CHAPTER 3

Models Based on Decision Trees

Learning Objectives

At the end of this chapter, you will be able to:

- Describe decision trees and impurity measures
- Explain how decision trees are used in classification and regression
- Explain the bias–variance trade-off
- Describe predictive models based on decision trees
- Explain how AdaBoost, random forest and gradient boosting are used in prediction

3.1 Introduction to Decision Trees

Decision tree is a popular structure in ML. It is built by splitting the data set associated with a node into subsets that typically have less entropy or better purity. Splitting is carried out using the values of a selected feature; this feature-based decision-making at each of the internal nodes offers transparency and is ideally suited for explanation and for easier understanding by application domain experts.

Some properties of the decision tree (DT) building algorithm are:

- **It is greedy**: The best feature, in terms of purity in splitting, among the given set of features is invariably used at the root node of the DT. So, the feature used at each of the child nodes will depend on the feature selected at the parent node. Such a sequential greedy process may fail to be optimal on the whole.
- **It can be expensive**: Selection of a feature depends both on the number of features and the number of patterns. So, tests conducted on the feature values at each node need to be simple. Even simple tests can become computationally demanding when the dimensionality is large.
- **It can overfit**: The depth of the DT can increase as more splits are considered sequentially. Such a deep DT can overfit the training data and perform poorly on the validation/test data. A solution to this problem is provided by building the DT using the training data without worrying about depth and then pruning the more recent subtrees based on its performance on a validation data set. This process is called **pruning** and it helps in solving the overfitting problem.

The first two issues mentioned above are solved using a combination of classifiers. Some of the popular combinational ML models are:

- **Random Forest**: It is a collection of decision trees. The model predicts by combining the predictions of the trees in the collection.
- **AdaBoost**: It combines multiple weak learners to realize a good learning algorithm; a weak learner is a learner that provides more than 50% accuracy. One can use a variety of ML models as possible weak learners; however, simple DT-based weak learners are popular. Here, the combinational model employs weighted majority voting to realize the overall outcome.
- **Gradient Boost**: It is based on a sequence of ML models where each model is built to correct the error in the prediction of the previous model. Here also, it is possible to use any ML model in the sequence, but again, DT models are more popular. In this case, learnt models are used one after the other to predict the overall outcome.

These state-of-the-art models have become popular because they can deal with large-scale high-dimensional data sets that are common in current-day applications. We examine all these DT-based models in this chapter. The notion of overfitting is linked to a very fundamental concept in ML called the bias–variance trade-off. Simply stated, simple models tend to have higher bias while complex models tend to have higher variance.

3.2 Decision Trees for Classification

A decision tree is a tree with a root node and each child of the root node forms the root node of the subtree below it. We associate the entire training data set with the root node. The process of building the decision tree depends upon splitting the data set associated with a node.

A popular approach is to split, based on a feature, the set of patterns associated with a node so that the resulting subsets are least impure. These subsets are associated with the children nodes of the node. Impurity of the split at a node is captured using an impurity measure; the most popular one is **Shannon's entropy**. It is given by

$$-\sum_{i=1}^{C} p_i log(p_i),$$

where p_i is the probability of class i, $i = 1, 2, \cdots, C$ assuming that there are C classes. These probabilities are typically estimated based on the fraction of elements from each class among the set of patterns associated with the node.

Example 1: Consider a two-class problem, that is, $C = 2$. Consider the values of p_1 and p_2 corresponding to the 5 different cases shown in Table 3.1. The number of elements in the set associated with the node in each case is assumed to be 100.

Here, we use the fact that the limit of $plog(p)$is 0 as $p \rightarrow 0$ in cases 1 and 5 in the table where the entropy is 0. Some interesting observations from the table are:

- Entropy is non-negative; it is greater than or equal to zero. This is because $p_i \geq 0$ and $log(p_i) \leq 0$ for all i. So, $-log(p_i) \geq 0$ for all i. As a consequence, if there are C classes, then

$$-\sum_{i=1}^{C} p_i log(p_i) \geq 0$$

So, the minimum possible value is 0.

TABLE 3.1 Five different probability distributions and their entropy values

Case	Class 1 (p_1)	Class 2 (p_2)	Entropy
1	100 ($p_1 = 1.0$)	0 ($p_2 = 0.0$)	0.0
2	75($p_1 = 0.75$)	25 ($p_2 = 0.25$)	0.811
3	50 ($p_1 = 0.5$)	50 ($p_2 = 0.5$)	1.0
4	25 ($p_1 = 0.25$)	75($p_2 = 0.75$)	0.811
5	0($p_1 = 0.0$)	100 ($p_2 = 1.0$)	0.0

- In a two-class scenario, entropy is symmetric, that is

$$entropy(p_1, p_2) = entropy(p_2, p_1)$$

This is because

$$entropy(p_1, p_2) = -p_1 log(p_1) - p_2 log(p_2) = -p_2 log(p_2) - p_1 log(p_1) = entropy(p_2, p_1)$$

- In a two-class problem, entropy is maximum when $p_1 = p_2 = 0.5$. This can be explained by computing the partial derivatives of entropy and equating them to 0. The partial derivative of $entropy(p_1, p_2) = -p_1 log(p_1) - p_2 log(p_2)$ with respect to p_1 is $\frac{\partial entropy(p_1,p_2)}{\partial p_1} = -1 - log(p_1)$. Similarly, the partial derivative with respect to p_2 is $-1 - log(p_2)$. If we equate these two quantities to 0, then we get

$$-1 - log(p_1) = -1 - log(p_2) = 0 \Rightarrow log(p_1) = log(p_2) \Rightarrow p_1 = p_2$$

However, $p_1 + p_2 = 1$ and if $p_1 = p_2$, then $p_1 = p_2 = 0.5$.

- This corresponds to the maximum as $\frac{\partial^2 entropy(p_1,p_2)}{\partial^2 p_1}$ equals $\frac{-1}{p_1}$ and is negative as $p_1 \geq 0$; $\frac{\partial^2 entropy(p_1,p_2)}{\partial p_1 \partial p_2} = \frac{\partial^2 entropy(p_1,p_2)}{\partial p_2 \partial p_1} = 0$ and $\frac{\partial^2 entropy(p_1,p_2)}{\partial^2 p_2}$ equals $\frac{-1}{p_2}$ and is negative as $p_2 \geq 0$. The Hessian matrix, H, is

$$H = \begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix}$$

when $p_1 = p_2 = 0.5$ as $-\frac{1}{p_1} = -\frac{1}{p_2} = -2$.

- A matrix $A_{C \times c}$ is **negative definite** if $x^t A x < 0$ for any real vector x of size $C \times 1$. So, in this case, the Hessian matrix, H, is negative definite as

$$x^t H x = -2x(1)^2 - 2x(2)^2$$

- So, the value of $p_1 = p_2 = 0.5$ corresponds to the maximum.

Entropy depends on the probability distribution, the values of p_1 and p_2 in a two-class case. However, for the sake of brevity, we use entropy without arguments when the context is clear.

EXAMPLE 2: We will consider the example data set shown in Table 3.2 to examine the splitting process. We show the collection of these two-dimensional patterns in Fig. 3.1. The two classes are represented using $*$ (Class 1) and # (Class 2).

TABLE 3.2 A data set used to illustrate splitting

Pattern	Feature1	Feature2	Class
1	1	1	1
2	1	3	1
3	2	1	1
4	2	3	1
5	4	1	1
6	4	3	2
7	5	1	2
8	5	3	2

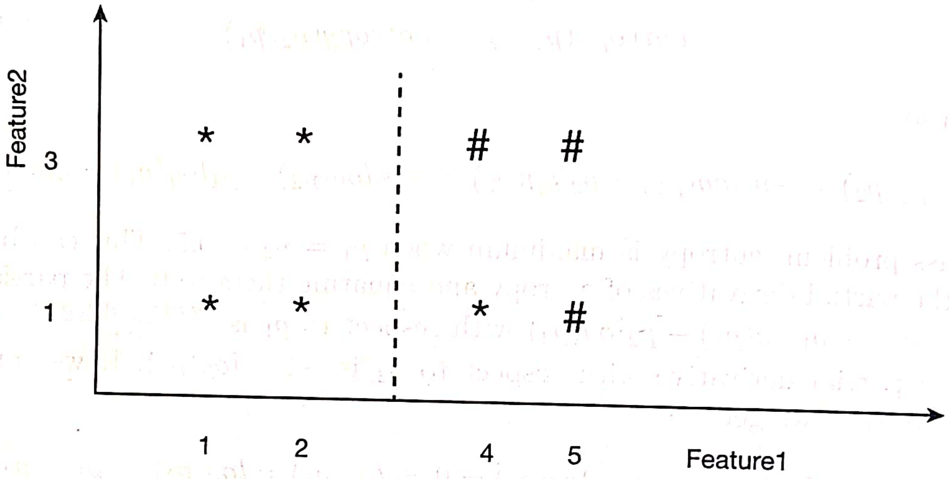


FIG. 3.1 Two-dimensional patterns of the data set given in Table 3.2

The entire collection of 8 two-dimensional patterns is associated with the root node. We split this set into subsets based on test or a splitting criterion that employs a single feature. We have only two features, *feature*1 and *feature*2, to select from.

If we use *feature*1, then one possible split is based on *feature*1 < 3 or not. The corresponding subsets are {1,2,3,4} and {5,6,7,8}. We compute the entropy of each subset and weigh it with the ratio of the size of the subset to the size of the set of its parent. The first subset is pure, that is, all the elements are from the same class (1). The weights are equal and are $\frac{4}{8} = \frac{1}{2}$ for each subset.

We use the proportions of classes as estimates for probabilities. In this example, $C = 2$. So, $p_1 = 1$ and $p_2 = 0$ for the first subset. Hence, entropy is

$$\frac{4}{4} log(1) + 0 log 0 = 0$$

So, for 4 out of 8 patterns the entropy is 0. Similarly the entropy for the second subset is

$$-\frac{1}{4} log\left(\frac{1}{4}\right) - \frac{3}{4} log\left(\frac{3}{4}\right) = \frac{1}{4} log(4) + \frac{3}{4} log\left(\frac{4}{3}\right) = 0.244219$$

We obtain the weighted value of the entropy of the split by using the weighted combination of the entropy values of the two subsets; weights are proportions of the number of elements in each subset. Here, each of the subsets has 4 elements out of the total of 8 elements present at their parent. So, the weights are equal to $\frac{4}{8} = \frac{1}{2}$. So, the weighted value of entropy impurity is

$$\frac{1}{2} \times 0 + \frac{1}{2} \times 0.244219 = 0.12211$$

So, total weighted entropy associated with the split is 0.12211.

It is possible to split the set based on other values of $feature1$. One possible split is based on $feature1 \leq 4$. The split will result in two subsets, the first set having 6 out of 8 patterns and the second set having the remaining 2 elements from the same class (2). So the entropy is 0.19567 and if we weigh it in proportion $\frac{6}{8}$, we get 0.146757; this is the weighted impurity associated with the split. This is larger than the entropy obtained for the earlier split, which was 0.12211. So, the best (minimum) value of entropy of splitting based on $feature1$ is 0.12211.

The other possibility is to consider splitting of $feature2$; there is only one possible split in this case. Using the test $feature2 \leq 1$ or not, we can split the data into two sets: {1,3,5,7} and {2,4,6,8}. The entropy of the first set is 0.244219 with a weight of $\frac{1}{2}$ (4 out of 8 patterns). The second set has entropy of 0.301 and the weight is $\frac{1}{2}$ (4 out of 8 patterns). So, the weighted impurity is

$$\frac{1}{2}(0.244219 + 0.301) = 0.2726$$

Observe that this value is larger than the 0.12211 obtained using $feature1$. So, $feature1$ is used at the root as it leads to purer subsets. This is depicted in Fig. 3.2.

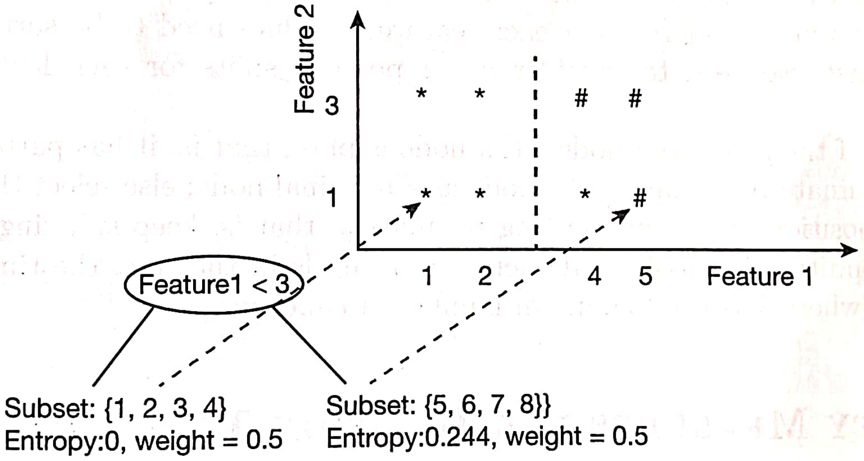


Fig. 3.2 The process of splitting at the root (for colour figure, please see Colour Plate 1)

Note that there could be several (infinite) ways to indicate the same split. For example, $feature1 > 1$, $feature1 > 1.5$ or even $feature1 \leq 2$ will all lead to the same split in terms of the resulting subsets, and so, all of them are equivalent.

The left child of the root is pure, that is, all the elements are from the same class, Class 1; so, there is no need to split it further as all the patterns are from the same class. However, the right child corresponding to the second subset is not pure. So, we need to split the set at the right child further. This process is recursive and goes on till some stopping criterion is satisfied. The final decision tree is shown in Fig. 3.3.

We depict leaf or terminal nodes using rectangles and non-terminal or internal nodes using ellipses in Fig. 3.3. Each non-terminal node is also called a decision node as there is a branching below the node based on the outcome of a test (satisfying a condition) associated with the non-terminal node.

These tests are typically binary and are used to split the set of patterns into two subsets. Note the difference in the entries used at the root node in Figs 3.1 and 3.3; in Fig. 3.1, the test is *Feature*1 < 3 and in Fig. 3.3, it is *Feature*1 < 4. Note that both yield the same subsets after splitting. This is done intentionally, to alert the reader that different tests could be equivalent. For example, *Feature*1 ≤ 2 is another equivalent test that yields the same split.

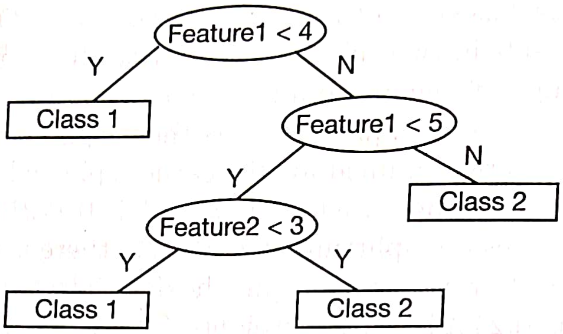


FIG. 3.3 Decision tree for the data set

The process of constructing a decision tree (DT) may be abstracted as follows:

1. Associate the entire training data set with the root and select the best feature and optimal splitting value for splitting. If the split leads to p subsets, then the root will have p children, each associated with a subset. If there are n l-dimensional patterns, then every one of the l features needs to be considered. For each feature, n values need to be sorted; for n patterns, in the worst case, we need to consider $n - 1$ possible splits for each feature. This requires $\mathcal{O}(n \log n)$ time.
2. Consider each of the p children nodes. If a node is pure, that is, it has patterns from only one class, then terminate by declaring the node as a terminal node; else select the best feature and optimal split position and keep splitting recursively, that is, keep splitting the nodes till the termination condition is satisfied. If there are m levels in the tree, the time required will be $\mathcal{O}(mkn \log n)$, where k is the maximum number of children.

3.3 IMPURITY MEASURES FOR DECISION TREE CONSTRUCTION

Even though Shannon's entropy is popular for capturing the impurity associated with a split, there are other impurity measures. Here, by p_i, we mean the probability of the i^{th} class. Some of the important ones are:

- **Gini Index:** The impurity at a node for a C class problem is defined as

$$1 - \sum_{i=1}^{C} p_i^2$$

- **Misclassification Impurity:** It is defined as

$$1 - \max_j \; p_j$$

Consider a node at which the associated set has 80% of patterns from Class 1, 15% from Class 2 and 5% from Class 3. So, $p_1 = 0.8$, $p_2 = 0.15$, $p_3 = 0.05$. Then, the misclassification impurity of the node is $1 - 0.8 = 0.2$. Note that it is the sum of the two smaller probabilities p_2 and p_3.

Note that entropy, Gini index and misclassification impurity are all impurity measures. They can be used to find the impurity of a node. We can also use them in computing the impurity associated with a split by considering the weighted impurity used in Example 2. In the same example, we used entropy as the impurity measure.

Example 3: We illustrate the Gini impurity measure and some of the important properties of the decision tree classifier (DTC) using the example data set shown in Table 3.3. There are three independent features: *Outlook*, *Temperature* in degrees Fahrenheit, and percentage *Humidity*. For the classes, there are two possibilities: P standing for Play and NP standing for No Play.

Table 3.3 A data set used to illustrate properties of decision trees

Pattern	Outlook	Temp(F)	Humidity	Class
1	Sunny	65	91	NP
2	Sunny	78	70	P
3	Overcast	85	82	P
4	Overcast	62	71	P
5	Rainy	70	65	NP
6	Rainy	79	62	P

If we use *Outlook* for splitting, based on its value, we will obtain three subsets: {1,2}, {3,4} and {5,6}. The second subset is pure $p_1 = 1$ and $p_2 = 0$ and the Gini index value for it is 0 $(1 - 1^2 - 0^2 = 0)$. In the cases of the first and third subsets, $p_1 = p_2 = 0.5$. So, the Gini index value for each of them is $1 - (0.5)^2 - (0.5)^2 = 0.5$ and each subset has 2 out of 6 elements. So, the weights are $\frac{2}{6}$ each.

So, the Gini impurity value for the whole split is $\frac{2}{6} \times 0.5 + \frac{2}{6} \times 0 + \frac{2}{6} \times 0.5 = \frac{1}{3}$. So, using *Outlook* for splitting, we get $\frac{1}{3}$ as the impurity value.

If we use *Humidity* for splitting and use the test *Humidity* < 91, then we get two subsets, {1} and {2,3,4,5,6}. Note that this is an optimal split using *Humidity* which will be considered in the exercises at the end of the chapter. The first subset {1} is pure and has 0 Gini index value. For the second set, $p_1 = \frac{4}{5}$ and $p_2 = \frac{1}{5}$. So, the Gini index value is $1 - (\frac{4}{5})^2 - (\frac{1}{5})^2 = \frac{8}{25}$ and the weight is $\frac{5}{6}$ as 5 out of 6 patterns are present in the second subset.

So, the weighted value is

$$\frac{1}{6} \times 0 + \frac{5}{6} \times \frac{8}{25} = \frac{4}{15}$$

If we perform the split based on *Temperature* > 70, then the two subsets we get are {2, 3, 6} and {1,4,5}. Note that the first subset is pure and its Gini index value is 0. For the second subset, $p_1 = \frac{1}{3}$ and $p_2 = \frac{2}{3}$ and the Gini value is $1 - \frac{4}{9} - \frac{1}{9} = \frac{4}{9}$. The weight for this subset is $\frac{3}{6}$ and the weighted value of impurity is $\frac{2}{9}$. This value is the smallest among all the three features considered. So, we choose *Temperature* at the root.

By using $Temperature > 70$ at the root, we obtain a pure subset {2,3,6} and the second subset {1,4,5} needs to be split further. Without getting into details, we have two options for splitting the set {1,4,5}. One is based on $Temperature > 62$ (Fig. 3.4 (a)) and other is based on *Outlook* (Fig. 3.4 (b)); both of them lead to the same impurity value of 0.

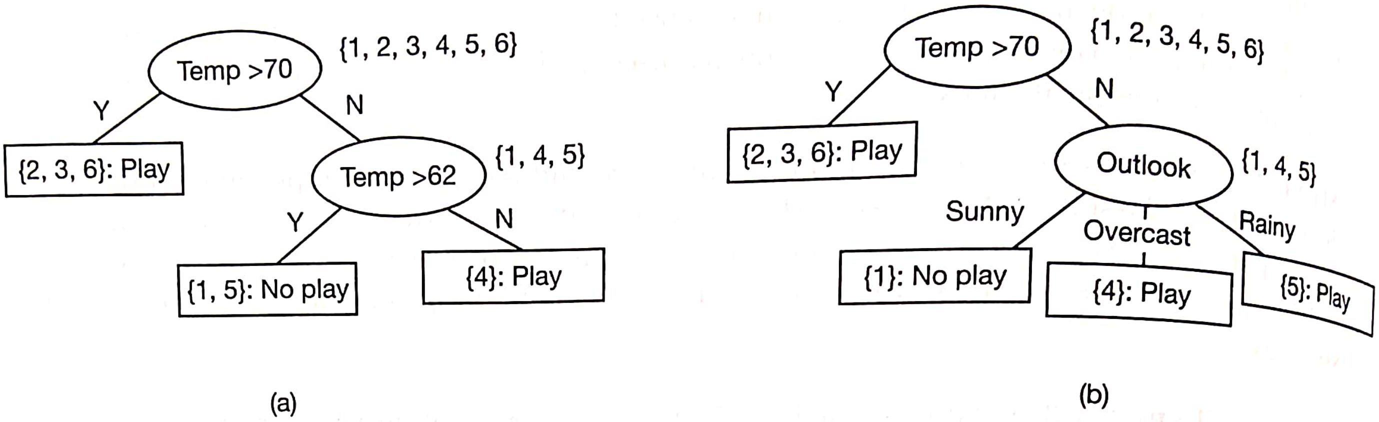


Fig. 3.4 Decision trees for the data set

We skipped some of the details involved in building the decision tree. They will be considered in some exercise problems at the end of the chapter.

3.4 Properties of the Decision Tree Classifier (DTC)

We have seen how to build decision trees from the given training data. We will now examine some important details in building decision trees and using them in classification.

1. **Splitting rule:** We have considered a simple test for splitting.

   a. Split is based on the values of a single feature. In Fig. 3.3, we have considered tests of the form $Feature1 < 4$. This is called an **axis-parallel split** and is depicted in Fig. 3.2 using a broken line that is parallel to the axis of $Feature2$. It is possible to use more complex rules for splitting. For example, a split rule of the form $20 \times Feature1 + 10 \times Feature2 > 100$ will split the data set shown in Table 3.2 into two subsets: {1,2,3,4,5} and {6,7,8}, leading to minimum impurity in these subsets. So, one test is adequate to build the decision tree (has only one node) as shown in Fig. 3.5. Such a split is called an **oblique split**. However, exploring all possible oblique splits is computationally expensive; it will require time that is exponential in the number of features. So, almost all of the practical implementations use axis-parallel splits.

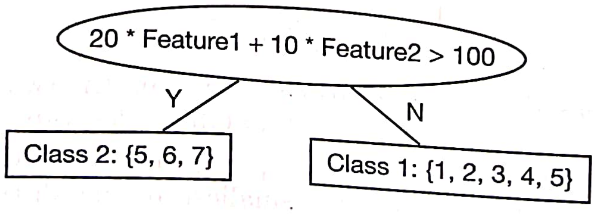


Fig. 3.5 Decision tree for the data set given in Table 3.2

b. We considered binary splits, that is, there will be two children for any node; a set of patterns is split into two subsets. Such binary splits are popular when the feature is numerical. However, for categorical features, it may be convenient to use non-binary splits where the number of children is more than 2. In Example 3, we considered such a categorical feature, *Outlook*, where the set of patterns is split into 3 subsets, as shown in Fig. 3.4.

2. **Criterion for splitting:** In Examples 1 and 2, we used the minimal impurity value to select the best feature for splitting. However, instead of minimizing impurity, it is possible to maximize the purity of a split by using **information gain.**

Example 4: Information gain (IG) at a node p in the tree is defined as

$$\text{IG(p)} = \text{Entropy(p)} - \text{Weighted Impurity(children(p))}$$

For the data in Example 1, we can compute IG at the root node for each of the three features as follows. The root is associated with the collection of all the 6 patterns, out of which 2 are from class NP and the remaining 4 are from P. So, $p_1 = \frac{4}{6}$ and $p_2 = \frac{2}{6}$. So, entropy at the root is

$$-\frac{2}{6}log\left(\frac{2}{6}\right) - \frac{4}{6}log\left(\frac{4}{6}\right) = 0.918$$

Now IG using each of the three features is:

- *Outlook*: The impurity at the root is 0.918 and the weighted impurity, by using *Outlook* for splitting, is $\frac{2}{6}(-\frac{1}{2}log(\frac{1}{2}) - \frac{1}{2}log(\frac{1}{2})) + \frac{2}{6}(-\frac{1}{2}log(\frac{1}{2}) - \frac{1}{2}log(\frac{1}{2})) = \frac{2}{3} = 0.6666$. So, IG is $0.918 - 0.6666 = 0.2514$.
- *Humidity:* Here, IG is $0.918 - 0.602 = 0.316$.
- *Temperature:* The IG value at the root using this feature is $0.918 - 0.459 = 0.459$.

Among the three features, *Temperature* has the maximum IG value of 0.459. So, we select *Temperature* > 70 for testing at the root. Note that selecting a split based on minimum weighted entropy is the same as selecting a split based on maximum IG at any node.

A problem with information gain is that it favours features that have more values. A correction is proposed in the form of **information gain ratio.** It is defined as

$$\frac{\text{IG}}{\text{Weighted Impurity}}$$

for each attribute at a node.

Example 5: So, the IG values for the three features considered in Example 2 are:

- *Outlook*: the ratio is $\frac{0.2514}{0.6666} = 0.1676$
- *Humidity*: the ratio is $\frac{0.316}{0.602} = 0.525$
- *Temperature*: the IG ratio is $\frac{0.459}{0.459} = 1.0$

So, we select *Temperature* based on the IG ratio as it has the maximum value among the three IG ratio values.

3. **Binary or Non-binary:** We have observed that we can split a set of patterns associated with a node into two or more subsets; each subset is associated with a child of the node under consideration. Consider the decision trees shown in Fig. 3.4. The entire set of patterns is associated with the root node. A subset {1,4,5} is associated with a child node of the root.

This subset is further split using *Temperature* > 62 into two subsets in Fig. 3.4 (a). Note that the resulting decision tree is a binary tree. However, in Fig. 3.4 (b), the subset {1,4,5} is split further into three subsets, each having a single element using the feature *Outlook*. So, this is not a binary tree. However, we can create a binary decision tree here by changing the test to *Outlook* = *Overcast*, as shown in Fig. 3.6.

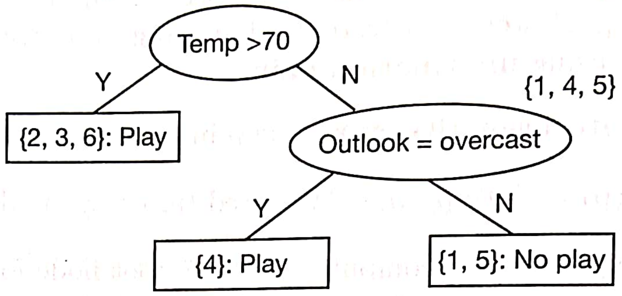


FIG. 3.6 Binary decision tree for the data set given in Fig. 3.4 (b)

4. **Termination condition:** We have seen in the examples that we stop splitting a node once the node is pure or has all the associated patterns from the same class. This is fine in the case of some of the examples we considered. However, in practical applications, it is advisable to permit a small percentage of impurity for termination. Some popular schemes could be:
   a. Terminate when the impurity level is less than, say, one or two percent. That is, out of 100 patterns associated with a node, 99 or 98 are from the same class and the remaining 1 or 2 are from different class/classes.
   b. Use a separate validation data set. Build the decision tree using the training data and check whether the performance of the tree is acceptable on the validation data.
5. **Class labels:** Each terminal node is represented by a rectangular box and it is associated with a class label. If the node is pure, then automatically the class label of the associated patterns is the class label. Consider for example, Fig. 3.6. Here, there are three terminal nodes, one at level 1 and two at level 2. The subset of patterns associated with the class label are depicted in the respective nodes. When an impurity is present, label the node using the class of the majority patterns associated with the node.
6. **Classification:** Given a test pattern, classification involves traversing the tree starting from the root and selecting the edge based on checking for the satisfiability of the condition at each node in the path till a leaf node is encountered. The class label associated with the leaf node is the class label to be assigned to the test pattern.
   For example, consider the patterns (*Outlook* = *Sunny*) and *Temperature* = 64 and *Humidity* = 72. Using the DT in Fig. 3.6, we take the right branch at the root node as *Temperature* > 70 is violated by the pattern. The next decision node checks for *Outlook* = *Overcast* and this is also violated by the pattern. So, we take the right branch at this decision node also and encounter a terminal node that is labelled *No Play*. So, the given pattern is classified as *No Play*.
7. **Transparency:** The path from the root to a leaf or terminal node, in a DT, may be viewed as a classification rule. For example, in the binary DT in Fig. 3.6, if we traverse from the root to the rightmost leaf node, we have a rule of the form

   If *Temperature* > 70 and *Outlook* $\neq$ *Overcast*, then *No Play*

So, a DT can be thought of as an abstraction of several classification rules and each such classification rule could be used to explain the reason behind classifying a pattern. Thus, DT is an excellent structure for explaining the reason behind a prediction.

8. **Handling mixed data types:** Decision tree classifiers are versatile in terms of dealing with both numerical and categorical features characterizing the data. Consider the data set shown in Table 3.3. Here, *Outlook* is categorical and *Temperature* and *Humidity* are numerical features. Using this data set, the decision tree in Fig. 3.6 is constructed. So, DTC can deal with mixed data types.
9. **Eliminating irrelevant features:** The DT automatically abstracts away features that are not relevant for classifying the data. Only relevant features are used. Consider the data set shown in Table 3.3. The feature *Humidity* is not a part of the tree in Fig. 3.6. For the same data set, the DT in Fig. 3.4 (a) employs only *Temperature*; the other two features, *Outlook* and *Humidity*, are irrelevant.
10. **Pruning a decision tree:** In some data sets, the decision tree can keep growing in height and size. Such a tree may overfit the training data; as a consequence, its performance on the testing data may be poor. In order to appreciate this problem, we use the DT shown in Fig. 3.7.

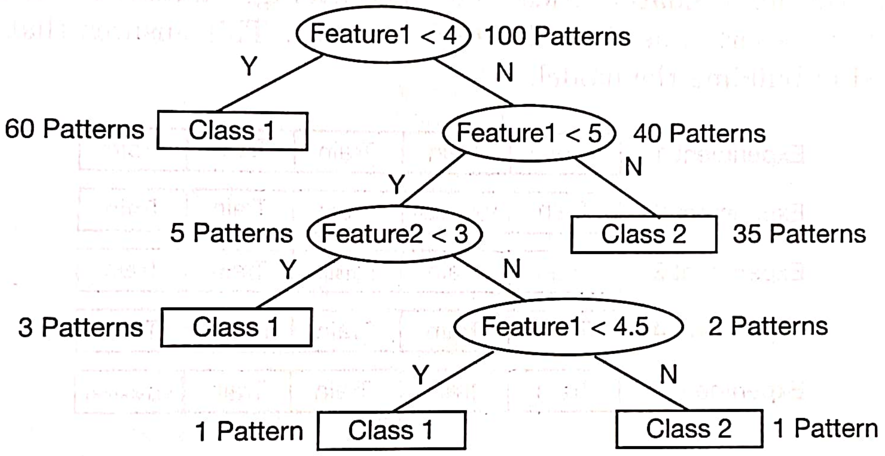


FIG. 3.7 Decision tree that can overfit

- Note that there are 100 training patterns that are associated with the root. Next to each node, non-terminal or terminal, we have indicated the number of patterns associated with the node. Note that there are 64 patterns from Class 1 and 36 patterns from Class 2.
- These are associated with different leaf nodes. The decision tree has 4 levels.
- Observe that split at the third level leading to two terminal nodes at the fourth level explains only two patterns and each resulting leaf node is associated with only one pattern.
- Such a tall tree can lead to overfitting. In some practical applications, the number of levels in the tree can be very large, it can be a few thousands. Some thumb rules say that the number of levels should be around $\log(l)$ where l is the number of features.

Pruning is a technique used to counter this overfitting problem. The solutions offered are:

a. Let the tree grow till the termination condition is satisfied. Now prune the leaf nodes that deal only with less than 3% of the data. In Fig. 3.7, the split based on $Feature1 < 4.5$ and the split based on $Feature2 < 3$ need not be performed; instead the non-terminal node corresponding to the test $Feature2 < 3$ can be converted into a leaf node and associated with Class 1 as 4 out of 5 patterns are from Class 1 (majority decision). This gives us the decision tree shown in Fig. 3.8.

We will see some more classifiers that have this property, and hence, they can be used in feature selection.

3.6 Regression Based on Decision Trees

Regression and classification are related predictive tasks. We have seen how the KNN-based scheme handles both classification and regression in the previous chapter. We will now examine the role of decision trees in regression.

Example 6: Let us consider the data set we have used to illustrate KNN regression in the previous chapter; we show it in Table 3.5. There are 9 two-dimensional patterns and the respective target values. We call the features f_1 and f_2.

Table 3.5 Data for decision tree-based regression

Number	f_1	f_2	Target (y)
1	0.2	0.4	8
2	0.4	0.2	8
3	0.6	0.4	12
4	0.8	0.6	16
5	1.0	0.7	19
6	0.8	0.4	14
7	0.6	0.2	10
8	0.5	0.5	12
9	0.2	0.6	10

In the case of DTC-based regression, a popular measure for splitting is the squared error (SE). Let a set S of n elements be split into sets S_1 and S_2 having n_1 and n_2 elements, respectively. Further, let the sample means of the target (y) values of the sets S_1 and S_2 be μ_1 and μ_2, respectively. Then the weighted squared error for the split is

$$\frac{n_1}{n}\sum_{x_i \in S_1, i=1}^{n_1}(y_i - \mu_1)^2 + \frac{n_2}{n}\sum_{x_i \in S_2, i=1}^{n_2}(y_i - \mu_2)^2$$

Note that

$$\mu_1 = \frac{1}{n_1}\sum_{x_i \in S_1, i=1}^{n_1} y_i \text{ and } \mu_2 = \frac{1}{n_2}\sum_{x_i \in S_2, i=1}^{n_2} y_i$$

For f_1, we need to consider splits based on $f_1 < 0.4$, $f_1 < 0.5$, $f_1 < 0.6$, $f_1 < 0.8$, $f_1 < 1.0$, $f_1 \leq 1.0$. The respective SEs are given in Table 3.6.

TABLE 3.6 Squared error (SE) values for possible splits

Test	SE of S_1	SE of S_2	Weighted SE
$f_1 < 0.4$	2	82	64.22
$f_1 < 0.5$	2.67	52.93	36.11
$f_1 < 0.6$	11	48.8	32.04
$f_1 < 0.8$	16	12.66	**14.9**
$f_1 < 1.0$	55.5	0	49.33
$f_1 \leq 1.0$	108	0	108

Note that the minimum value of weighted SE, that is, 14.9, is obtained for the split based on the test $f_1 < 0.8$. Subsets S_1 and S_2 for this optimal split are

$$S_1 = \{1, 2, 3, 7, 8, 9\} \text{ and } S_2 = \{4, 5, 6\}$$

Further,

$$\mu_1 = 10 \text{ and } \mu_2 = 16.33$$

It is possible to compute and show that the best split using f_2 is obtained for the test $f_2 < 0.4$; the corresponding weighted SE is 33.55, which is larger than the optimal value of 14.9 obtained using f_1. So, we use the test $f_1 < 0.8$ at the root. Proceeding further in splitting the children nodes, we obtain the DTC shown in Fig. 3.12. Note that for each leaf node, the subset of patterns along with the corresponding average target (y) value is indicated. For example, in the leftmost leaf node, the entry is {1,2,9}:8.66, where the subset indicates the collection of patterns 1, 2 and 9; their target values are 8, 8 and 10, respectively, and their average value is $\frac{8+8+10}{3} = 8.66$.

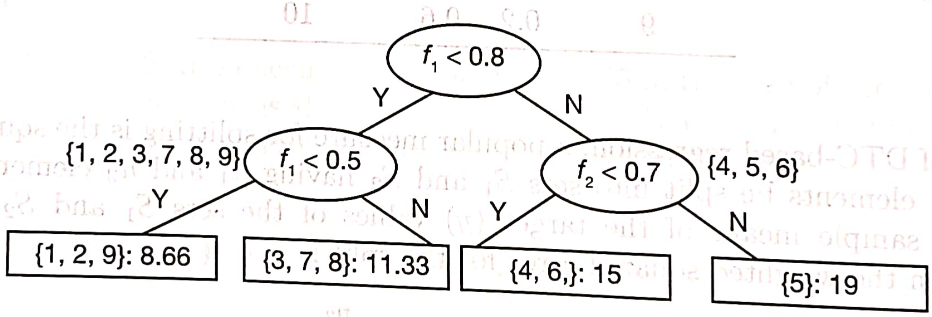


FIG. 3.12 Decision tree for the regression data set

Now if we are given a test pattern of the form (0.3, 0.4), then $f_1 = 0.3$ and $f_2 = 0.4$. So, we test at the root and go to its left child because $f_1 < 0.8$. Now we take the left child of the left child of the root node; we do this because $f_1 < 0.5$. So, we end up in the leftmost leaf node; this node is represented by its average value of 8.66.

We know based on the discussion in Example 15 in Chapter 2 that the true target value of (0.3, 0.4) is 9. So, the squared error is $(9 - 8.66)^2 = \frac{1}{9} = 0.1111$.

Note that for the KNN regression in Example 15 in Chapter 2, we predicted a value of 9.33 for the pattern (0.3, 0.4) which again means a squared error of 0.1111, of course, with a different prediction.

We can abstract the process of building a DT for regression as follows:

1. Associate the entire training data set with the root of the decision tree. Select the best feature and best split point based on the weighted SE value of the target (y) values.

2. Split the set of examples using the selected test (feature and split point) and associate the resulting subsets with the children nodes of the current node.
3. If a child node does not satisfy the termination condition, split it based on best test (feature and split selected) and repeat steps 2 and 3 till there is no need to split.

The termination condition checks for the purity of a node and decides to split if the impurity is above a threshold. Compute the mean value of the target for each leaf node and use it in prediction by associating this value with the leaf node. Given a test pattern, traverse the DT starting from the root and going down a path till a leaf node is encountered. The predicted value of the test pattern is the average target value of the leaf node.

3.7 BIAS–VARIANCE TRADE-OFF

Bias and variance are two important concepts in ML. We illustrate these notions using a simple example.

EXAMPLE 7: Consider the data shown in Table 3.7. There are four examples with their respective target values; consider only the first three columns in the table. Let us select the first 3 rows to train the model and the 4^{th} row for testing. Let us start with a simple linear model; it is a straight-line fit of the form $y = mx + c$, where m and c are the unknowns. The least-square fit for the first three points gives us the following two equations based on the two unknowns:

$$\sum_{i=1}^{3} y_i = 3c + m\sum_{i=1}^{3} x_i \Rightarrow 11 = 3c + 3m$$

and

$$\sum_{i=1}^{3} x_i y_i = c\sum_{i=1}^{3} x_i + m\sum_{i=1}^{3} x_i^2 \Rightarrow 17 = 3c + 5m$$

Solving these equations gives us the values of $m = 3$ and $c = \frac{2}{3}$. Now if we use these values, the estimated value of *target* for the test pattern with $x = \frac{1}{2}$ is $3 \times \frac{1}{2} + \frac{2}{3} = 2.16$. Note that the fourth column in the table corresponds to the straight-line fit given by y_{sl}.

TABLE 3.7 Polynomial fits to illustrate bias and variance

Example	x	Target (y)	y_{sl}	y_{p1}	y_{p2}
1	0	1	$\frac{2}{3}$	1	1
2	1	3	$\frac{11}{3}$	3	3
3	2	7	$\frac{20}{3}$	7	7
4	$\frac{1}{2}$	$\frac{7}{4}$	2.16	1.84	1.87

For this data of three points, we can also fit (overfit) a degree-3 polynomial. In general, we can have infinite polynomials of degree 3 that can fit the three training patterns. We consider two such polynomials that capture all the three training examples, that is, the first 3 rows, perfectly. The polynomials are

$$y_{p1} = 1 + \frac{3}{2}x + \frac{1}{4}x^2 + \frac{1}{4}x^3$$

and

$$y_{p2} = 1 + \frac{5}{3}x + \frac{1}{3}x^3$$

The values predicted using these two polynomials are shown in columns 5 and 6 of the table. Note that y_{p1} and y_{P2} give us the same values as the target y for the training examples. Observe that the 5^{th} and 6^{th} columns are identical and agree with column 3 on the three training examples. However, on the test point (4^{th} row in the table), they give different values: 1.84 and 1.87, respectively.

Now the basic question is which of these models is correct? Note that using three training points, we can fit a degree-2 polynomial uniquely. Starting with a quadratic of the form

$$y = ax^2 + bx + c$$

and plugging in the values of x and y from the first three rows of the table, we get $a = b = c = 1$.

So, the polynomial underlying the data is $1 + x + x^2$; use the 4^{th} row to check that this polynomial exactly satisfies the 4^{th} row also. So, the correct polynomial in this example is of degree-2. When we use the straight-line fit y_{sl}, we are underfitting the data as we get an error even on the training data, as can be seen from columns 3 and 4 of the table. This is a simpler model. Using the least-square fit, we obtain the same linear equation that was seen earlier; so the predicted value of the target is fixed when we use a straight-line fit and the least-square method to estimate the parameters.

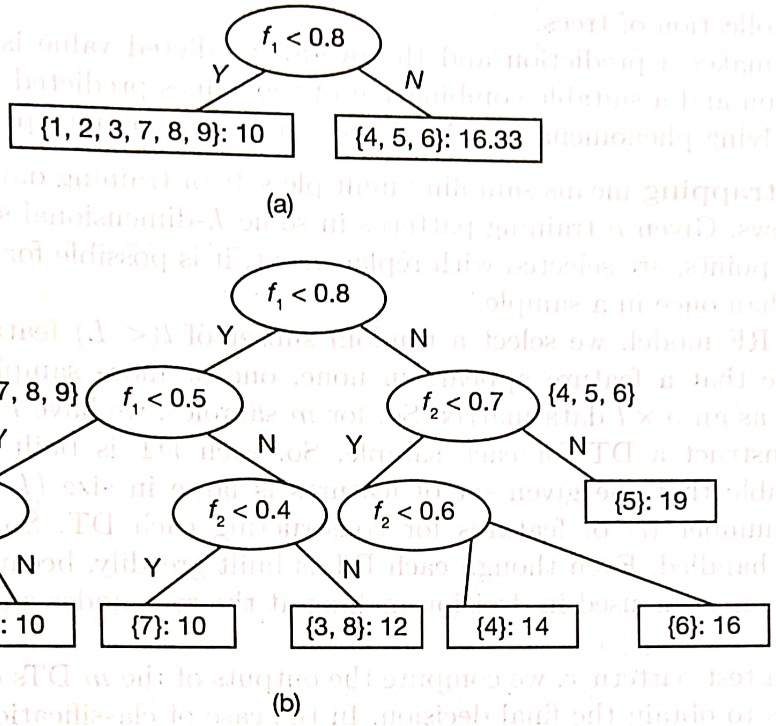
On the other hand, we can have infinite higher degree polynomials fitting the first three rows of the table. This is because a degree-3 polynomial fitting the three training examples is not unique; we can draw infinite degree-3 polynomials passing through any three points. Of course, there will be a unique degree-3 polynomial if we use four points.

We have considered two degree-3 polynomials which overfit the training data and can give different estimates for the target, as seen in columns 3, 5 and 6 of the fourth row of the table. Thus, there is non-zero variance among the target values estimated by the two degree-3 polynomial models, even though they successfully fit the training examples correctly.

So, a linear model underfits the training data; it may make mistakes even on the training data and there is no variance in the estimate of the target. This is because we have a single linear model fitting the data points using the least-squares and we predict a unique value, on the test pattern, using such a linear model. However, degree-3 polynomials overfit the training data and each polynomial gives a different target value estimate, leading to variance in the predicted values. So, simple models underfit; they have low variance and high bias as they fail to capture even the training data perfectly.

On the contrary, complex models overfit; they commit no errors on the training data and make different predictions on the test data, leading to low bias and high variance. This phenomenon goes under the name of **bias–variance trade-off**. Making the model simple affects bias and making the model complex affects the variance of the predictions. One needs to learn a model that strikes a balance.

We illustrate this notion further using the data used to construct the decision tree obtained in Fig. 3.12. Consider the decision trees in Fig. 3.13.



(a)

(b)

FIG. 3.13 Decision trees for the regression data set

The decision tree in Fig. 3.13 (a) is obtained by pruning the decision tree in Fig. 3.12 so as to have the terminal nodes at level 2. On the other hand, the DT in Fig. 3.13 (b) is permitted to have its leaf nodes till level 4. So, the DT in Fig. 3.13 (a) is simpler than the one in Fig. 3.12 and it underfits the training data. Whereas the DT in Fig. 3.13 (b) is more complex than the one in Fig. 3.12 and it overfits the training data.

For the test pattern (0.3, 0.4), we predicted the value to be 9.33 against the correct value of 9, leading to an SE value of 0.1111 using the decision tree in Fig. 3.12. However, the decision tree in Fig. 3.13 (a) predicts the target to be 10, whereas the DT in Fig. 3.13 (b) predicts the value to be 8. So, both of them make a larger error as the DT in Fig. 3.13 (a) is variance-friendly and the one in Fig. 3.13 (b) is bias-friendly; thus both of them make an error on the test data.

Bias–variance is the most important issue in ML model building. There is no clear cut solution to this problem. However, we can tune the parameters of the model, the number of levels and number of nodes in this case, so that the performance of the validation data set is acceptable. Note that pruning DTs is a mechanism to deal with the issue of bias–variance trade-off. It reduces variance.

3.8 RANDOM FORESTS FOR CLASSIFICATION AND REGRESSION

As mentioned earlier in the chapter, DT building can be expensive and the resulting tree may not be optimal because of the greedy nature of the building algorithm; these problems prevail significantly in high-dimensional data sets. So, DTCs are not typically used in applications dealing with high-dimensional data. One popular model used to deal with these problems is the random forest (RF). Its features are as follows:

- It employs a collection of randomly generated DTs in prediction. It is called a random forest as it is a collection of trees.
- Each DT makes a prediction and the overall predicted value is based on majority voting in classification and a suitable combination of the values predicted by the DTs.
- The underlying phenomena may be called bagging or bootstrap aggregation. Here,
  - **Bootstrapping** means sampling multiple sets of training data. In the case of RF, it works as follows. Given n training patterns in some L-dimensional space, m samples, each of size n data points, are selected with replacement; it is possible for the same data point to occur more than once in a sample.
  - In the RF model, we select a random subset of $l(< L)$ features in each sample; so, it is possible that a feature appears in none, one or more samples. So, each sample may be viewed as an $n \times l$ data matrix. So, for m samples, we have m such $n \times l$ data matrices.
  - We construct a DT for each sample. So, each DT is built based on only l features; it is possible that the given set of features is large in size (L is large), but we use only a small number (l) of features for constructing each DT. So, high-dimensional data sets can be handled. Even though each DT is built greedily, because of multiple DTs, different features may be used in decision making at the root nodes and other decision nodes of the m trees.
  - Given a test pattern x, we compute the outputs of the m DTs on x and **aggregate** these m outputs to obtain the final decision. In the case of classification, we take a majority of the m class labels and declare it as the class label of the RF model. In the case of regression, the output of the RF model is the average of all the m DT predictions.

3.8.1 Comparison of DT and RF Models on Olivetti Face Data

We compare the performance of the DT and RF models on the Olivetti Face data set. We show the results of using the DTC, by varying α, on the Olivetti Face data set in Fig. 3.14. Refer to the pruning scheme discussed in Section 3.4 for the use of α. Note that as the value of α increases,

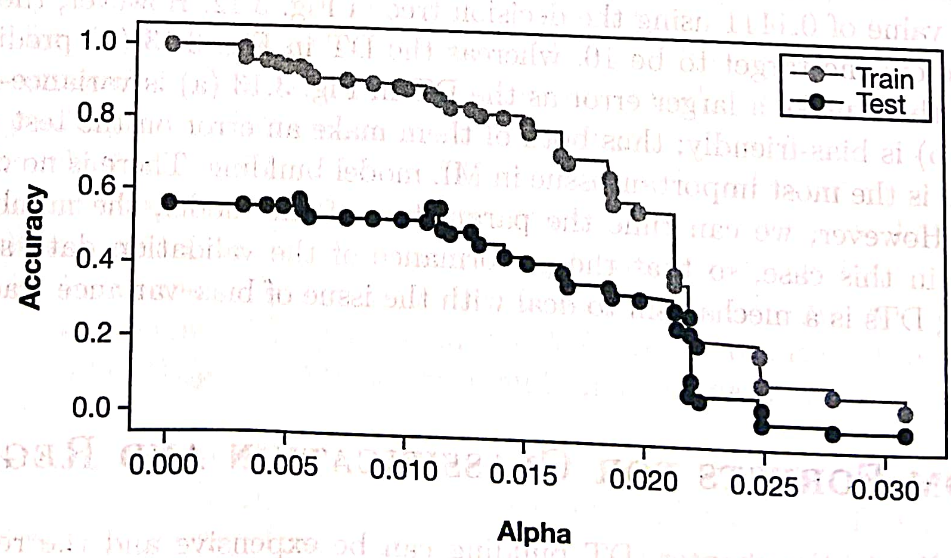


FIG. 3.14 Alpha versus accuracy on the Olivetti Face data set (for colour figure, please see Colour Plate 1)