ent}(D) = - \sum_{k=1}^2 p_k \log_2 p_k = - \left(\frac{8}{17} \log_2 \frac{8}{17} + \frac{9}{17} \log_2 \frac{9}{17} \right) = 0.998.$$

■ **Tab. 4.1** The watermelon data set 2.0

ID	color	root	sound	texture	umbilicus	surface	ripe
1	green	curly	muffled	clear	hollow	hard	true
2	dark	curly	dull	clear	hollow	hard	true
3	dark	curly	muffled	clear	hollow	hard	true
4	green	curly	dull	clear	hollow	hard	true
5	light	curly	muffled	clear	hollow	hard	true
6	green	slightly curly	muffled	clear	slightly hollow	soft	true
7	dark	slightly curly	muffled	slightly blurry	slightly hollow	soft	true
8	dark	slightly curly	muffled	clear	slightly hollow	hard	true
9	dark	slightly curly	dull	slightly blurry	slightly hollow	hard	false
10	green	straight	crisp	clear	flat	soft	false
11	light	straight	crisp	blurry	flat	hard	false
12	light	curly	muffled	blurry	flat	soft	false
13	green	slightly curly	muffled	slightly blurry	hollow	hard	false
14	light	slightly curly	dull	slightly blurry	hollow	hard	false
15	dark	slightly curly	muffled	clear	slightly hollow	soft	false
16	light	curly	muffled	blurry	flat	hard	false
17	green	curly	dull	slightly blurry	slightly hollow	hard	false

Then, we need to calculate the information gain of each feature in the current feature set {color, root, sound, texture, umbilicus, surface}. Suppose that we have selected color, which has three possible values {green, dark, light}. If D is split by color, then there are three subsets: D^1 (color = green), D^2 (color = dark), and D^3 (color = light).

Subset D^1 includes six samples {1, 4, 6, 10, 13, 17}, in which $p_1 = \frac{3}{6}$ of them are positive and $p_2 = \frac{3}{6}$ of them are negative. Subset D^2 includes six samples {2, 3, 7, 8, 9, 15}, in which $p_1 = \frac{4}{6}$ of them are positive and $p_2 = \frac{2}{6}$ of them are negative. Subset D^3 includes five samples {5, 11, 12, 14, 16}, in which $p_1 = \frac{1}{5}$ of them are positive and $p_2 = \frac{4}{5}$ of them are negative. According to (4.1), the entropy of the three child nodes are

$$\text{Ent}(D^1) = - \left(\frac{3}{6} \log_2 \frac{3}{6} + \frac{3}{6} \log_2 \frac{3}{6} \right) = 1.000,$$

$$\text{Ent}(D^2) = - \left(\frac{4}{6} \log_2 \frac{4}{6} + \frac{2}{6} \log_2 \frac{2}{6} \right) = 0.918,$$

$$\text{Ent}(D^3) = - \left(\frac{1}{5} \log_2 \frac{1}{5} + \frac{4}{5} \log_2 \frac{4}{5} \right) = 0.722.$$

Then, we use (4.2) to calculate the information gain of splitting by color as

$$\begin{aligned} \text{Gain}(D, \text{color}) &= \text{Ent}(D) - \sum_{v=1}^3 \frac{|D^v|}{|D|} \text{Ent}(D^v) \\ &= 0.998 - \left(\frac{6}{17} \times 1.000 + \frac{6}{17} \times 0.918 + \frac{5}{17} \times 0.722 \right) \\ &= 0.109. \end{aligned}$$

Similarly, we calculate the information gain of other features:

$$\begin{aligned} \text{Gain}(D, \text{root}) &= 0.143; & \text{Gain}(D, \text{sound}) &= 0.141; \\ \text{Gain}(D, \text{texture}) &= 0.381; & \text{Gain}(D, \text{umbilicus}) &= 0.289; \\ \text{Gain}(D, \text{surface}) &= 0.006. \end{aligned}$$

Since splitting by texture gives the highest information gain, it is chosen as the splitting feature. ■ Figure 4.2 shows the result of splitting the root node by texture.

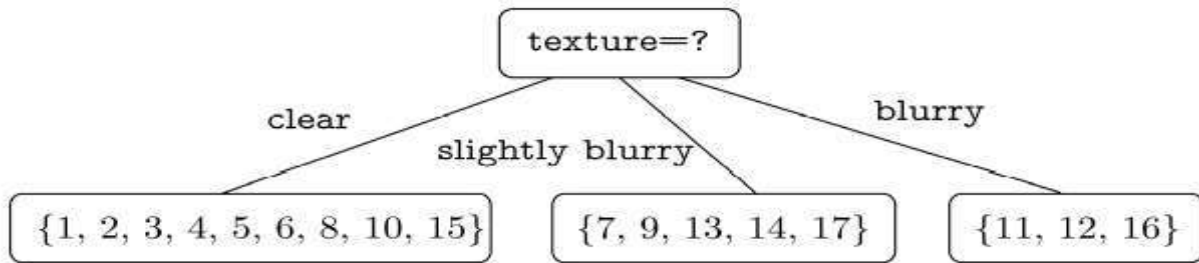


Fig. 4.2 Splitting the root node by texture

Then, each child node is further split by the decision tree algorithm. For example, the first child node (i.e., texture = clear) includes nine samples: $D^1 = \{1, 2, 3, 4, 5, 6, 8, 10, 15\}$, and the available feature set is {color, root, sound, umbilicus, surface}. We calculate the information gains of these candidate features on D^1 :

$$\begin{aligned} \text{Gain}(D^1, \text{color}) &= 0.043; & \text{Gain}(D^1, \text{root}) &= 0.458; \\ \text{Gain}(D^1, \text{sound}) &= 0.331; & \text{Gain}(D^1, \text{umbilicus}) &= 0.458; \\ \text{Gain}(D^1, \text{surface}) &= 0.458. \end{aligned}$$

Since root, umbilicus, and surface lead to the highest information gains, any of them can be chosen as the splitting feature. Repeating this process for every node, we can obtain the final decision tree, as shown in ■ Figure 4.3.

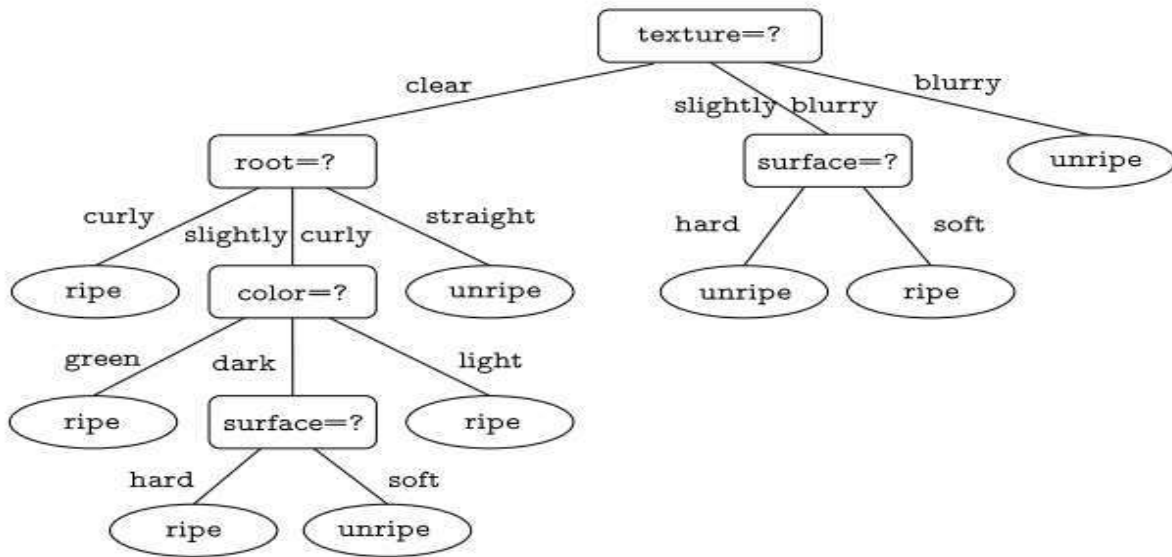


Fig. 4.3 The information gain-based decision tree generated from ■ Table 4.1

2.2 Gain Ratio

The process described above intentionally ignored the column ID. If we consider ID as a candidate splitting feature, then, from (4.2), we know its information gain is 0.998, which is much higher than that of any other features. This is reasonable since ID produces 17 child nodes, and each node has only a single sample with maximum purity. However, such a decision tree does not have generalization ability and cannot effectively predict new samples.

It turns out that the information gain criterion is biased toward features with more possible values. To reduce this bias, the renowned decision tree algorithm C4.5 (Quinlan 1993) employs *gain ratio* to select features instead of employing information gain. Using a notation similar to (4.2), the gain ratio of feature a is defined as

$$\text{Gain_ratio}(D, a) = \frac{\text{Gain}(D, a)}{\text{IV}(a)}, \quad (4.3)$$

where

$$\text{IV}(a) = - \sum_{v=1}^V \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|} \quad (4.4)$$

is called the *intrinsic value* of feature a (Quinlan 1993). $\text{IV}(a)$ is large when feature a has many possible values (i.e., large V). Taking the watermelon data set 2.0 as an example, we have: $\text{IV}(\text{surface}) = 0.8/4$ ($V = 2$), $\text{IV}(\text{color}) = 1.580$ ($V = 3$), and $\text{IV}(\text{ID}) = 4.088$ ($V = 17$).

It should be noted that, in contrast to information gain, the gain ratio is biased toward features with fewer possible values. For this reason, the C4.5 algorithm does not use gain ratio directly for selecting the splitting feature, but uses a heuristic method (Quinlan 1993): selecting the feature with the highest gain ratio from the set of candidate features with an information gain above the average.

2.3 Gini Index

CART Breiman et al. (1984) employs the *Gini index* for selecting the splitting feature. Using a notation similar to (4.1), the Gini value of data set D is defined as

$$\begin{aligned} \text{Gini}(D) &= \sum_{k=1}^{|D|} \sum_{k' \neq k} p_k p_{k'} \\ &= 1 - \sum_{k=1}^{|D|} p_k^2. \end{aligned} \quad (4.5)$$

Intuitively, $\text{Gini}(D)$ represents the likelihood of two samples we randomly selected from data set D belonging to different classes. The lower the $\text{Gini}(D)$, the higher the purity of data set D .

Using a notation similar to (4.2), the Gini index of feature a is defined as

$$\text{Gini_index}(D, a) = \sum_{v=1}^V \frac{|D^v|}{|D|} \text{Gini}(D^v). \quad (4.6)$$

Given a candidate feature set A , we select the feature with the lowest Gini index as the splitting feature, that is, $a_* = \arg \min_{a \in A} \text{Gini_index}(D, a)$.

3 Pruning

Pruning is the primary strategy of decision tree learning algorithms to deal with overfitting. To correctly classify the training samples, the learner repeats the split procedure. However, if there are too many branches, then the learner may be misled by the peculiarities of the training samples and incorrectly consider them as the underlying truth. Hence, we can prune some of the branches to reduce the risk of overfitting.

The general pruning strategies include *pre-pruning* and *post-pruning* (Quinlan 1993). Pre-pruning evaluates the improvement of the generalization ability of each split and cancels a split if the improvement is small, that is, the node is marked as a leaf node. In contrast, post-pruning re-examines the non-leaf

nodes of a fully grown decision tree, and a node is replaced with a leaf node if the replacement leads to improved generalization ability.

How do we know if the generalization ability has been improved? We can use the performance evaluation methods

For example, we can use the hold-out method to reserve part of the data as a validation set for performance evaluation. Given the watermelon data set 2.0 in ■ Table 4.1, suppose the samples are randomly partitioned into a training set {1, 2, 3, 6, 7, 10, 14, 15, 16, 17} and a validation set {4, 5, 8, 9, 11, 12, 13}, as shown in ■ Table 4.2.

■ **Tab. 4.2** Splitting the watermelon data set 2.0 into a training set (above the double dividing line) and a validation set (below the double dividing line)

ID	color	root	sound	texture	umbilicus	surface	ripe
1	green	curly	muffled	clear	hollow	hard	true
2	dark	curly	dull	clear	hollow	hard	true
3	dark	curly	muffled	clear	hollow	hard	true
6	green	slightly curly	muffled	clear	slightly hollow	soft	true
7	dark	slightly curly	muffled	slightly blurry	slightly hollow	soft	true
10	green	straight	crisp	clear	flat	soft	false
14	light	slightly curly	dull	slightly blurry	hollow	hard	false
15	dark	slightly curly	muffled	clear	slightly hollow	soft	false
16	light	curly	muffled	blurry	flat	hard	false
17	green	curly	dull	slightly blurry	slightly hollow	hard	false
ID	color	root	sound	texture	umbilicus	surface	ripe
4	green	curly	dull	clear	hollow	hard	true
5	light	curly	muffled	clear	hollow	hard	true
8	dark	slightly curly	muffled	clear	slightly hollow	hard	true
9	dark	slightly curly	dull	slightly blurry	slightly hollow	hard	false
11	light	straight	crisp	blurry	flat	hard	false
12	light	curly	muffled	blurry	flat	soft	false
13	green	slightly curly	muffled	slightly blurry	hollow	hard	false

Suppose we use the information gain criterion described in Sect. 4.2.1 for deciding the splitting features, then ■ Figure 4.4 shows the decision tree trained on the data set in ■ Table 4.2. For ease of discussion, we numbered some nodes in the figures.

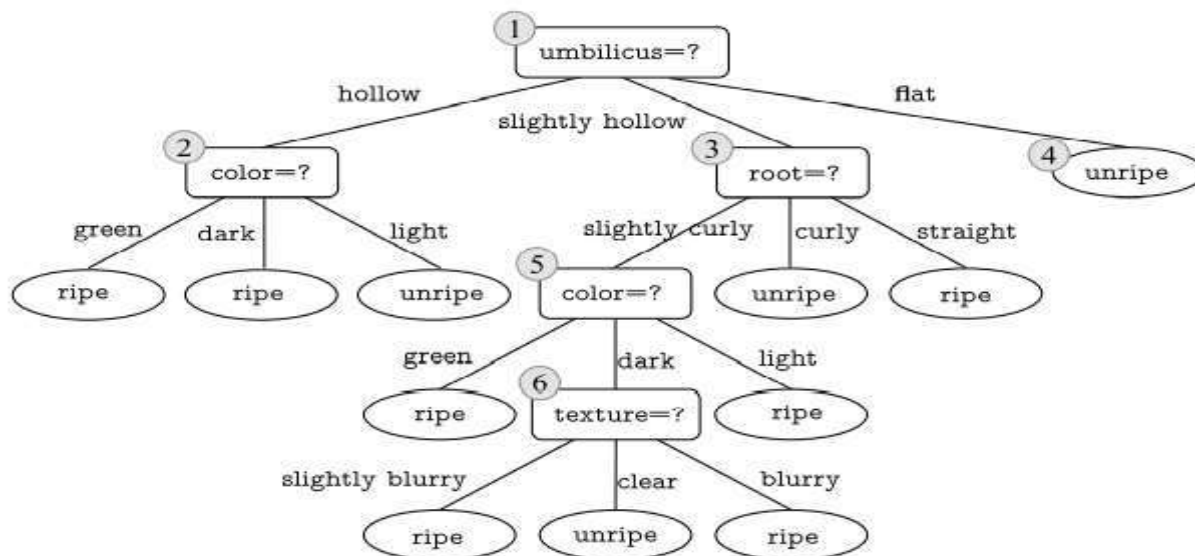


Fig. 4.4 The unpruned decision tree generated from ■ Table 4.2

3.1 Pre-pruning

Let us take a look at pre-pruning first. According to the information gain criterion, umbilicus should be chosen to split the training set into three branches, as shown in ■ Figure 4.5. However, shall we proceed with this split? Pre-pruning decides by comparing the generalization abilities before and after splitting.

Prior to splitting, all samples are in the root node. When no splitting is performed, this node is marked as a leaf node according to line 6 of ■ Algorithm 4.1, and its label is set to the majority class (i.e., ripe). By evaluating this single-node decision tree using the validation set in ■ Table 4.2, we have the samples {4, 5, 8} correctly classified and the other four samples misclassified. Then, the validation accuracy is $\frac{3}{7} \times 100\% = 42.9\%$.

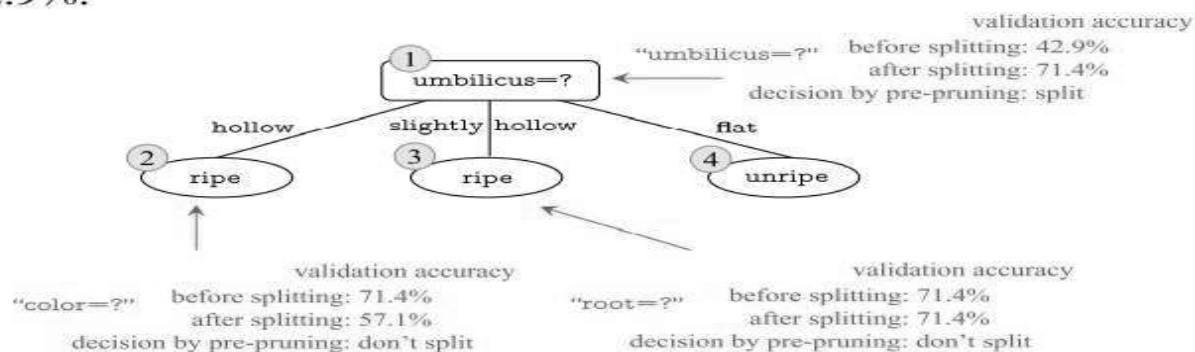


Fig. 4.5 The pre-pruned decision tree generated from ■ Table 4.2

After splitting the root node by umbilicus, the samples are placed into three child nodes, as shown in ■ Figure 4.5: node ② with the samples {1, 2, 3, 14}, node ③ with the samples {6, 7, 15, 17}, and node ④ with the samples {10, 16}. We mark these 3 nodes as leaf nodes and set the labels to the majority classes, that is, ② is ripe, ③ is ripe, and ④ is unripe. Then, the validation accuracy is $\frac{5}{7} \times 100\% = 71.4\% > 42.9\%$. Since the validation accuracy is improved, the splitting using umbilicus is adopted.

After that, the decision tree algorithm moves on to split node ②, and color is chosen based on the information gain criterion. However, since the sample {5} in the validation set is misclassified, the validation accuracy drops to 57.1%. Hence, the pre-pruning strategy stops splitting node ②. For node ③, the best feature to split on is root. However, since the validation accuracy after splitting remains the same as 71.4%, pre-pruning strategy stops splitting node ③. For node ④, no splitting is needed since all samples belong to the same class.

Finally, the pre-pruning decision tree constructed based on the data in ■ Table 4.2 is given in ■ Figure 4.5, and its validation accuracy is 71.4%. Because there is only one splitting, such a decision tree is also called a *decision stump*.

By comparing ■ Figures 4.5 and ■ 4.4, we can see that applying pre-pruning reduces the branches of the decision tree, which reduces not only the risk of overfitting but also the computational cost of training and testing. On the other hand, although some branches are prevented by pre-pruning due to little or even negative improvement on generalization ability, it is still possible that their subsequent splits can lead to significant improvement. These branches are pruned due to the greedy nature of pre-pruning, and it may introduce the risk of underfitting.

3.2 Post-pruning

Post-pruning allows a decision tree to grow into a complete tree, e.g., ■ Figure 4.4 shows a fully grown decision tree based on data in ■ Table 4.2. The validation accuracy of this decision tree is 42.9%.

In ■ Figure 4.4, node ⑥ is the first one examined by post-pruning. If the subtree led by node ⑥ is pruned and replaced with a leaf node, then it includes the samples {7, 15} and its label is set to the majority class ripe. Since the validation accuracy increases to 57.1%, the pruning is performed, resulting in the decision tree, and the result is shown in ■ Figure 4.6.

Next, post-pruning examines node ⑤. If the subtree led by node ⑤ is replaced by a leaf node, then it includes the samples {6, 7, 15} and its label is set to the majority class ripe. Since the validation accuracy remains at 57.1%, no pruning is performed.

If the subtree led by node ② is replaced by a leaf node, then it includes the samples {1, 2, 3, 14} and its label is set to the majority class ripe. Since the validation accuracy increases to 71.4%, the pruning is performed.

For nodes ③ and ①, replacing them as leaf nodes gives the validation accuracies 71.4% and 42.9%, respectively. Since there is no improvement in both cases, the nodes remain unchanged.

Finally, the post-pruning decision tree constructed using data in ■ Table 4.2 is given in ■ Figure 4.6, and its validation accuracy is 71.4%.

By comparing ■ Figs. 4.6 and ■ 4.5, we can see that post-pruning keeps more branches than pre-pruning. In general, post-pruning is less prone to underfitting and leads to better generalization ability compared to pre-pruning. However, the training time of post-pruning is much longer since it takes a bottom-up strategy to examine every non-leaf node in a completely grown decision tree.

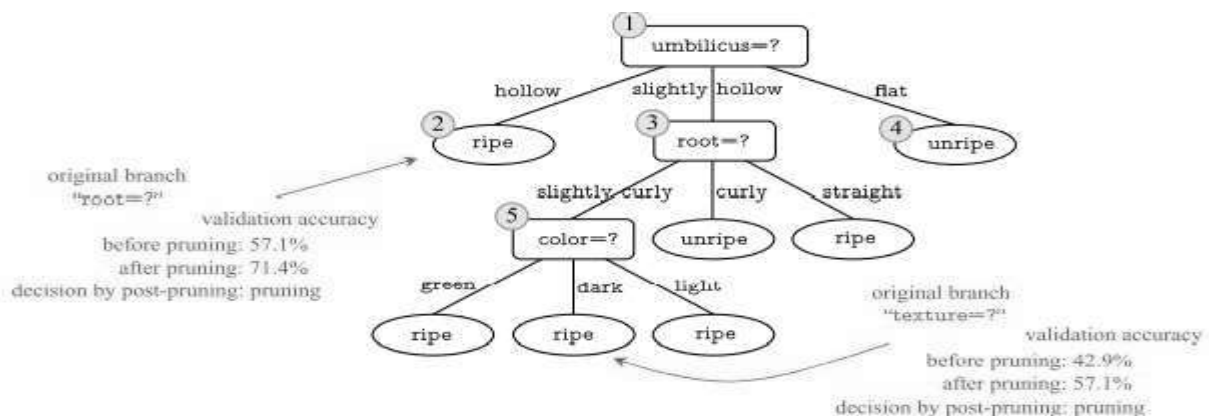


Fig. 4.6 The post-pruned decision tree generated from ■ Table 4.2

4 Continuous and Missing Values

4.1 Handling Continuous Values

Our discussions so far are limited to discrete features. However, since continuous features are also common in practice, it is necessary to know how to incorporate continuous features into decision trees.

We cannot directly split nodes with continuous features since their values are infinite. The discretization techniques come in handy in such cases. The most straightforward discretization strategy is bi-partition, which is used by C4.5 decision tree (Quinlan 1993).

Given a data set D and a continuous feature a , suppose n values of a are observed in D , and we sort these values in ascending order, denoted by $\{a^1, a^2, \dots, a^n\}$. With a split point t , D is partitioned into the subsets D_t^- and D_t^+ , where D_t^- includes the samples with the value of a not greater than t , and D_t^+ includes the samples with the value of a greater than t . For adjacent feature values a^i and a^{i+1} , the partitions are identical for choosing any t in the interval $[a^i, a^{i+1})$. As a result, for continuous feature a , there are $n - 1$ elements in the following set of candidate split points:

$$T_a = \left\{ \frac{a^i + a^{i+1}}{2} \mid 1 \leq i \leq n - 1 \right\}, \quad (4.7)$$

where the midpoint $\frac{a^i + a^{i+1}}{2}$ is used as the candidate split point for the interval $[a^i, a^{i+1})$. Then, the split points are examined in the same way as discrete features, and the optimal split points are selected for splitting nodes. For example, we can modify (4.2) as

$$\begin{aligned} \text{Gain}(D, a) &= \max_{t \in T_a} \text{Gain}(D, a, t) \\ &= \max_{t \in T_a} \text{Ent}(D) - \sum_{\lambda \in \{-, +\}} \frac{|D_t^\lambda|}{|D|} \text{Ent}(D_t^\lambda), \end{aligned} \quad (4.8)$$

where $\text{Gain}(D, a, t)$ is the information gain of bi-partitioning D by t , and the split point with the largest $\text{Gain}(D, a, t)$ is selected.

For illustration, we create the watermelon data set 3.0 in ■ Table 4.3 by adding two continuous features density and sugar to the watermelon data set 2.0. Now, we build a decision tree using this new data set.

■ **Tab. 4.3** The watermelon data set 3.0

ID	color	root	sound	texture	umbilicus	surface	density	sugar	ripe
1	green	curly	muffled	clear	hollow	hard	0.697	0.460	true
2	dark	curly	dull	clear	hollow	hard	0.774	0.376	true
3	dark	curly	muffled	clear	hollow	hard	0.634	0.264	true
4	green	curly	dull	clear	hollow	hard	0.608	0.318	true
5	light	curly	muffled	clear	hollow	hard	0.556	0.215	true
6	green	slightly curly	muffled	clear	slightly hollow	soft	0.403	0.237	true
7	dark	slightly curly	muffled	slightly blurry	slightly hollow	soft	0.481	0.149	true
8	dark	slightly curly	muffled	clear	slightly hollow	hard	0.437	0.211	true
9	dark	slightly curly	dull	slightly blurry	slightly hollow	hard	0.666	0.091	false
10	green	straight	crisp	clear	flat	soft	0.243	0.267	false
11	light	straight	crisp	blurry	flat	hard	0.245	0.057	false
12	light	curly	muffled	blurry	flat	soft	0.343	0.099	false
13	green	slightly curly	muffled	slightly blurry	hollow	hard	0.639	0.161	false
14	light	slightly curly	dull	slightly blurry	hollow	hard	0.657	0.198	false
15	dark	slightly curly	muffled	clear	slightly hollow	soft	0.360	0.370	false
16	light	curly	muffled	blurry	flat	hard	0.593	0.042	false
17	green	curly	dull	slightly blurry	slightly hollow	hard	0.719	0.103	false

At the beginning, all 17 training samples have different density values. According to (4.7), the candidate split point set includes 16 values: $T_{\text{density}} = \{0.244, 0.294, 0.351, 0.381, 0.420, 0.459, 0.518, 0.574, 0.600, 0.621, 0.636, 0.648, 0.661, 0.681, 0.708, 0.746\}$. According to (4.8), the information gain of density is 0.262, and the corresponding split point is 0.381.

For the feature sugar, its candidate split point set includes 16 values: $T_{\text{sugar}} = \{0.049, 0.074, 0.095, 0.101, 0.126, 0.155, 0.179, 0.204, 0.213, 0.226, 0.250, 0.265, 0.292, 0.344, 0.373, 0.418\}$. Similarly, the information gain of sugar is 0.349 according to (4.8), and the corresponding split point is 0.126.

Combining the results from Sect. 4.2.1, the information gains of features in ■ Table 4.3 are

$$\begin{aligned}
 \text{Gain}(D, \text{color}) &= 0.109; & \text{Gain}(D, \text{root}) &= 0.143; \\
 \text{Gain}(D, \text{sound}) &= 0.141; & \text{Gain}(D, \text{texture}) &= 0.381; \\
 \text{Gain}(D, \text{umbilicus}) &= 0.289; & \text{Gain}(D, \text{surface}) &= 0.006; \\
 \text{Gain}(D, \text{density}) &= 0.262; & \text{Gain}(D, \text{sugar}) &= 0.349.
 \end{aligned}$$

Since splitting by **texture** has the largest information gain, it is selected as the splitting feature for the root node. The splitting process proceeds recursively, and the final decision tree is shown in ■ Figure 4.7.

Unlike discrete features, a continuous feature can be used as a splitting feature more than once in a decision sequence.

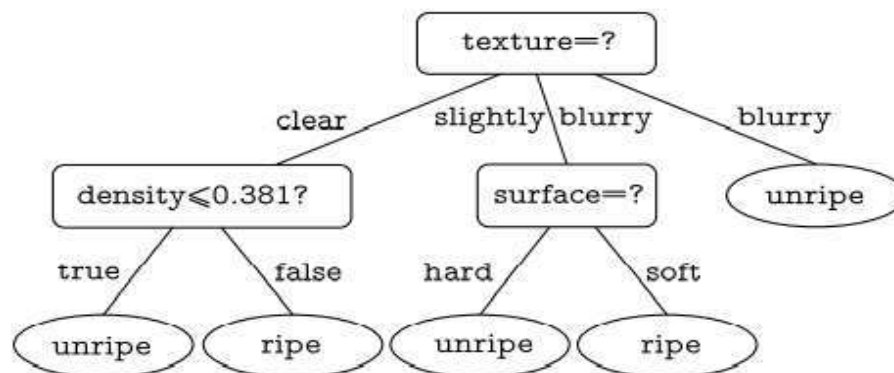


Fig. 4.7 The information gain-based decision tree generated from ■ Table 4.3

4.2 Handling Missing Values

In practice, data is often incomplete, that is, some feature values are missing in some samples. Taking medical diagnosis data as an example, feature values such as HIV test results could be unavailable due to privacy concerns. Sometimes we may have a large number of incomplete samples, especially when there are many features. Though we can simply discard the incomplete samples, it is a huge waste of data. For example, ■ Table 4.4 shows a watermelon data set with missing values. If we discard the incomplete samples, then we will have only four samples {4, 7, 14, 16} left for training. Apparently, we need a method to utilize incomplete samples.

■ **Tab. 4.4** The watermelon data set 2.0a

ID	color	root	sound	texture	umbilicus	surface	ripe
1	-	curly	muffled	clear	hollow	hard	true
2	dark	curly	dull	clear	hollow	-	true
3	dark	curly	-	clear	hollow	hard	true
4	green	curly	dull	clear	hollow	hard	true
5	-	curly	muffled	clear	hollow	hard	true
6	green	slightly curly	muffled	clear	-	soft	true
7	dark	slightly curly	muffled	slightly blurry	slightly hollow	soft	true
8	dark	slightly curly	muffled	-	slightly hollow	hard	true
9	dark	-	dull	slightly blurry	slightly hollow	hard	false
10	green	straight	crisp	-	flat	soft	false
11	light	straight	crisp	blurry	flat	-	false
12	light	curly	-	blurry	flat	soft	false
13	-	slightly curly	muffled	slightly blurry	hollow	hard	false
14	light	slightly curly	dull	slightly blurry	hollow	hard	false
15	dark	slightly curly	muffled	clear	-	soft	false
16	light	curly	muffled	blurry	flat	hard	false
17	green	-	dull	slightly blurry	slightly hollow	hard	false

Learning from incomplete samples raises two problems: (1) how to choose the splitting features when there are missing values? (2) how to split a sample with the splitting feature value missing?

Given a training set D and a feature a , let \tilde{D} be the subset of samples in D that has values of a . For problem (1), we can simply use \tilde{D} to evaluate a . Let $\{a^1, a^2, \dots, a^V\}$ denote the V possible values of a , \tilde{D}^v denote the subset of samples in \tilde{D} taking the value a^v , and \tilde{D}_k denote the subset of samples in \tilde{D} belonging to the k th class, where $k = 1, 2, \dots, |\mathcal{Y}|$. Then, we have $\tilde{D} = \bigcup_{k=1}^{|\mathcal{Y}|} \tilde{D}_k$ and $\tilde{D} = \bigcup_{v=1}^V \tilde{D}^v$. We assign a weight w_x to each sample x , and define

$$\rho = \frac{\sum_{x \in \tilde{D}} w_x}{\sum_{x \in D} w_x}, \quad (4.9)$$

$$\tilde{p}_k = \frac{\sum_{x \in \tilde{D}_k} w_x}{\sum_{x \in \tilde{D}} w_x} \quad (1 \leq k \leq |\mathcal{Y}|), \quad (4.10)$$

$$\tilde{r}_v = \frac{\sum_{x \in \tilde{D}^v} w_x}{\sum_{x \in \tilde{D}} w_x} \quad (1 \leq v \leq V). \quad (4.11)$$

Intuitively, for the feature a , ρ represents the proportion of samples without missing values, \tilde{p}_k represents the proportion of the k th class in all samples without missing values, and \tilde{r}_v represents the proportion of samples taking the feature value a^v in all samples without missing values. Then, we have $\sum_{k=1}^{|\mathcal{Y}|} \tilde{p}_k = 1$ and $\sum_{v=1}^V \tilde{r}_v = 1$.


With the above definitions, we extend the information gain (4.2) to

$$\begin{aligned} \text{Gain}(D, a) &= \rho \times \text{Gain}(\tilde{D}, a) \\ &= \rho \times \left(\text{Ent}(\tilde{D}) - \sum_{v=1}^V \tilde{r}_v \text{Ent}(\tilde{D}^v) \right), \end{aligned} \quad (4.12)$$

where, according to (4.1),

$$\text{Ent}(\tilde{D}) = - \sum_{k=1}^{|\mathcal{Y}|} \tilde{p}_k \log_2 \tilde{p}_k.$$

For problem (2), when the value of a is known, we place the sample \mathbf{x} into the corresponding child node without changing its weight $w_{\mathbf{x}}$. When the value of a is unknown, we place the sample \mathbf{x} into all child nodes, and set its weight in the child node of value a^v to $\tilde{r}_v \cdot w_{\mathbf{x}}$. In other words, we place the same sample into different child nodes with different probabilities.

The above solution is employed by the C4.5 algorithm (Quinlan 1993), and we will use it to construct a decision tree for  Table 4.4.

In the beginning, the root node includes all of the 17 samples in D , and all samples have the weight of 1. Taking color as an example, the set of samples without missing values of this feature includes 14 samples $\{2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17\}$, denoted by \tilde{D} . The entropy of \tilde{D} is calculated as

$$\begin{aligned} \text{Ent}(\tilde{D}) &= - \sum_{k=1}^2 \tilde{p}_k \log_2 \tilde{p}_k \\ &= - \left(\frac{6}{14} \log_2 \frac{6}{14} + \frac{8}{14} \log_2 \frac{8}{14} \right) = 0.985. \end{aligned}$$

Let \tilde{D}^1 , \tilde{D}^2 , and \tilde{D}^3 be the subsets of samples with color = green, color = dark, and color = light, respectively. Then, we have

$$\text{Ent}(\tilde{D}^1) = - \left(\frac{2}{4} \log_2 \frac{2}{4} + \frac{2}{4} \log_2 \frac{2}{4} \right) = 1.000,$$

$$\text{Ent}(\tilde{D}^2) = - \left(\frac{4}{6} \log_2 \frac{4}{6} + \frac{2}{6} \log_2 \frac{2}{6} \right) = 0.918,$$

$$\text{Ent}(\tilde{D}^3) = - \left(\frac{0}{4} \log_2 \frac{0}{4} + \frac{4}{4} \log_2 \frac{4}{4} \right) = 0.000.$$

The information gain of color for subset \tilde{D} is

$$\begin{aligned} \text{Gain}(\tilde{D}, \text{color}) &= \text{Ent}(\tilde{D}) - \sum_{v=1}^3 \tilde{r}_v \text{Ent}(\tilde{D}^v) \\ &= 0.985 - \left(\frac{4}{14} \times 1.000 + \frac{6}{14} \times 0.918 + \frac{4}{14} \times 0.000 \right) \\ &= 0.306. \end{aligned}$$

The information gain of color for data set D is

$$\text{Gain}(D, \text{color}) = \rho \times \text{Gain}(\tilde{D}, \text{color}) = \frac{14}{17} \times 0.306 = 0.252.$$

Similarly, we can calculate the information gain of all features for D :

$$\begin{aligned} \text{Gain}(D, \text{color}) &= 0.252; & \text{Gain}(D, \text{root}) &= 0.171; \\ \text{Gain}(D, \text{sound}) &= 0.252; & \text{Gain}(D, \text{texture}) &= 0.424; \\ \text{Gain}(D, \text{umbilicus}) &= 0.289; & \text{Gain}(D, \text{texture}) &= 0.006. \end{aligned}$$

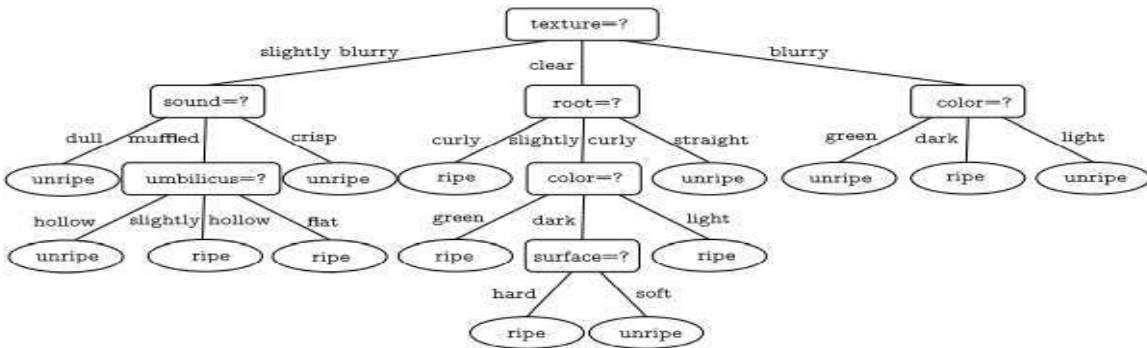


Fig. 4.8 The information gain-based decision tree generated from Table 4.4

Since splitting by texture has the largest information gain, it is selected for splitting the root node. Specifically, the samples {1, 2, 3, 4, 5, 6, 15} are placed into the child node of `texture = clear`, the samples {7, 9, 13, 14, 17} are placed into the child node `texture = slightly blurry`, and the samples {11, 12, 16} are placed into the child node of `texture = blurry`. The weights of these samples (i.e., 1) remain unchanged in the child nodes. However, since the value of `texture` is missing for the sample {8}, the sample is placed into all of the three child nodes with different weights: $\frac{7}{15}$, $\frac{5}{15}$, and $\frac{3}{15}$. The sample {10} is processed similarly. The splitting process proceeds recursively, and the final constructed decision tree is shown in ■ Figure 4.8.

CLUSTERING

1 Clustering Problem

Unsupervised learning aims to discover underlying properties and patterns from unlabeled training samples and lays the foundation for further data analysis. Among various unsupervised learning techniques, the most researched and applied one is *clustering*.

Clustering aims to partition a data set into disjoint subsets, where each subset is called a *cluster*. Through the partitioning, each cluster is potentially corresponding to a concept (category), such as “light green watermelon”, “dark green watermelon”, “seeded watermelon”, “seedless watermelon”, or even “locally grown watermelon” and “imported watermelon”. Note that clustering algorithms are unaware of such concepts before clustering and are only responsible for creating the clusters. The concept carried by each cluster, however, is interpreted by the user.

Formally, given a data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$ containing m unlabeled samples, where each sample $\mathbf{x}_i = (x_{i1}; x_{i2}; \dots; x_{in})$ is an n -dimensional vector. Then, a clustering algorithm partitions the data set D into k disjoint clusters $\{C_l \mid l = 1, 2, \dots, k\}$, where $C_{l'} \cap_{l' \neq l} C_l = \emptyset$ and $D = \bigcup_{l=1}^k C_l$. Accordingly, we denote $\lambda_j \in \{1, 2, \dots, k\}$ as the *cluster label* of sample \mathbf{x}_j (i.e., $\mathbf{x}_j \in C_{\lambda_j}$). Then, the clustering result can be represented as a cluster label vector $\boldsymbol{\lambda} = (\lambda_1; \lambda_2; \dots; \lambda_m)$ with m elements.

Clustering can be used by itself to identify the inherent structure of data, while it can also serve as a pre-processing technique for other learning tasks such as classification. For example, a business may want to classify new users into different “categories”, but this may not be easy. In such a case, we can use clustering to group all users into clusters, where each cluster represents a user category. Then, a classification model can be built upon the clusters for classifying the category of new users.